Systematic Product Line Testing: Methodologies, Automation, and Industrial Application

Shuai Wang

Thesis submitted for the degree of Ph.D.

Department of Informatics
Faculty of Mathematics and Natural Sciences
University of Oslo
November 2014
Abstract

Testing products in a cost-efficient way remains an attractive topic for product line testing in both academia and industry, which can be addressed by employing systematic and automated approaches utilizing models such as feature models. Driven by the needs of our industrial problem for testing Video Conferencing Systems (VCSs) product line developed by Cisco Systems, Norway, cost-effective testing can be further formulated into four main problems, including: 1) Test Selection: Automatically and systematically select a set of relevant test cases for a product from the entire suite available for a product line; 2) Test Minimization: Minimizing the test suite obtained by the selection to eliminate redundant test cases for reducing the cost of testing (e.g., execution time) while preserving high effectiveness (e.g., fault detection capability); 3) Test Generation: Automatically and systemically generate test cases when new functionalities are introduced to the product line by the product; and 4) Test Prioritization: Prioritizing the minimized test suite with the aim at balancing the trade-off between cost and effectiveness. To tackle the above-mentioned four challenges, this thesis presents a set of methodologies on industrial systems for cost-effective testing of a product in a product line, namely Product Line Model-based Testing Methodologies (PL-MTM), which includes: 1) a systematic and automated test selection methodology using feature model; 2) an automated search-based test minimization approach; 3) an automated and systematic test generation methodology using feature model and 4) an automated search-based test prioritization approach.

For test selection, the contribution of this thesis is a systematic and automated test selection methodology using: 1) Feature Model for Testing (FM_T) to capture commonalities and variabilities of a product line; 2) Component Family Model for Testing (CFM_T) to model the structure of test case repository; 3) A tool to automatically build restrictions from CFM_T to FM_T and traces from CFM_T to the actual test cases. Using our methodology, a test engineer is only required to select relevant features through FM_T at a higher level of abstraction for a product and the corresponding test cases will be obtained automatically. We evaluate our methodology by applying it to a VCS product line called Saturn with seven commercial products and the results showed that our methodology can significantly reduce cost measured as test selection time and at the same time achieves higher effectiveness (feature coverage, feature pairwise coverage and fault detection) as compared with the current manual process. Moreover, we conducted a questionnaire-based study to solicit the views of test engineers who are involved in
developing FM_T and CFM_T. The results showed that test engineers are positive about adopting our methodology in their current practice. Furthermore, we conducted a thorough controlled experiment to further collect evidence about the benefits of our proposed automated methodology. The results showed that our methodology is cost-effective as compared with the manual approach.

For test minimization, the contribution of this thesis is a search-based test minimization technique. More specifically, the minimization problem is as a search problem and we defined a fitness function considering various optimization objectives based on the required cost-effective objectives (e.g., feature pairwise coverage and fault detection capability). To assess the performance of our fitness function, we conducted an extensive empirical evaluation by investigating the fitness function with three weight-based Genetic Algorithms (GAs) and seven multi-objective search algorithms using an industrial case study and 500 artificial problems inspired from the industrial case study. The results showed that Random-Weighted Genetic Algorithm (RWGA) significantly outperforms the other algorithms since RWGA can balance all the objectives together by dynamically updating weights at each generation. Based on the results of our empirical evaluation, we also implemented a tool called TEst Minimization using Search Algorithms (TEMSA) to support test minimization using various search algorithms in the context of product lines.

For test generation, the contribution of this thesis is an extended systematic and automated test generation methodology using FM_T and Component Family Model for Behaviors (CFM_B). FM_T captures testing functionalities of a product line, whereas CFM_B provides an abstraction layer on top of the configurable state machines. With our current methodology, a test engineer doesn’t need to acquire expertise on behavioral modeling and can simply configure models for a product by selecting features in FM_T and configuring provided attributes in CFM_B. The configured models are then given input to our model-based testing tool, TRansformation-based tool for Uml-baSed Testing (TRUST) for executable test case generation. We applied our extended methodology to a product line of video conferencing system developed by Cisco Systems, Norway. Results showed that the methodology significantly reduces the complexity of configuration; thereby significantly reducing required effort and cost (e.g., in terms of training). In addition, it does not require test engineers to have expertise in UML modelling, aspect-oriented modelling, and OCL specification and therefore eases the adoption in industry.

For test prioritization, the contribution of this thesis is a search-based multi-objective test prioritization technique, considering multiple cost and effectiveness measures. In
particular, our multi-objective optimization setup includes the minimization of execution cost (e.g., time), and the maximizing the number of prioritized test cases, feature pairwise coverage and fault detection capability. Based on cost-effectiveness measures, a novel fitness function is defined for such test prioritization problem. The fitness function is empirically evaluated together with three commonly used search algorithms (e.g., (1+1) Evolutionary algorithm (EA)) and Random Search as a comparison baseline based on the Cisco’s industrial case study and 500 artificial designed problems. The results showed that (1+1) EA achieves the best performance for solving the test prioritization problem and it scales up to solve the problems of varying complexity.
Acknowledgements

First and foremost, I would like to thank my supervisors Shaukat Ali and Arnaud Gotileb. Without their invaluable knowledge and expertise in software engineering research and unselfish guidance throughout my PhD, this work couldn’t be mature for this stage. Shaukat is an excellent supervisor who owns comprehensive knowledge and fanatical passion, which inspire me to conduct high quality research throughout the entire process of my PhD. He has provided me deep and insightful training on many perspectives, such as research methods, technical implementation and various experiment evaluations. Arnaud is a prominent researcher and supervisor and his thoughtful knowledge about constraint programing, optimization problem solving guided me to perform meaningful and useful research rather than useless paper work. I would also like to thank Simula Research Laboratory (SRL) and Simula School of Research and Innovation (SSRI) for funding my PhD and providing an excellent international research environment with a lot of multicultural interactions.

I would like to thank all the group members at Certus, especially for the senior research scientist Tao Yue. We have had a lot of interesting and valuable discussions and her rich knowledge related with empirical study and model-driven engineering provided me brilliant inspiration and largely improved the quality of this thesis. I also would like to thank all the other group members in software engineering department, who provided me a great environment for conducting my research.

I also want to thank Marius Liaaen and his team members from Cisco. We together identified very interesting and meaningful testing topics related with real industrial practice for my thesis. Moreover, a valuable case study was provided by his testing team for the proposed solutions in this thesis, which enabled the successful evaluation.

I would like to specially thank my girlfriend Ruixia Huang, for her unconditional support in everything during the entire stage of my PhD. Last but not least, I would also like to thank my parents for their love, concern, support and thoughtful care.
List of papers

The following papers are included in this thesis:

**Paper 1. A Systematic Test Case Selection Methodology for Product Lines: Results and Insights from an Industrial Case Study**


Accepted for a publication in the Journal of Empirical Software Engineering (EMSE).

DOI: 10.1007/s10664-014-9345-5

**Paper 2. Automated Product Line Test Case Selection: Industrial Case Study and Controlled Experiment**


Invited journal paper that is submitted to the Journal of Software and Systems Modeling (SOSYM).


S. Wang, S. Ali and A. Gotlieb.


DOI: 10.1016/j.jss.2014.08.024

**Paper 4. Random-Weighted Search-Based Multi-Objective Optimization Revisited**

S. Wang, S. Ali and A. Gotlieb.


**Paper 5. Using Feature Model to Support Model-Based Testing of Product Lines: An Industrial Case Study**


In: Proceedings of 18th International Software Product Line Conference (SPLC 2014),

The six papers are self-contained and thus some information might be redundant across the papers. Different abbreviations may have been used in the papers.

My contributions
For all the above-mentioned papers, I was the main contributor for the idea, implementation, and case study design and application. My supervisors were involved throughout all phases of the work. For paper 1, I was the main contributor for designing the methodology followed by applying to an industrial case study and conducting a questionnaire-based study. For paper 2, I was responsible for conducting the thorough controlled experiment with the help of my supervisors and Marius Liaaen (who is a QA manager at Cisco). For paper 3, I was the main contributor for proposing the search-based techniques and evaluating the techniques using an industrial case study and 500 artificial problems. For paper 4, I was mainly responsible for proposing the new weight assignment strategy together with my supervisors and performing an empirical study based on an industrial case study and artificial problems. For paper 5, I took the main role for designing the methodology for test generation and conducting an industrial case study with the help of Tao Yue (who is a senior research scientist in Simula) and Marius. For paper 6, I was the main contributor for designing and developing the search-based techniques for test prioritization. David Buchmann (who is a test engineer at Cisco) and Marius helped to refine the proposed technique and perform the evaluation. Dipesh Pradhan (who is a master student in Simula) was involved into the implementation of the search techniques.

In addition, during my PhD study, I also contributed in other papers, which are not included in this thesis. Paper 7 (which got Best Application Paper award) is not included since the journal versions of the paper (Paper 1 and Paper 2) are included in this thesis. Paper 8 is excluded since its journal version (Paper 3) is also included in the thesis. Paper 9, 10 and 11 (9 and 10 are workshop papers and 11 is a poster paper) are not included into
the thesis because their contributions have been covered in the above-listed journal and conference papers.

**Paper 7. Automated Test Case Selection using Feature Model: An Industrial Case Study**

S. Wang, S. Ali, A. Gotlieb and M. Liaaen


**Paper 8. Minimizing Test Suites in Software Product Lines using Weight-Based Genetic Algorithms**

S. Wang, S. Ali and A. Gotlieb


**Paper 9. Automatic Selection of Test Execution Plans from a Video Conferencing System Product Line**

S. Wang, A. Gotlieb, M. Liaaen and L.C. Briand


S. Wang and S. Ali


**Paper 11. Automated Product Line Methodologies to Support Model-Based Testing**

S. Wang, S. Ali and A. Gotlieb

Contents

Summary........................................................................................................................................1
1 Introduction..................................................................................................................................1
2 Background..................................................................................................................................4
  2.1 Modelling..................................................................................................................................4
    2.1.1 Feature Model..................................................................................................................4
    2.1.2 Component Family Model...............................................................................................5
  2.2 Search-based Software Testing (SBST).................................................................................6
3 Product Line Testing Methodologies.........................................................................................8
  3.1 Test Selection Methodology using Feature Model...............................................................8
  3.2 Search-Based Test Minimization Approach.........................................................................10
  3.3 Test Generation Methodology using Feature Model..........................................................12
  3.4 Search-Based Test Prioritization Approach.........................................................................13
4 Research Method.......................................................................................................................15
  4.1 Problem Identification..........................................................................................................15
  4.2 Problem Formulation............................................................................................................16
  4.3 Solution Realization.............................................................................................................16
  4.4 Solution Evaluation...............................................................................................................17
5 Summary of Results..................................................................................................................18
  5.1 Paper 1....................................................................................................................................18
  5.2 Paper 2....................................................................................................................................19
  5.3 Paper 3....................................................................................................................................20
  5.4 Paper 4....................................................................................................................................21
  5.5 Paper 5....................................................................................................................................22
  5.6 Paper 6....................................................................................................................................23
6 Future Directions........................................................................................................................24
7 Conclusion....................................................................................................................................25
8 References for the Summary.......................................................................................................27

Paper 1: A Systematic Test Case Selection Methodology for Product Lines: Results and Insights from an Industrial Case Study ..........................................................30
  1 Introduction..............................................................................................................................31
  2 Background..............................................................................................................................33
    2.1 Feature Model......................................................................................................................33
    2.2 Component Family Model.................................................................................................35
  3 Research Method.....................................................................................................................36
    3.1 Problem Identification..........................................................................................................37
    3.2 Problem Formulation...........................................................................................................38
    3.3 Solution Realization.............................................................................................................42
    3.4 Solution Evaluation.............................................................................................................43
  4 Methodology............................................................................................................................43
    4.1 Running Example..................................................................................................................44
    4.2 Feature Model for Testing (FM_T).......................................................................................44
    4.3 Component Family Model for Testing (CFM_T)...............................................................47
    4.4 Process to Select Test Cases for a Product.........................................................................49
    4.5 Consistency Between Different Artefacts..........................................................................50
  5 Automation...............................................................................................................................53
  6 Evaluation................................................................................................................................55
    6.1 Industrial Case Study............................................................................................................55
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.2</td>
<td>Questionnaire-Based Study</td>
<td>61</td>
</tr>
<tr>
<td>6.3</td>
<td>Threats to Validity</td>
<td>65</td>
</tr>
<tr>
<td>7</td>
<td>Lessons Learnt</td>
<td>67</td>
</tr>
<tr>
<td>8</td>
<td>Related Works</td>
<td>71</td>
</tr>
<tr>
<td>8.1</td>
<td>Product Line Testing</td>
<td>71</td>
</tr>
<tr>
<td>8.2</td>
<td>Feature Modeling</td>
<td>73</td>
</tr>
<tr>
<td>8.3</td>
<td>Consistency Checking</td>
<td>74</td>
</tr>
<tr>
<td>8.4</td>
<td>Regression Testing</td>
<td>75</td>
</tr>
<tr>
<td>9</td>
<td>Conclusion and Future Work</td>
<td>75</td>
</tr>
<tr>
<td>Paper 2: Automated Product Line Test Case Selection: Industrial Case Study and Controlled Experiment</td>
<td>80</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Introduction</td>
<td>81</td>
</tr>
<tr>
<td>2</td>
<td>Background</td>
<td>83</td>
</tr>
<tr>
<td>2.1</td>
<td>Feature Model</td>
<td>83</td>
</tr>
<tr>
<td>2.2</td>
<td>Component Family Model</td>
<td>84</td>
</tr>
<tr>
<td>3</td>
<td>Running Example</td>
<td>84</td>
</tr>
<tr>
<td>4</td>
<td>Methodology</td>
<td>85</td>
</tr>
<tr>
<td>4.1</td>
<td>Feature Model for Testing (FM_T)</td>
<td>86</td>
</tr>
<tr>
<td>4.1.1</td>
<td>Modeling Testing Functionalities using FM_T</td>
<td>86</td>
</tr>
<tr>
<td>4.1.2</td>
<td>Modeling Relations using FM_T</td>
<td>87</td>
</tr>
<tr>
<td>4.1.3</td>
<td>Summary for FM_T</td>
<td>88</td>
</tr>
<tr>
<td>4.2</td>
<td>Component Family Model for Testing (CFM_T)</td>
<td>88</td>
</tr>
<tr>
<td>4.2.1</td>
<td>Modeling Test Structure using CFM_T</td>
<td>88</td>
</tr>
<tr>
<td>4.2.2</td>
<td>Linking FM_T and CFM_T using Restrictions</td>
<td>90</td>
</tr>
<tr>
<td>4.2.3</td>
<td>Summary for CFM_T</td>
<td>90</td>
</tr>
<tr>
<td>4.3</td>
<td>Process to Select Test Cases for a Product</td>
<td>90</td>
</tr>
<tr>
<td>Paper 2: Automated Product Line Test Case Selection: Industrial Case Study and Controlled Experiment</td>
<td>91</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Automation</td>
<td>91</td>
</tr>
<tr>
<td>6</td>
<td>Evaluation for Industrial Case Study</td>
<td>92</td>
</tr>
<tr>
<td>6.1</td>
<td>Industrial Case Study</td>
<td>93</td>
</tr>
<tr>
<td>6.1.1</td>
<td>Abstraction and Automation</td>
<td>94</td>
</tr>
<tr>
<td>6.1.2</td>
<td>Reduced Selection Effort and Test Coverage</td>
<td>94</td>
</tr>
<tr>
<td>6.1.3</td>
<td>Less Reliance on Domain Expertise</td>
<td>95</td>
</tr>
<tr>
<td>6.1.4</td>
<td>Reduced Maintenance Effort</td>
<td>95</td>
</tr>
<tr>
<td>6.1.5</td>
<td>Adoption in Other Contexts</td>
<td>96</td>
</tr>
<tr>
<td>6.1.6</td>
<td>Limitations of the Methodology</td>
<td>96</td>
</tr>
<tr>
<td>6.2</td>
<td>Questionnaire-Based Study</td>
<td>96</td>
</tr>
<tr>
<td>6.2.1</td>
<td>Planning and Design</td>
<td>97</td>
</tr>
<tr>
<td>6.2.2</td>
<td>Results and Analysis for FM_T</td>
<td>97</td>
</tr>
<tr>
<td>6.2.3</td>
<td>Results and Analysis for CFM_T</td>
<td>98</td>
</tr>
<tr>
<td>6.2.4</td>
<td>Threats to Validity</td>
<td>99</td>
</tr>
<tr>
<td>Paper 2: Automated Product Line Test Case Selection: Industrial Case Study and Controlled Experiment</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Evaluation for Controlled Experiment</td>
<td>100</td>
</tr>
<tr>
<td>7.1</td>
<td>Experiment Planning</td>
<td>100</td>
</tr>
<tr>
<td>7.1.1</td>
<td>Goal, Research Questions and Hypotheses</td>
<td>101</td>
</tr>
<tr>
<td>7.1.2</td>
<td>Participants</td>
<td>101</td>
</tr>
<tr>
<td>7.1.3</td>
<td>Case Study System and Material</td>
<td>102</td>
</tr>
<tr>
<td>7.1.4</td>
<td>Variables</td>
<td>102</td>
</tr>
<tr>
<td>7.1.5</td>
<td>Training</td>
<td>103</td>
</tr>
<tr>
<td>7.1.6</td>
<td>Experiment Design</td>
<td>103</td>
</tr>
<tr>
<td>7.1.7</td>
<td>Overview of Statistical Tests</td>
<td>104</td>
</tr>
<tr>
<td>7.2</td>
<td>Results and Discussions</td>
<td>105</td>
</tr>
<tr>
<td>7.2.1</td>
<td>Results and Analysis for RQ1</td>
<td>105</td>
</tr>
<tr>
<td>7.2.2</td>
<td>Results and Analysis for RQ2</td>
<td>106</td>
</tr>
</tbody>
</table>
Paper 3: Cost-Effective Test Suite Minimization in Product Lines Using Search Techniques

1. Introduction ........................................................................................................ 125
2. Description of the Selected Algorithms ............................................................... 127
   2.1 Overview of Search Algorithms ....................................................................... 128
   2.2 Evolutionary Algorithms (EAs) ....................................................................... 129
   2.3 Swarm Algorithm ............................................................................................ 131
   2.4 Hybrid Algorithm ........................................................................................... 131
   2.5 Stochastic Algorithm ...................................................................................... 131
3. Problem Representation and Fitness Function .................................................... 131
   3.1 Problem Representation .................................................................................. 132
      3.1.1 Basic Concepts ......................................................................................... 132
      3.1.2 Cost/Effectiveness Measures .................................................................... 132
      3.1.3 Problem Representation ........................................................................... 133
   3.2 Definitions and Functions for Effectiveness/Cost Measures ......................... 133
      3.2.1 Effectiveness Measures ........................................................................... 133
      3.2.2 Cost Measures ......................................................................................... 136
   3.3 Fitness Function ............................................................................................... 137
4. Industrial Case Study and Artificial Problems ................................................... 138
5. Empirical Evaluation ............................................................................................ 139
   5.1 Experiment Design ......................................................................................... 139
      5.1.1 Research Questions .................................................................................. 139
      5.1.2 Selected Criteria for the Algorithms and Parameter Settings .................. 140
      5.1.3 Evaluation Mechanisms .......................................................................... 141
   5.2 Statistical Tests ............................................................................................... 142
   5.3 Experiment Execution ..................................................................................... 143
6. Results and Analysis ........................................................................................... 144
   6.1 Results and Analysis for the Selected Weight-Based GAs ............................... 144
      6.1.1 Results and Analysis for Industrial Case Study ....................................... 144
      6.1.2 Results and Analysis for Artificial Problems .......................................... 148
      6.1.3 Timing and Analysis for Weight-based GAs .......................................... 150
   6.2 Results and Analysis for the Best Weight-Based GAs (RWGA) and the Other Selected Search Algorithms ................................................... 150
      6.2.1 Results and Analysis for Industrial Case Study ....................................... 150
      6.2.2 Results and Analysis for Artificial Problems .......................................... 157
      6.2.3 Timing and Analysis for RWGA and the Other Selected Search Algorithms 160
7. Discussions ......................................................................................................... 160
   7.1 Discussions Related to RQ1-RQ3 .................................................................... 161
   7.2 Discussions Related to RQ4 ............................................................................ 162
Paper 4: Random-Weighted Search-Based Multi-Objective Optimization Revisited ............ 178
1. Introduction ........................................................................ 178
2. Background ..................................................................... 180
3. Uniformly Distributed Weights ........................................ 181
4. Case Studies ..................................................................... 183
5. Empirical Evaluation ....................................................... 185
   5.1 Experiment Design ......................................................... 185
   5.2 Statistical Tests ............................................................. 186
   5.3 Results and Analysis ...................................................... 187
   5.4 Overall Discussion ......................................................... 192
6. Threats to Validity ............................................................ 194
7. Related Works .................................................................. 195
8. Conclusion and Future Work ............................................ 196

Paper 5: Using Feature Model to Support Model-Based Testing of Product Lines: An Industrial Case Study ............................................................... 198
1 Introduction ...................................................................... 199
2 Background ..................................................................... 201
   2.1 Aspect State Machine (AspectSM) ... .............................. 201
   2.2 Feature Model (FM) and Component Family Model (CFM) ... 201
   2.3 Prior Modeling and Configuration Methodology ... ............ 202
3 Running Example ............................................................. 203
4 Methodology .................................................................... 204
   4.1 Notations ................................................................... 204
   4.2 State Machine Variability .............................................. 206
   4.3 State Machine Model Element Variability ....................... 207
   4.4 Class Attribute Value Variability ..................................... 209
   4.5 Class Attributes Selection Variability ............................... 210
   4.6 Consistency between Different Artifacts ......................... 211
5 Automation ...................................................................... 212
6 Evaluation ...................................................................... 214
   6.1 Comparison with our Previous Work .................... ............. 214
   6.2 Industrial Case Study ...................................................... 215
7 Related Works .................................................................. 216
8 Conclusion and Future Work ............................................ 218

Paper 6: Multi-Objective Test Prioritization in Software Product Line Testing: An Industrial Case Study ............................................................... 221
1 Introduction ...................................................................... 222
2 Background ..................................................................... 224
3 Test Prioritization Problem Description and Representation .... 225
   3.1 Test Prioritization Problem in Industry ......................... 225
Summary

1 Introduction

Product line engineering (PLE) is a systematic process to capture commonalities and variability across a set of products belonging to a product line [1, 2]. PLE has demonstrated several benefits in both academia and industry including: reducing development time and cost, speeding up product time-to-market and reducing required modelling effort for Model-based Testing (MBT) through the mechanism of reuse [3, 4]. By employing PLE, cost-effective testing products in a product line can be classified into four main problems; 1) Test Selection: Automatically and systemically select a subset of test cases including a set of relevant test cases for a product from the entire suite available for a product line; 2) Test Minimization: Minimizing the test suite obtained by the selection to eliminate redundant test cases for reducing the cost of testing (e.g., execution time) while preserving high effectiveness (e.g., fault detection capability); 3) Test Generation: Automatically and systemically generating test cases when new functionalities are introduced to the product line by the product and 4) Test Prioritization: Prioritizing the minimized test suite by balancing the trade-off between cost and effectiveness.

Driven by the needs of our industrial problems of testing Video Conferencing Systems (VCSs) product line developed by Cisco, Norway, this thesis proposes a set of methodologies to address the above-mentioned problems, for cost-effective testing of a product, namely Product Line Model-based Testing Methodologies (PL-MTM). More
specifically, PL-MTM mainly includes four activities to tackle the above four problems respectively.

For the **Test Selection** problem, a product line modelling methodology is proposed for automated test case selection with the aims of reducing selection effort at the same time covering all required test functionalities. The methodology consists of the following main parts: 1) defining a Feature Model for Testing (FM_T) to model a product line for testing; 2) defining a Component Family Model for Testing (CFM_T) to model the structure of test cases in the repository; and 3) linking CFM_T and FM_T via restrictions. With our methodology, test engineers only need to perform selection of features in FM_T and the related test cases can be chosen automatically from the repository.

For the **Test Minimization** problem, the thesis proposes and evaluates a fitness function in conjunction with ten multi-objective search algorithms, to minimize test suites in the context of product line testing. The fitness function takes effectiveness measures (i.e., Test Minimization Percentage (TMP), Feature Pairwise Coverage (FPC), Fault Detection Capability (FDC) and Average Execution Frequency (AEF)) and cost measures (i.e., Overall Execution Time (OET)) into account to guide the search. The search algorithms, along with our fitness function, are evaluated on an industrial case study and 500 artificial problems of varying size and complexity using a tool we have developed for this purpose, called TEst Minimization with Search Algorithms (TEMSA).

For the **Test Generation** problem, the thesis proposes an extension to our previous product line modelling and configuration methodology [3] to support Model-Based Testing (MBT) with the aims of reducing modelling effort and eliminating the needs to acquire expertise of UML modelling, Aspect-Oriented modelling (AOM) and Object Constraint Language (OCL). The extended methodology includes the following main steps: 1) reusing the existing Feature Model for Testing (FM_T) to model testing functionalities of a product line; 2) defining a Component Family Model for Behaviors (CFM_B) to associate its configurable parameters to model elements of behavioral models and related OCL constraints; 3) linking CFM_B with FM_T via restrictions. With our methodology based on FM_T and CFM_B, various types of behavioral variability can be configured by feature selection and attribute configuration in FM_T. The configured models are then given input to our model-based testing tool, TRansformation-based tool for Uml-baSed Testing (TRUST) [34] for executable test case generation.
For the Test Prioritization Problem, the thesis introduces a search-based multi-objective test prioritization technique. More specifically, cost and effectiveness measures are first defined with respect to the required objectives for prioritization. One cost measure and three effectiveness measures are carefully defined based on the expertise of VCS testing, i.e., Overall Execution Cost (OEC), Prioritized Extent of test cases (PE), Feature Pairwise Coverage (FPC) and Fault Detection Capability (FDC). A novel fitness function is further defined considering all these cost/effectiveness measures and integrated with three commonly used search algorithms (Alternating Variable Method (AVM), Genetic Algorithm (GA) and (1+1) Evolutionary Algorithm (EA)) and Random Search (RS) as a comparison baseline. When particular products need to be tested, after obtaining the relevant test cases using the proposed selection and minimization techniques, test engineers can further cost-effectively prioritize the selected test cases for finding the optimal order using our proposed search-based technique.

This thesis has two parts:

**Summary:** This part of the thesis consists of the following sections: Section 2 provides the background information required to understand the summary of the thesis. Section 3 briefly presents the contributions of the thesis and Section 4 presents the research methodology that was followed during the thesis. Section 5 provides salient results from the research papers submitted as part of the thesis; Section 6 outlines future research directions, whereas Section 7 concludes the thesis.

**Papers:** The second part of the thesis presents the published or submitted research papers, which are included in this thesis.
2 Background

In this section, brief background knowledge is presented for better understanding the rest of thesis. Section 2.1 introduces background related with product line modelling, i.e., feature model, and component family model. Section 2.2 provides details in terms of search-based software testing (SBST) since we aim at applying various search techniques to support product line testing.

2.1 Modelling

In this section, background is first presented related with feature model (Section 2.1.1) followed by a brief introduction of component family model (Section 2.1.2).

2.1.1 Feature Model

Feature modeling was first proposed in the Feature-Oriented Domain Analysis (FODA) approach [25], which is a hierarchical modeling method with the aim at representing software product lines by capturing the commonalities and variabilities within a large number of products. Feature model (FM) has been successfully applied in a range of academia and industrial domains [1, 2, 25]. More specifically, FM can be represented as a 2-tuple \((features, constraints)\) with four types of features, namely \textit{mandatory}, \textit{optional}, \textit{alternative} and \textit{or}. Fig. 1 illustrates a simplified FM inspired by the laptop industry.

![Feature Model Diagram](image)

Fig. 1. A sample feature model

A \textit{mandatory} feature (e.g., \textit{Processor} feature shown as exclamation mark in Fig. 1) means it must be included if its parent feature is included in the current selection. An \textit{optional} feature (e.g., USB driver feature represented as question mark in Fig. 1) indicates the feature is not necessarily chosen even if its parent feature is included. A parent feature with a set of \textit{alternative} features (e.g., \textit{1GB 1600MHz} feature and \textit{4GB 1600MHz} feature shown as double-arrow mark in Fig. 1) describes that only one of the \textit{alternative} features
can be included if their parent feature is included. A parent feature with a set of or features (e.g., Camera feature and Radio feature described as cross-line mark in Fig. 1) means at least one of the or features is included if their parent feature is included.

In addition, FM contains cross-tree constraints, which specify complementary relations between two features in the FM. Such cross-tree constraints are usually defined based on the domain experts’ knowledge with the aim of eliminating invalid configurations. More specifically, two types of cross-tree constraints are typically used in the existing literatures [1], namely require and mutually exclusive constraints. A require relation among two features (a source and a target) means if the source feature is included into the current selection, the targeted feature must also be included. A mutually exclusive relation has the opposite meaning, saying that if the source feature is included then the target feature cannot be included into the current selection. For instance, there exists a require relation from High Resolution feature to 4GB 1600MHz feature, showing that a laptop can support screen with high resolution only if its memory achieves 4GB 1600MHz. Notice that there also exist other types of constraints (e.g., conjunctive normal form (CNF)-based constraints [26]) but in our case only require and mutually exclusive relations were applicable. However, this doesn’t limit our approach only to these two relations and other types of relations can be used and enforced with the relevant tool support.

2.1.2 Component Family Model

Component Family Model (CFM) is used to represent how products are assembled and generated in a product line by modeling relations among software architectural elements [27, 28]. More specifically, CFM can be represented as a 4-tuple (components, parts, source elements, restrictions). Components are named entities organized into a tree-like structure that can be of any depth. Each component represents one or more functional elements of the products in a product line (e.g. C functions, Java classes). Parts are named and typed entities. Each part belongs to a component and contains one or more source elements. A part can be associated with given programming language features, classes or objects, but it can also be associated with other key elements. A source element is an unnamed but typed entity. Source elements are usually used to determine how the source code for the specified element is generated.
Fig. 2 shows a simplified CFM also inspired by the laptop industry, which can be associated with the FM presented in Section 2.1.1. For instance, USB Driver is a component including one part, i.e., a class called VoidDriver. The class is realized by one source element namely VoidDriver.java, which a physical java file implemented for the USB driver functionality.

![Diagram of component family model](image)

**Fig. 2.** A sample component family model

When configuring a product, each element (component, part and source element) in CFM is included into the selection by default if it has no parent element or its parent element is included. Restrictions are defined to specify conditions under which a component, part or source element may be excluded from a final selection [27, 28]. More specifically, restrictions can be formalized as $A \rightarrow B$ showing that the element $B$ can be included into the final selection for configuration only if the restrictions evaluate to true (i.e., the element $A$ is selected). Notice that FM and CFM can be linked by restrictions, i.e., the elements (component, part and source element) in CFM can be included into the final configuration for a specific product only if the relevant features are selected in FM. For instance, a restriction can be assigned from the VoidDriver part in the CFM (shown as Fig. 2) to the USB driver feature in the FM (shown as Fig. 1), showing that the class VoidDriver and its child element (the Voiddriver.java source element) will not be included into the final configuration unless the feature USB driver is included into the feature selection.

### 2.2 Search-based Software Testing (SBST)

Generally speaking, a large number of software engineering problems (especially for various testing problems such as test selection problem) can be reformulated as an
optimization problem, which is well solved by various search algorithms [22, 30, 31]. Search algorithms aim at mimicking natural phenomenon, for example, natural evolution process or behaviors of animals (e.g., bird flocking) to search optimal solutions for various optimization problems [29]. The main advantage for applying search-based techniques is to find an optimal solution from an exhaustive space of potential solutions using an acceptable level of time. Several successful results of using search algorithms are reported in the literature for many types of software engineering problems [21, 22, 30-32].

To apply search algorithms, fitness function (objective function) should be defined mathematically to guide the search with aims at assessing the capability of the solutions. Different types of search algorithms can be applied [33] and Genetic Algorithms (GAs) are the most well known that is inspired by the Darwinian evolution theory. More specifically, for GAs, genes, individuals (also called chromosomes) should be carefully defined beforehand. More specifically, individuals refer to potential solutions with respect to the optimization problem and each individual (solution) is composed of a set of genes, which present units for the solution, e.g., test cases for the test suite in our case. The set of individuals handled by GAs is represented as population.

After properly defining fitness function (objective function), the search algorithms will be repeatedly run by applying various operators (e.g., selection) to produce the best individual (solution) until the terminating condition is satisfied (e.g., number of fitness evaluation). Three operators are typically used search algorithms, namely selection, crossover (also called recombination) and mutation [29]. More specifically, 1) Selection operator corresponds to the operation of selecting the solutions (e.g., individuals for evolutionary algorithms) with the best values based on the defined fitness function. Notice that the Selection operator can be applied to all the search algorithms; 2) Crossover operator is mainly applied in various GAs and refers to the operation of selecting two individuals (usually called parent individuals) and exchanging their genes partially. The genes that will be switched between the individuals are usually determined at random. New individuals obtained by crossover operator are usually called offspring; and 3) Mutation operator is also mostly adapted in GAs and refers to updating the individuals by changing the properties of genes randomly based on the predefined mutation rate (i.e., probability of changing the properties of genes).
3 Product Line Testing Methodologies

In this section, a set of methodologies is presented for addressing each research question (test selection, test minimization, test generation and test prioritization). Recall that the main motivation of the thesis is to facilitate systematic and automated testing of products in a cost-effective manner. In overall, the thesis can be divided by four main parts related to test selection, test minimization, test generation and test prioritization, respectively (Section 3.1-3.4)

3.1 Test Selection Methodology using Feature Model (Paper 1 and Paper 2)

Driven by the practical needs of industry, the first essential step for testing a specific product is to seek efficient test selection strategies to select relevant test thereby reducing the cost (e.g., selection time) and at the same time not compromising the effectiveness of the selected test cases. The current test case selection practice at one of our industrial partner (Cisco) is to select test cases manually from a repository of test cases, whenever a new VCS is to be tested. Due to the increasing complexity of functionalities and diversity of VCS products, manual selection poses several challenges. First, manual test case selection consumes a significant amount of time, which reduces the efficiency of testing; Second, the manual selection process is mainly driven by the expertise of test engineers and hence it is not an objective and repeatable process (i.e., different test engineers may select different sets of test cases for the same product). Third, manual selection may result in a set of test cases that has low effectiveness, e.g., coverage of testing functionalities (i.e., all required testing functionalities may not be covered by the selected test cases), because a focus has been placed on the testing of specific functionalities. Finally, no guidelines or methodology is provided to train recently hired engineers to select test cases. This means that the current practice of test selection largely depends on the expertise of test engineers and is not scalable when more VCSs are developed and are to be tested.

To cope with the above-mentioned challenges, a systematic methodology was proposed to support automated test case selection for product lines using feature model and component family model. As shown in Fig. 3, a Feature Model for Testing (FM_T) is first designed to capture commonalities and variabilities of a product line relying on domain experts’ knowledge and product line information (e.g., system documentation). Second, a Component Family Model for Testing (CFM_T) is built to model the repository of test
cases that is developed for testing the entire product line. When a new product $Product_i$ needs to be tested, test engineers are only required to perform selection through the Test Selection Front-end for FM_T and the relevant test cases will be chosen automatically, based on the restrictions from CFM_T to FM_T, and traces from CFM_T to the test cases in the Product Line Test Case Repository. Such restrictions and traces are automatically built using a tool we developed called Import Plugin and Transformation (IPT) that implements our methodology.

![Fig. 3. An overview of the test selection methodology](image)

The proposed test selection methodology was evaluated by applying it to the Saturn product line in Cisco with seven commercial products. The results show that the proposed methodology significantly reduces the cost of test selection (i.e., selection time) and at the same time achieves higher effectiveness (feature/feature pairwise coverage and fault detection) as compared to the current manual process. We also conducted a questionnaire-based study to solicit the views from the test engineers about our methodology. The obtained results show that they are positive about adapting our methodology to improve their current manual practice for test case selection. In addition, we present the tool support of our methodology and share a set of lessons learnt based on our experience of working with Cisco. In addition, to further assess the performance of our automated methodology as compared with the manual approach, we report a controlled experiment involving 20 graduate students from Beijing University of Aeronautics and Astronautics (BUAA), Beijing, China. The performance is assessed from three different perspectives, i.e., cost, effectiveness and efficiency using seven commercial VCS products. The results of the experiment showed that our automated methodology is significantly cost-effective as
compared with the manual approach. Moreover, the efficiency of our methodology is not significantly affected with the increased complexity of VCS products.

3.2 Search-Based Test Minimization Approach (Paper 3 and 4)

Followed by the test case selection step, we discovered that even after test selection, the number of test cases is still large (more than 2000 requiring 3-4 days to execute per product) and many of the test cases are redundant (e.g., several test cases cover the same functionality). On top of that, new test cases for product lines are continuously added further increasing the number of test cases. Executing all these test cases is practically impossible within the allocated testing time and thus warrants efficient and sophisticated test minimization techniques [7,18-21]. However, such minimization may reduce cost (e.g., execution time), but on the other hand may also reduce effectiveness (e.g., fault detection capability). This means an efficient technique must achieve a desirable balance among cost and effectiveness measures, which is a multi-objective optimization problem. A number of such objectives have been studied for multi-objective test optimization in regression testing in the existing literature such as execution time, test coverage and fault detection capability [8, 18]. Multi-objective search algorithms are well adapted for the type of problem we are solving in this paper [22, 23].

Fig. 4. An overview of the test minimization approach

To tackle such test minimization problem, as shown in Fig. 4, we defined a fitness function based on the following cost/effectiveness measures based on our industrial
collaboration and the existing literature [18, 19, 22, 23]: 1) Cost: Test Minimization Percentage (TMP) and Overall Execution Time (OET); 2) Effectiveness: Feature Pairwise Coverage (FPC), Fault Detection Capability (FDC) and Average Execution Frequency (AEF). An empirical evaluation for the proposed fitness function was further performed in conjunction with three weight-based multi-objective search algorithms and seven other multi-objective search algorithms on an industrial case study and assessing the scalability using 500 artificial problems of varying complexity. In addition, a tool called TEst Minimization with Search Algorithms (TEMSA) was designed and implemented to support our search-based test minimization technique (Paper 3). Notice that the proposed search-based test minimization approach was applied with the test selection methodology (Paper 3) but it can also be applied in other contexts separately.

The results of our empirical evaluation showed that: 1) Based on the industrial case study, all the selected search algorithms except Improved Strength Pareto Evolutionary Algorithm (SPEA2) are cost-effective for solving the test minimization problem and Random-Weighted Genetic Algorithm (RWGA) achieves the best performance; 2) For the 500 artificial problems, the results are consistent with the industrial case study, i.e., RWGA outperforms all the other search algorithms. Furthermore, RWGA managed to solve the problems of varying complexity (Paper 3).

Moreover, since weight-based search techniques (RWGA) can achieve the best performance for solving the minimization problem in our case, we further investigated the existing weight assignment strategies and proposed a new weight assignment strategy called Uniformly Distributed Weights (UDW) (Paper 4). We evaluate the proposed weight strategy UDW as compared with the two existing commonly-used weight assignment strategies, i.e., Fixed Weight (FW) strategy and Randomly-Assigned Weights (RAW) strategy using an industrial problem of test minimization for Video Conferencing Systems (VCSs) product line from Cisco Systems.

More specifically, a fitness function defined on all these five objectives is used in conjunction with the following search algorithms: Genetic Algorithm (GA), (1+1) Evolutionary Algorithm (EA), and Alternating Variable Method (AVM) to compare the three distinct weight assignment strategies. Random Search (RS) is used as the comparison base line. Moreover, inspired by the industrial problem, we created 500 artificial problems of varying complexity to evaluate the three weight assignment strategies in conjunction
with all the four algorithms. The obtained results show that: 1) With FW, RAW and UDW, (1+1) EA significantly outperformed the other search algorithms; 2) With (1+1) EA, UDW significantly performed better than FW and RAW; 3) The performance of (1+1) EA and GA with UDW was significantly improved with the increasing complexity of problems.

3.3 Test Generation Methodology using Feature Model (Paper 5)

When the product introduces new functionalities, new test cases should be designed and implemented for testing the new functionalities where Model-Based Testing (MBT) can support automated generation of executable test cases from models systematically [5, 24]. It is well known that major effort to apply MBT in practice is to develop models for System Under Test (SUT) in order to generate test cases. In our previous work [3], four types of variability for a product line were captured using standard Unified Modelling Language (UML) class diagrams, UML state machine diagrams, aspect class diagrams and aspect state machines, which were developed by a commercial tool called IBM Rational Software Architect (RSA). All these models are stored in a repository Product Line Behavioral Model Repository and used for test case generation after configuring them for each new product [3]. However, using such models for generation, test engineers are required to be familiar with concepts of all the behavioral models (e.g., UML class diagram). To ease the adoption of MBT in practice, a test generation methodology was proposed using FM and CFM to shield test engineers from all the above modelling expertise as shown in Fig. 5.

![Fig. 5. An overview of the test generation methodology](image-url)

More specifically, the proposed methodology includes the following main steps: 1) reusing the existing Feature Model for Testing (FM_T) to model testing functionalities of a
product line; 2) defining a Component Family Model for Behaviors (CFM_B) to associate its configurable parameters to model elements of a large number of behavioral models and related OCL constraints; 3) linking CFM_B with FM_T via restrictions. With our methodology based on FM_T and CFM_B, test engineers are only required to perform selection and configuration through the Test Generation Front-end for FM_T, such that all relevant behavioral models can be selected and configured automatically, based on the links built between FM_T and CFM_B, and between CFM_B and the Product Line Behavioral Model Repository. The configured models are then given input to our model-based testing tool, TRansformation-based tool for Uml-baSed Testing (TRUST) [34] for executable test case generation.

We evaluated our methodology with two means. First, we compared the configuration process of our proposed methodology with our existing methodology [3] and provided discussion on similarities and differences on the processes. Second, we applied our proposed methodology to the Saturn product line and configured its four products. The results showed that our methodology significantly reduces the complexity of configuration; thereby reducing required modelling effort. Moreover, the need to acquire expertise of modelling is also eliminated.

3.4 Search-Based Test Prioritization Approach (Paper 6)

Based on our further experience gained from the industrial collaboration, we also observe that test prioritization is crucial due to the ever-increasing number of test cases for the product line. In practice, considering limited budget (i.e., available time and resources), it is usually not feasible to execute all the possible test cases for testing the products thus it requires an efficient strategy to prioritize the given test cases. The target of such prioritization is minimization of the cost and maximization of the effectiveness of testing while meeting the given limited budget. However, it is commonly recognized that it is a trade-off among various costs (e.g., required resources) and effectiveness (e.g., feature coverage) objectives, which can be formulated as a multi-objective optimization problem.

To deal with the above-mentioned problem, we introduce a search-based multi-objective test prioritization technique as shown in Fig. 6. More specifically, cost and effectiveness measures are first defined with respect to the required objectives for prioritization. In our case, one cost measure and three effectiveness measures are carefully
defined based on the expertise of VCS testing, i.e., Overall Execution Cost (OEC), Prioritized Extent of test cases (PE), Feature Pairwise Coverage (FPC) and Fault Detection Capability (FDC). A novel fitness function is further defined considering all these cost/effectiveness measures and integrated with three commonly used search algorithms (Alternating Variable Method (AVM), Genetic Algorithm (GA) and (1+1) Evolutionary Algorithm (EA)) and Random Search (RS) as a comparison baseline. When particular products need to be tested, after obtaining the relevant test cases from the test case repository, test engineers can further cost-effectively prioritize the selected test cases for finding the optimal order using our proposed search-based technique (Fig. 6). Notice that the proposed prioritization approach was applied with the test selection methodology in our case (Paper 6) but it can also be applied separately or with other techniques, such as the proposed test minimization approach or test generation methodology.

Moreover, the defined fitness function along with the search algorithms is empirically evaluated using the Cisco’s industrial case study and 500 designed artificial problems. The results show that (1+1) EA achieves the best performance for solving the multi-objective test prioritization problem and it scales up to solve the problems of varying complexity.

Fig. 6. An overview of the search-based test prioritization approach
4 Research Method

In this section, we describe our research method used for the entire thesis that is based on the collaboration with one of our industrial partners (Cisco Systems, Norway) focusing on the automated test case selection with the aim of testing VCS product line. At a higher level, our research method can be divided into four steps, which are: 1) problem identification (Section 4.1): 2) problem formulation (Section 4.2): 3) solution realization (Section 4.3): 4) solution evaluation (Section 4.4).

4.1 Problem Identification

As an industry-driven research, the topic of the thesis was initiated through the collaboration with one of our industrial partners, Cisco with aims at understanding the industrial domain and identifying research problems. The identification of research problems took place through several meetings with Cisco for learning the domain of video conferencing system (VCS), the current testing activities and the potential challenges for VCS testing. Notice that such process of identification involved a group that is related to automated testing of VCSs, which includes one quality assurance (QA) manager and three test engineers. Based on the outputs from the meetings, the faced research problems were proposed in an industrial setting, i.e., cost-effective testing of products in product line in a systematic and automated way.

More specifically, when a new product comes into play, the first critical task for testing it is to select relevant test cases from the entire test suite developed for the product line (Test Selection). Moreover, we observed that redundant test cases may exist in the selected test cases and thus it requires eliminating the redundant ones with aims at improving the efficiency of testing (Test Minimization). In addition, if the product introduces a new functionality that cannot be tested by the existing test cases, new test cases should be developed or generated for testing the new functionality (Test Generation). Last, since the budget for testing is usually limited (e.g., time, test resources), it also requires performing prioritization for the obtained test cases before execution (Test Prioritization). Notice that these four tasks are closely associated and existing the whole testing process of VCS products. As a result, the problem for cost-effective testing products can be further divided into four research questions, i.e., test selection, test minimization, test generation and test prioritization, which become the main focus of the thesis.
4.2 Problem Formulation

As a first step to solve the above-mentioned problems, the existing testing techniques were first reviewed to match the proposed problems [5-8]. As a result, we decided to apply feature model (FM) and component family model (CFM) for tackling the test selection and generation problems in the context of product line since model-based techniques are systematic and automated without involving much human intervention [9-11]. Moreover, for test minimization and prioritization problems, search-based techniques were determined for applying since our potential solution space is extensively huge and the application of search-based techniques have proved to be cost-effective in terms of solving such complicated optimization problems [12-17].

Moreover, through careful investigation and discussion with test engineers, we identified and formulated a set of cost/effectiveness objectives as cost/effectiveness measures that should be considered when dealing with the proposed problems, such as feature pairwise coverage and fault detection capability (e.g., Paper 1, 2, 3, 5, 6). Notice that all these objectives were mathematically defined and formulated with aims at further applying the selected techniques and conducting empirical evaluation.

4.3 Solution Realization

This step mainly focuses on realizing the proposed solutions according to the results of step 2 for addressing each research question. Notice that before going into the detailed implementation, we came up with the proposed solutions (e.g., FM, CFM and search-based techniques) to the test engineers at Cisco to ensure the solutions compatible with the real industrial environment surrounded. The process of solution realization was iterative that involved a lot of interaction with the test engineers at Cisco. For instance, for test selection problem, it took three months (around twelve meetings and each one took on average two hours) to build and refine the feature model for the Saturn product line for ensuring the built feature model can be used to accommodate the requirements of VCS testing. In addition, we also designed and implemented corresponding tool support with respect to the realized solutions, e.g., Import Plugin and Transformation (IPT) for test selection (Paper 1 and 2), TEst Minimization using Search Algorithms (TEMSA) for test minimization (Paper 3) and Import Plugin and Transformation for Behaviors (IPTB) for test generation (Paper 5).
4.4 Solution Evaluation

An important fundamental part of the thesis was to carry out various evaluation for assessing the proposed methodologies in terms of each research problem. More specifically, we evaluated our proposed methodologies via different means, such as industrial case study, artificial problems, questionnaire-based study and controlled experiments. The goal for such extensive evaluation is to ensure that the proposed methodologies are beneficial that can assist to improve the current testing practice. For instance, for test selection problem, our methodology was evaluated via three ways. First, we applied our methodology to the Saturn product line of Videoconferencing Systems developed by Cisco, Norway and performed test case selection for its seven products (Paper 1 and 2). Second, we conducted a questionnaire-based study to solicit the views of our automated methodology from test engineers at Cisco (Paper 1 and 2). Third, we report the results of a carefully designed controlled experiment with the aim of evaluating the performance of our automated methodology as compared with manual approach in terms of cost, effectiveness and efficiency (Paper 2). Another example is for test minimization problem, we conducted an extensive empirical evaluation to assess the performance of the proposed fitness function along with ten search algorithms based on an industrial case study and 500 artificial problems inspired from the industrial case study (Paper 3).
5 Summary of Results

In this section, a summary is presented for each paper submitted as part of this thesis together with the key results.

5.1 Paper 1

This paper aims at addressing the test selection challenges in the context of product lines. More specifically, test case selection aims at obtaining a set of relevant test cases for a product from the entire set of test cases available for a product line. While working on a research-based innovation project on automated testing of product lines of Video Conferencing Systems (VCSs) developed by Cisco, we felt the need to devise a cost-effective way of selecting relevant test cases for a product. To fulfil such need, we propose a systematic and automated test selection methodology using: 1) Feature Model for Testing (FM_T) to capture commonalities and variabilities of a product line relying on domain experts’ knowledge and product line information (e.g., system documentation); 2) Component Family Model for Testing (CFM_T) to model the repository of test cases that is developed for testing the entire product line; 3) A tool to automatically build restrictions from CFM_T to FM_T and traces from CFM_T to the actual test cases. Using our methodology, a test engineer is only required to select relevant features through FM_T at a higher level of abstraction for a product and the corresponding test cases will be obtained automatically.

We evaluate our methodology by applying it to a VCS product line called Saturn with seven commercial products. The results showed that our methodology can significantly reduce test selection time and at the same time preserves better effectiveness as compared with the current manual process. Second, we conducted a questionnaire-based study to solicit the views of our proposed methodology from test engineers at Cisco. The results showed that the test engineers are positive about adapting our methodology in their current practice. We also presented the tool support for the methodology, which has already been adopted in Cisco’s practice for test case selection. In addition, we shared a set of lessons
learnt based on our industrial collaboration with the aim to provide guidance to other practitioners.

5.2 Paper 2


Invited journal paper that is submitted to the Journal of Software and Systems Modeling (SOSYM).

This paper still focuses on the problem related test case selection in the context of product line. Notice that both this paper and Paper 1 are invited journal extensions from a conference version (Paper 7). Both the two papers are based on the same test selection methodology using feature model and component family model. As compared with paper 1, the main difference is that a thorough controlled experiment is designed to further collect evidence about the benefits of our proposed automated methodology. The main goal for such controlled experiment is to compare cost, effectiveness, and efficiency of our automated methodology (Automated) with the manual approach (Manual) using trained graduate students. For Cost, one measure is defined to measure the time to select test cases for a product based on the required testing functionalities namely test selection time (Sel-Time). For Effectiveness, four measures are defined, including: 1) Number of covered features (NCF) is to measure how many features can be covered by the selected test cases for a product; 2) Feature coverage (FC) is to calculate the percentage of relevant features covered by the selected test cases; 3) Number of selected test cases (NST) is calculate how many test cases are selected for a product; and 4) Percentage of selected test cases (PST) is to measure the percentage of relevant test cases selected for a product. For Efficiency, two measures are defined, i.e., 1) NCF efficiency (Eff_F) is to measure the number of features that are covered per unit of selection time and 2) NST efficiency (Eff_TC) is to measure the number of test cases that are selected per unit of selection time.

Through such controlled experiments, the following research questions were answered.

**RQ1**: Does Automated significantly reduce the Cost of test case selection as compared to Manual?

The results showed that Automated took significantly less time than Manual and thus
we can conclude that Cost of test case selection for Automated is significantly less than Manual.

**RQ2:** Does Automated significantly improve the Effectiveness of test case selection as compared to Manual?

The results show that Automated has significantly higher Effectiveness than Manual, i.e., Automated can significantly improve the effectiveness in terms of number of covered features (NCF), feature coverage (FC), number of selected test cases (NST), and percentage of selected test cases (PST) when compared with Manual.

**RQ3:** Does Automated significantly improve the Efficiency of test case selection as compared to Manual?

The results show that Automated significantly outperforms Manual in terms of Efficiency, i.e., Automated can significantly improve the efficiency of test case selection measured as efficiency for covering features (Eff_F) and efficiency for selecting test cases (Eff_TC) as compared with Manual.

**RQ4:** How does the increasing complexity of systems affect Cost, Effectiveness and Efficiency of Automated and Manual as compared to each other?

The results show that Automated can preserve high performance (cost, effectiveness and efficiency) even with the increase in the complexity of products.

**RQ5:** How does the performance of Experts compare with graduate students in terms of Cost, Effectiveness and Efficiency?

The results show that there is no significant difference between the results obtained by experts and graduate students in terms of Sel-Time, NCF, FC, NST, PST, Eff_F and Eff_TC when using Automated.

### 5.3 Paper 3


This paper particularly aims at tackling cost-effective test minimization for product lines, which identifies and eliminates redundant test cases from test suites in order to
reduce the total number of test cases to execute, thereby improving the efficiency of testing. However, such minimization may result in the minimized test suite with low test coverage, low fault revealing capability, low priority test cases, and require more time than the allowed testing budget (e.g., time) as compared to the original test suite. To deal with the above issues, we formulated the minimization problem as a search problem and defined a fitness function considering various optimization objectives based on the above issues.

To assess the performance of our fitness function, we conducted an extensive empirical evaluation by investigating the fitness function with three weight-based Genetic Algorithms (GAs) and seven multi-objective search algorithms using an industrial case study and 500 artificial problems inspired from the industrial case study. The results showed that Random-Weighted Genetic Algorithm (RWGA) significantly outperforms the other algorithms since RWGA can balance all the objectives together by dynamically updating weights during each generation. Based on the results of our empirical evaluation, we also implemented a tool called TEst Minimization using Search Algorithms (TEMSA) to support test minimization using various search algorithms in the context of product lines.

5.4 Paper 4


This paper is an incremental work based on Paper 3, which focuses on various weight assignment strategies that can have huge impact on the performance of weight-based search techniques. More specifically, weight-based multi-objective optimization requires assigning appropriate weights using a weight strategy to each of the objectives such that an overall optimal solution can be obtained with a search algorithm. Choosing weights using an appropriate weight strategy has a huge impact on the obtained solutions and thus warrants the need to seek the best weight strategy. In this paper, we proposed a new weight strategy called Uniformly Distributed Weights (UDW), which generates weights from uniform distribution, while satisfying a set of user-defined constraints among various cost and effectiveness measures.
We compare UDW with two commonly used weight strategies, i.e., Fixed Weights (FW) and Randomly-Assigned Weights (RAW), based on five cost/effectiveness measures for an industrial problem of test minimization defined in the context of Video Conferencing System Product Line developed by Cisco Systems. We empirically evaluate the performance of UDW, FW, and RAW in conjunction with four search algorithms ((1+1) Evolutionary Algorithm (EA), Genetic Algorithm, Alternating Variable Method, and Random Search) using the industrial case study and 500 artificial problems of varying complexity. Results show that UDW along with (1+1) EA achieves the best performance among the other combinations of weight strategies and algorithms.

5.5 Paper 5


This paper tries to deal with the challenges of test generation in the context of Model-Based Testing (MBT) of product lines. In our previous work [3], a methodology was proposed to capture variability in configurable UML state machines and aspect state machines. For each product, these state machines are to be configured for generating executable test cases. In this paper, an extended methodology is proposed using Feature Model for Testing (FM_T) and Component Family Model for Behaviors (CFM_B). FM_T captures variable testing functionalities of a product line, whereas CFM_B provides an abstraction layer on top of the configurable state machines. With our current methodology, a test engineer doesn’t need to acquire expertise on behavioral modelling and can simply configure models for a product by selecting features in FM_T and configuring provided attributes in CFM_B. The configured models are then given input to our model-based testing tool, TRansformation-based tool for Uml-baSed Testing (TRUST) for executable test case generation.

We applied our extended methodology to a product line of video conferencing system developed by Cisco Systems, Norway. Results show that the methodology significantly reduces the complexity of configuration; thereby significantly reducing required effort and cost (e.g., in terms of training). In addition, it does not require test engineers to have
expertise in UML modelling, aspect-oriented modelling, and OCL specification and therefore eases the adoption of MBT in industry.

5.6 Paper 6


This paper focuses on product line test prioritization, which is crucial for testing products in a product line considering limited budget in terms of available time and resources. In general, it is not practically feasible to execute all the possible test cases and so, ordering test case execution permits test engineers to discover faults earlier in the testing process. An efficient prioritization of test cases for one or more products requires a clear consideration of the trade-off among various costs (e.g., time, required resources) and effectiveness (e.g., feature coverage) objectives. As an integral part of the future Cisco’s test scheduling system for validating video conferencing products, we introduce a search-based multi-objective test prioritization technique, considering multiple cost and effectiveness measures.

In particular, our multi-objective optimization setup includes the minimization of execution cost (e.g., time), and the maximization of number of prioritized test cases, feature pairwise coverage and fault detection capability. Based on cost-effectiveness measures, a novel fitness function is defined for such test prioritization problem. The fitness function is empirically evaluated together with three commonly used search algorithms (e.g., (1+1) Evolutionary algorithm (EA)) and Random Search as a comparison baseline based on the Cisco’s industrial case study and 500 artificial designed problems. The results show that (1+1) EA achieves the best performance for solving the test prioritization problem and it scales up to solve the problems of varying complexity.
6 Future Directions

In this section, we discuss possible future directions, based on the above-mentioned four perspectives, i.e., test selection, test minimization, test generation, and test prioritization.

Regarding test selection, we plan to adapt our proposed test case selection methodology to other product lines and conduct a large-scale empirical study to assess the cost/effectiveness of our proposed methodology.

Regarding test minimization, the short-term future plan includes the analysis of another industrial case study in order to complement our experimental results and refine the design of our tool TEst Minimization using Search Algorithms (TEMSA) for supporting the proposed minimization technique. We also plan to refine the fitness function by considering other objectives, such as the number of available computing resources, and evaluate other search algorithms, such as Archive-Based hYbrid Scatter Search (AbYSS), which combines both scatter search and genetic algorithms. In terms of proposed weight assignment strategy, i.e., Uniformly Distributed Weights (UDW), one plan is to replicate our experiments in other industrial case studies for assessing the proposed weight strategy UDW. Another plan could be investigation of the effect of uniformly distributed weights on a diverse range of search algorithms.

Regarding test generation, the possible future direction is to apply our extended methodology to more product lines for assessing in different industrial case studies since the methodology was only based on the needs of our industrial partners. Moreover, another potential direction is to conduct a questionnaire-based study with test engineers to further collect evidence in terms of the applicability of our methodology in an industrial setup.

Regarding test prioritization, the first plan is to integrate our solution into the existing practice of Cisco. The second plan is to conduct additional case studies to further strengthen our experimental results and the fitness function with additional cost/effectiveness measures if necessary. Another possible plan is to investigate more multi-objective search techniques (e.g., Pareto-based search algorithms) to assess the performance and scalability along with the defined fitness function in industrial settings.
7 Conclusion

This thesis reported a set of systematic and automated methodologies called Product Line Model-based Testing Methodologies (PL-MTM) with the aim of supporting cost-effective testing products in product line. The PL-MTM aims at addressing four essential testing activities driven by an industrial need, i.e., test selection, test minimization, test generation and test prioritization. For each testing activity, we discussed how we investigated a solution to address the existing challenges for achieving the ultimate goal of automated and systematic testing products. We also presented thorough empirical evaluation via various means (e.g., industrial case study, controlled experiment) and tool support for each solution.

More specifically, for test selection (Paper 1 and 2), we proposed an automated and systematic methodology using feature model and component family model. With our methodology, test engineers only require performing feature selection through feature model and the relevant test cases are retrieved automatically from the repository thus hiding the unnecessary implementation details of test cases that are not required for test selection. The results of evaluation showed that our test case selection methodology can significantly reduce test selection time and at the same time preserves better effectiveness as compared with the current manual practice.

For test minimization (Paper 3 and 4), we proposed and evaluated a fitness function in conjunction with ten multi-objective search algorithms, to minimize test suites in the context of product line testing. Our fitness function takes effectiveness measures (i.e., Test Minimization Percentage (TMP), Feature Pairwise Coverage (FPC), Fault Detection Capability (FDC) and Average Execution Frequency (AEF)) and cost measures (i.e., Overall Execution Time (OET)) into account to guide the search. The results of thorough experiments showed that Random-Weighted Genetic Algorithm (RWGA) achieved acceptable performance for all the objectives and significantly outperformed the other search algorithms. In addition, RWGA can solve a wider range of problems and its performance is improved with the growth of complexity of problems.

Moreover, we proposed a new weight assignment strategy called Uniformly Distributed Weights (UDW) to generate weights by solving constraints among them with uniform distribution for solving multi-objective optimization problems. Based on the results of
evaluation, we concluded that assigning weights based on \textit{UDW} can significantly improve the performance of (1+1) EA for multi-objective optimization as compared with the two existing weight assignment strategies.

For test generation (Paper 5), we proposed an extension to our previous product line modelling and configuration methodology [3] to support Model-Based Testing (MBT) using feature model and component family model. The aim for such methodology is to reduce modelling effort and eliminate the needs to acquire expertise of UML modelling, Aspect-Oriented Modelling (AOM) and Object Constraint Language (OCL). The evaluation results showed that our extended test generation methodology significantly reduced the complexity of configuration and eliminated the need to acquire expertise of modelling and thus reducing required modelling effort.

For test prioritization (Paper 6), we proposed a search-based technique for cost-effective prioritization of a given test cases considering a limited budget (test resources). The defined fitness function took one cost measure and three effectiveness measures into account and evaluated in conjunction with three search algorithms. The results for evaluation indicated that (1+1) Evolutionary Algorithm (EA) along with our defined fitness function achieved the performance and thus can assist test engineers to cost-effectively solve the test prioritization problems of varying complexity.
References for the Summary

CiteSeer, 2002.


A Systematic Test Case Selection Methodology for Product Lines: Results and Insights from an Industrial Case Study

Shuai Wang¹, ², Shaukat Ali¹, Arnaud Gotlieb¹, and Marius Liaaen³

¹Certus Software V&V Center, Simula Research Laboratory, Norway
²Department of Informatics, University of Oslo, Norway
³Cisco Systems Inc., Norway
{shuai, shaukat, arnaud}@simula.no
marliaae@cisco.com

Abstract. In the context of product lines, test case selection aims at obtaining a set of relevant test cases for a product from the entire set of test cases available for a product line. While working on a research-based innovation project on automated testing of product lines of Video Conferencing Systems (VCSs) developed by Cisco, we felt the need to devise a cost-effective way of selecting relevant test cases for a product. To fulfill such need, we propose a systematic and automated test selection methodology using: 1) Feature Model for Testing (FM_T) to capture commonalities and variabilities of a product line; 2) Component Family Model for Testing (CFM_T) to model the structure of test case repository; 3) A tool to automatically build restrictions from CFM_T to FM_T and traces from CFM_T to the actual test cases. Using our methodology, a test engineer is only required to select relevant features through FM_T at a higher level of abstraction for a product and the corresponding test cases will be obtained automatically. We evaluate our methodology by applying it to a VCS product line called Saturn with seven commercial products and the results show that our methodology can significantly reduce cost measured as test selection time and at the same time achieves higher effectiveness (feature coverage, feature pairwise coverage and fault detection) as compared with the current manual process. Moreover, we conduct a questionnaire-based study to solicit the views of test engineers who are involved in developing FM_T and CFM_T. The results show that test engineers are positive about adapting our methodology in their current practice. Finally, we
present a set of lessons learnt while applying product line engineering at Cisco for test case selection.

**Keywords:** Test Case Selection, Product Line, Feature Model, Component Family Model

### 1 Introduction

Product Line Engineering (PLE) is a collection of methods, tools, and techniques to develop similar products by systematic reuse of common functionalities and configuring variabilities across the products [1, 2]. PLE can be adapted to achieve a number of goals and has already shown promising benefits in both academia and industry such as increased productivity, reducing development time and cost, speeding up product time-to-market [3]. In the context of software systems, PLE doesn’t necessarily have to be applied from the beginning to the end of software development life cycle and can be employed at any stage. The work presented in this paper is an example of the application of PLE at the testing phase for exclusively supporting test case selection for product lines. More specifically, this work is one of the projects in Certus Software Verification and Validation Center—a research-based innovation center funded by Norwegian Research Council with several industrial and public administration user partners. This project is about automated testing of a Video Conferencing System (VCS) product line called *Saturn* developed by Cisco Systems, Norway [4, 5].

For testing *Saturn*, the current practice is to develop a test case repository containing a large number of test cases (on average 2000 test cases). The repository is modified and updated as new products are released or the current products are evolved. Due to the increasing complexity of functionalities and diversity of products, the number of test cases increases significantly making it practically impossible to execute all of them [6, 7]. It is therefore important to seek efficient test selection strategies to select relevant test cases for testing a specific product thereby reducing the cost (e.g., selection time) and at the same time not compromising the effectiveness of the selected test cases.

In recent years, more and more researchers have spent significant effort on fully automated strategies for test case selection, which have proven to be efficient as compared to manual strategies [7, 8]. However, based on our industrial collaboration, we found that the common practice is still manual having the following challenges: 1) It is time-
consumer and error-prone, which increases the selection effort thereby reducing the efficiency of testing; 2) largely dependent on domain expertise; 3) It is not objective and repeatable process and various test cases can be selected for the same product by different engineers; 4) test cases with low effectiveness (e.g., coverage and fault detection) may be obtained; 4) training new engineers requires a lot of effort (e.g., time) since there are no systematic guidelines and methodology.

To deal with the above challenges, we propose a systematic methodology to support automated test case selection for product lines using feature model and component family model. As shown in Fig. 1, a Feature Model for Testing (FM_T) is first designed to capture commonalities and variabilities of a product line relying on domain experts’ knowledge and product line information (e.g., system documentation). Second, a Component Family Model for Testing (CFM_T) is built to model the repository of test cases that is developed for testing the entire product line. When a new product Product_i needs to be tested, test engineers are only required to perform selection through the Test Selection Front-end for FM_T and the relevant test cases will be chosen automatically, based on the restrictions from CFM_T to FM_T, and traces from CFM_T to the test cases in the Product Line Test Case Repository. Such restrictions and traces are automatically built using a tool we developed called Import Plugin and Transformation (IPT) that implements our methodology.

We evaluate our proposed methodology by applying it to the Saturn product line in Cisco with seven commercial products. The results show that the proposed methodology significantly reduces the cost of test selection (i.e., selection time) and at the same time achieves higher effectiveness (feature/feature pairwise coverage and fault detection) as compared to the current manual process. We also conducted a questionnaire-based study to
solicit the views from the test engineers about our methodology. The obtained results show that they are positive about adapting our methodology to improve their current manual practice for test case selection. In addition, we present the tool support of our methodology and share a set of lessons learnt based on our experience of working with Cisco.

This paper is an extension of a conference paper [9]. The key differences with the conference version are: 1) Details on our research method to apply PLE in practice and additional details on our industrial problem/proposed methodology are added; 2) Problem representation including the definitions of one cost measure and three effectiveness measures; 3) A detailed comparison of the proposed methodology with the manual approach based on the defined cost/effectiveness measures using an industrial case study including seven commercial products; 4) A set of lessons learnt while adapting PLE in Cisco for test selection.

The rest of the paper is organized as follows: Section 2 provides background about feature model and component family model. Section 3 describes our research method used for the industrial collaboration. Section 4 proposes our methodology using FM_T and CFM_T including a running example to explain our methodology. Section 5 discusses the tool support. Section 6 presents the results of evaluation and a set of lessons learnt is reported in Section 7. Section 8 discusses the related work and Section 9 concludes the paper.

2 Background

In this section, we briefly introduce feature model (Section 2.1), followed by related background of component family model (Section 2.2).

2.1 Feature Model

Feature modeling was first proposed in the Feature-Oriented Domain Analysis (FODA) approach [10], which is a hierarchical modeling method with the aim at representing software product lines by capturing the commonalities and variabilities within a large number of products. Feature model (FM) has been successfully applied in a range of academia and industrial domains [2, 5, 10]. More specifically, FM can be represented as a
2-tuple \((\text{features}, \text{constraints})\) with four types of features, namely mandatory, optional, alternative and or. Fig. 2 illustrates a simplified FM inspired by the laptop industry.

A mandatory feature (e.g., Processor feature shown as exclamation mark in Fig. 2) means it must be included if its parent feature is included in the current selection. An optional feature (e.g., USB driver feature represented as question mark in Fig. 2) indicates the feature is not necessarily chosen even if its parent feature is included. A parent feature with a set of alternative features (e.g., 1GB 1600MHz feature and 4GB 1600MHz feature shown as double-arrow mark in Fig. 2) describes that only one of the alternative features can be included if their parent feature is included. A parent feature with a set of or features (e.g., Camera feature and Radio feature described as cross-line mark in Fig. 2) means at least one of the or features is included if their parent feature is included.

Fig. 2. A sample feature model

In addition, FM contains cross-tree constraints, which specify complementary relations between two features in the FM. Such cross-tree constraints are usually defined based on the domain experts’ knowledge with the aim of eliminating invalid configurations. More specifically, two types of cross-tree constraints are typically-used in the existing literatures (Benavides et al. 2010), namely require and mutually exclusive constraints. A require relation among two features (a source and a target) means if the source feature is included into the current selection, the targeted feature must also be included. A mutually exclusive relation has the opposite meaning, saying that if the source feature is included then the target feature cannot be included into the current selection [5]. For instance, there exists a require relation from High Resolution feature to 4GB 1600MHz feature, showing that a laptop can support screen with high resolution only if its memory achieves 4GB 1600MHz. Notice that there also exist other types of constraints (e.g., conjunctive normal form (CNF)-based constraints [11]) but in our case only require and mutually exclusive relations
were applicable. However, this doesn’t limit our approach only to these two relations and other types of relations can be used and enforced with the relevant tool support.

2.2 Component Family Model

Component Family Model (CFM) is used to represent how products are assembled and generated in a product line by modeling relations among software architectural elements [12, 13]. More specifically, CFM can be represented as a 4-tuple (components, parts, source elements, restrictions). Components are named entities organized into a tree-like structure that can be of any depth. Each component represents one or more functional elements of the products in a product line (e.g. C functions, Java classes). Parts are named and typed entities. Each part belongs to a component and contains one or more source elements. A part can be associated with given programming language features, classes or objects, but it can also be associated with other key elements. A source element is an unnamed but typed entity. Source elements are usually used to determine how the source code for the specified element is generated.

Fig. 3 shows a simplified CFM also inspired by the laptop industry, which can be associated with the FM presented in Section 2.1. For instance, USB Driver is a component including one part, i.e., a class called VoidDriver. The class is realized by one source element namely VoidDriver.java, which a physical java file implemented for the USB driver functionality.

Fig. 3. A sample component family model

When configuring a product, each element (component, part and source element) in CFM is included into the selection by default if it has no parent element or its parent element is included. Restrictions are defined to specify conditions under which a
*component, part or source element* may be excluded from a final selection [12, 13]. More specifically, restrictions can be formalized as $A \rightarrow B$ showing that the element $B$ can be included into the final selection for configuration only if the restrictions evaluate to true (i.e., the element $A$ is selected). Notice that FM and CFM can be linked by restrictions, i.e., the elements (*component, part and source element*) in CFM can be included into the final configuration for a specific product only if the relevant features are selected in FM. For instance, a restriction can be assigned from the *VoidDriver* part in the CFM (shown as Fig. 3) to the *USB driver* feature in the FM (shown as Fig. 2), showing that the class *VoidDriver* and its child element (the *VoidDriver.java* source element) will not be included into the final configuration unless the feature *USB driver* is included into the feature selection. In our case, we will apply restrictions for test case selection as discussed in Section 4.

3 Research Method

Recall that the work presented in this paper is one of the projects in Certus Software Verification and Validation Center in Norway focusing on automated testing (http://certus-sfi.no). The center is funded to bring research-based innovation in Norwegian industry by working closely with industrial partners. This work is in collaboration with Cisco Systems in Norway, which develops Video Conferencing Systems (VCSs), ranging from desktop versions to high-end hardware products.

In this section, we describe our research method used for the collaboration with Cisco focusing on the automated test case selection with the aim of testing VCS product line. At a higher level, our research method can be divided into four steps, which are: 1) problem identification (Section 3.1): 2) problem formulation (Section 3.2): 3) solution realization (Section 3.3): 4) solution evaluation (Section 3.4). Notice that the group we collaborate with is related to automated testing of VCSs, which includes one quality assurance (QA) manager and three test engineers. More specifically, the four industrial practitioners are working in the testing field for 13 years, 7 years, 5 years and 1 year, respectively, i.e., the group we have been collaborating with have significant experience in testing, especially in VCS testing. Notice that all of them were involved in the development of the proposed methodology with researchers from Certus.
3.1 Problem Identification

The process of problem identification started with a kickoff meeting with Cisco with the following three primary objectives: 1) Understanding the VCS domain at a high level; 2) Understanding current testing practice, tools, and technologies; 3) Discussion on the current testing challenges. Based on the kickoff meeting, we observed that diverse range of VCS products are developed that share common functionalities to a large extent and differ from each other for some functionalities. Such observation encouraged us to find a mapping of their current process to Product Line Engineering (PLE), although, the testing group wasn’t aware of PLE. Moreover, we identified the following testing challenges faced by them; 1) Too much manual effort for testing a new VCS; 2) High reliance on the domain experts to conduct testing. The overall objective that Cisco wanted to achieve was to improve the overall efficiency of testing new VCSs.

Afterwards, a set of workshops was held to build understanding of the concepts in PLE modeling such as feature models and PLE modeling tools. The purpose of workshops was to give test engineers a hand on experience with PLE. During the workshops, we spent time with the test engineers and tried to model various testing features for two main purposes: 1) To get test engineers familiarize with feature modeling; 2) Understanding the domain. Through these workshops, we agreed that to improve the efficiency of testing a new VCS, we will apply PLE.

Moreover, we arranged several meetings with the aim of understanding current testing tools and technologies being used. For example, we learned that the test cases for VCS testing refer to executable Python test scripts, which includes a set of statements for test setup, calls to APIs and assertions. We also learned that to test a new VCS, on average 300 new test cases are implemented, which means that the complexity of the test case repository is continuously increasing. The current test repository includes more than 2000 test cases for testing various VCS products. Each test case takes on average 500 seconds to execution and it usually requires on average 3-4 days for test execution per product. In addition, before execution, it also requires setting up a correct environment (e.g., correct network and software version), which is done manually by test engineer. We also observed that the execution of test cases is fully automated, but the reuse of test cases was manually done, which is time-consuming and error-prone and thus warrants a cost-effective reuse approach.
Generally speaking, the current practice for manual test case selection largely depends on the expertise of test engineers, which is not systematic and scalable. Therefore, the first problem to be tackled was Test Case Selection for a product in a product line, which can be more precisely defined as: Selecting a subset of existing test cases for testing a new product entirely or a subset of functionality, which can automatically and systematically meet a set of testing criteria. In other words, the goal is to ensure that the selected test cases should be cost-effective as compared to the ones obtained by the current manual process.

The identified problem was further discussed with the testing group with test engineers, who further confirmed that the identified problem must be tackled first.

3.2 Problem Formulation

As a first step to solve the above-mentioned problem, we reviewed the existing test case selection techniques in regression testing since regression testing has been studied in academia for a long time, which shares similarities with product line testing. Based on two systematic reviews [7, 8], we found that most of the techniques for test case selection in regression testing can be categorized into two aspects; 1) Selection based on code coverage, i.e., selecting test cases achieving maximum code coverage; 2) Selection based on specification changes, i.e., selecting test cases covering maximum changes of modified specification for requirements.

Through careful investigation and discussion with test engineers, we learned that the problem we were facing could be addressed by the existing techniques, but with slightly different context. In the case of regression testing, the context of testing is one software system, which is being changed and test case selection has to be performed for the changed/impacted parts of the software. In our current context, i.e., product lines, a product line consists of a number of products rather than a single evolving system. More specifically, our objective is to select relevant test cases for testing a new product rather than evolved version of single software system and the new product shares commonalities and owns its variabilities as compared with the existing products. Therefore, the main challenge in our context is to focus on the characteristics of product line (i.e., capture the commonalities and variabilities among a large number of products) and select relevant test cases for a new product, which requires to be tested, where feature model (FM) and
component family model (CFM) can be adapted to model the VCS product line and the test case repository (Section 2). Based on the above discussions, our test case selection problem can be formulated mathematically as below.

Let \( P = \{p_1, p_2, p_3 \ldots p_{np}\} \) is a product line with a set of products, where \( np \) is the number of products in \( P \).

\( F = \{f_1, f_2, f_3 \ldots f_{nf}\} \) is a feature model with a set of features to represent \( P \), where \( nf \) is the number of features, i.e., functionalities need to be tested for \( P \). Notice that the type of \( f_i \) in \( F \) can be mandatory, optional, alternative and or as discussed in Section 2.1.

\( Cons = \{cons_1, cons_2, cons_3 \ldots cons_{ncons}\} \) is a set of cross-tree constraints defined on \( F \), where \( ncons \) is the number of defined cross-tree constraints. Notice that in our case, \( cons_i \) is applied on feature model \( F \) using require or mutually exclusive, i.e., \( cons_i \) can be represented as \( cons_i = require (f_m, f_n) \) or \( cons_i = exclusive (f_m, f_n) \), where \( f_m \) is the source feature and \( f_n \) is the target feature (Section 2.1).

\( TS = \{t_1, t_2, t_3 \ldots t_{nt}\} \) is a test suite including a large number of test cases (\( nt \)) for testing \( P \).

\( F_{p_i} = \{f_1, f_2, f_3 \ldots f_{nf_{p_i}}\} \) be a subset of \( F \) to represent a specific product \( p_i \) which is to be tested where \( f_i \) can be any feature in \( F \) (\( f_i \in F \)) and \( nf_{p_i} \) is the number of features of \( F_{p_i} \) (\( 1 \leq nf_{p_i} \leq nf \)).

Moreover, we defined one cost measure and three effectiveness measures together with test engineers at Cisco based on the VCS testing domain knowledge, i.e., 1) Cost: selection time; and 2) Effectiveness: feature coverage, feature pairwise coverage and fault detection capability, which are defined as below.

**Cost Measures.** Cost measures are used to measure the cost taken by our proposed methodology as compared with the current manual approach. We defined Selection Time (Sel-Time) to measure the time for selection of test cases for testing the product \( p_i \). More specifically, Sel-Time in our case includes the time for 1): identifying the testing requirements to determine which functionalities should be tested; and 2) manually or automatically going through the entire test case repository and selecting relevant test cases to test the identified testing functionalities.
Effectiveness Measures. Such measures are defined and used to assess the effectiveness of our automated methodology as compared with the existing manual approach. We defined the following three measures.

Feature Coverage (FC) is defined to measure how much 1-wise coverage can be achieved, i.e., how many features are covered by the selected subset of $TS$, which can be calculated as below.

$$FC_{TS_{pi}} = \frac{Num_{F_{TS_{pi}}}}{nf_{pi}}$$

$Num_{F_{TS_{pi}}}$ is the unduplicated number of features covered by the selected $TS_{pi}$ that is a subset of $TS$, which can be measured as follows.

$$Num_{F_{TS_{pi}}} = \sum_{i=1}^{nf_{pi}} Num_{F_{tc_{i}}}$$

$Num_{F_{tc_{i}}}$ is the number of features covered by the test case $tc_{i}$. Notice that if some features are covered by multiple test cases, then the duplicated features will be not be considered when calculating $Num_{F_{TS_{pi}}}$. Notice that FC value ranges from 0 to 1 and a higher value represents better feature coverage. FC is the minimum coverage requirement that must be achieved by a selected subset of test cases as discussed with Cisco.

Feature Pairwise Coverage (FPC) is defined to measure how much pairwise coverage can be achieved, i.e., how many feature pairs are covered by the selected subset of $TS$ for testing the product $p_{i}$. We chose this type of coverage based on domain knowledge, discussion with test engineers, and history data about faults since a higher percentage of detected faults are sometimes due to the interactions between features. In our case, FPC is designed to compute the capability of covering feature pairs by the selected test cases, which can be calculated as follows.

$$FPC_{TS_{pi}} = \frac{Num_{FP_{TS_{pi}}}}{Num_{FP_{pi}}}$$

$Num_{FP_{TS_{pi}}}$ is the unique feature pairs covered by the selected test suite $TS_{pi}$ and $Num_{FP_{pi}}$ refers to all number of feature pairs for testing the product $p_{i}$, which can be
measured as \( \text{Num}_{FP_{pi}} = C^\text{size}(f_{pi}) = nt_{pi} \cdot (nt_{pi} - 1)/2 \). \( F_{pi} \) is the set of features representing the product \( p_i \) including \( nt_{pi} \) features, which is a selected subset of \( F \) (\( F \) includes all the features for the product line \( P \)). For instance, if \( p_i \) is represented by 50 features, all feature pairs covered by \( p_i \) are \( C^2_{50} = 1225 \). Notice that when calculating \( \text{Num}_{FP_{TSp_i}} \) and \( \text{Num}_{FP_{pi}} \), feature pairs only refer to valid feature pairs that are not violated against the defined cross-tree constraints. In our case, when obtaining a variant of feature model (selection of features in the FM_T) for testing a specific product \( p_i \), the defined cross-tree constraints on FM_T are automatically enforced using pure::variants (P::V) [14, 15], which prevents the selection of any invalid feature pairs. \( FPC \) value also ranges from 0 to 1 and a higher value shows better feature pairwise coverage. \( FPC \) value also ranges from 0 to 1 and a higher value shows better feature pairwise coverage.

**Fault Detection Capability (FDC)** is defined to measure the fault detection capability for the selected test suite. In our context, fault detection refers to the rate of successful execution for a test case in a given time, e.g., a week in our case. More specifically, the execution of a test case can be defined as a *success* if it can detect faults in a given time (a week in our case) and as a *fail* if it does not detect any fault. \( FDC \) can be calculated as below.

\[
    FDC_{TS_{pi}} = \frac{\sum_{i=1}^{nt_{pi}} SucR_{tc_i}}{nt_{pi}}
\]

\( TS_{pi} \) refers to the selected test suite including a set of test cases for testing the product \( p_i \) and \( nt_{pi} \) is the number of test cases included in \( TS_{pi} \), where \( 1 \leq nt_{pi} \leq nt \). \( SucR_{tc_i} \) is the successful rate of execution for test case \( tc_i \), which can be measured as below.

\[
    SucR_{tc_i} = \frac{\text{NumSuc}_{tc_i}}{\text{NumSuc}_{tc_i} + \text{NumFail}_{tc_i}}
\]

\( \text{NumSuc}_{tc_i} \) is the number of *success* for test case \( tc_i \) and \( \text{NumFail}_{tc_i} \) is the number of *fail* for test case \( tc_i \). For instance, a test case is usually executed 100 times per week in Cisco. So if it executes successfully for 80 times, the \( SucR \) is \( 80/100 = 0.8 \). Note that \( FDC \) value also ranges from 0 to 1 and a higher value represents better fault detection capability.
Notice that our goal is to systematically and automatically select a set of test cases that are cost-effective as compared to the manual approach. Our problem can be formalized as follows.

**Problem:** Select a subset $TS_{p_i}$ for testing the product $p_i$ to achieve the following two objectives as compared with the test cases ($TS_{man_{p_i}}$) obtained by the manual approach:

1) For Effectiveness: $FC_{TS_{p_i}} \geq FC_{TS_{man_{p_i}}}$, $FPC_{TS_{p_i}} \geq FPC_{TS_{man_{p_i}}}$ and $FDC_{TS_{p_i}} \geq FDC_{TS_{man_{p_i}}}$

2) For Cost: $Sel\_Time_{TS_{p_i}} \leq Sel\_Time_{TS_{man_{p_i}}}$

### 3.3 Solution Realization

At this stage, FM and CFM were proposed to tackle the test case selection problem from the research perspective. Afterwards, we came up with these two techniques (FM and CFM) to the test engineers at Cisco by organizing a workshop to provide a shorting training about FM and CFM. During the workshop, the test engineers were positive to adapt FM and CFM for solving their test case selection problem in practice since FM and CFM systematically model the product line and the relevant test case repository, which are not addressed in the current manual process. Moreover, by linking FM and CFM using restrictions, test cases corresponding to a specific product can be automatically obtained by performing feature selection in FM, which may shield the implementation details of test cases from test engineers, thereby reducing the selection effort (e.g., time). Therefore, FM and CFM were determined as a candidate solution for the test case selection problem based on the review of state-of-the-art. Notice that such candidate solution is designed based on the collaboration between researchers and test engineers at Cisco.

Specifically, FM was used to capture the commonalities and variabilities of VCS product line and each testing functionality can be modeled as a feature in FM. We built it for VCS product line together with the experienced test engineers to keep the candidate solution compatible with the real industrial environment surrounded. Notice that it took three months (around twelve meetings and each one took on average two hours) to build and refine a usable FM and such FM does not change significantly once built.

Moreover, CFM was proposed to model the test case repository (including more than 2000 test cases). Notice that the test case repository changes very frequently and it was
practically impossible to build CFM for the test case repository manually. To address such challenge, a tool called Import Plugin and Transformation (IPT) was designed and developed with aim to building CFM automatically (Section 5, which limited the expensive effort for building and maintaining CFM as clicking a button). Note that such tool was also designed together with the test engineers at Cisco with the objective of making it easier to adapt the IPT tool in the context of VCS product line testing.

3.4 Solution Evaluation

At this stage, our solution has been realized and we evaluated our solution by: 1) Performing an industrial case study (Section 6.1); 2) Conducting a questionnaire-based survey to solicit opinions from the industrial practitioners to obtain their feedback about the proposed solution (Section 6.2).

Notice that we also conducted evaluation for the steps of problem identification and formulation. More specifically, we presented the obtained results for each step (e.g., the identified test case selection problem) to the industrial practitioners (e.g., test engineers) and asked them to evaluate whether the results suited the real industrial context. Meanwhile, we also acquired their feedback for refining and improving the obtained results. The main objective for such evaluation is to ensure that there is no misunderstanding between researchers and industrial practitioners.

4 Methodology

In this section, we present our methodology that is based on Feature Model (FM) and Component Family Model (CFM) for automated test case selection. To exemplify our methodology, we first present a running example (Section 4.1), which is a simplified version of the Saturn product line of Cisco with a set of products (e.g., C20, C40, C60, C90, SX20, MX200 and MX300). Since our context is related with product line testing, we will call our FM as FM for Testing—FM_T and CFM as CFM for Testing—CFM_T. More specifically, FM_T is first presented to capture the commonalities and variabilities of a product line (Section 4.2) followed by CFM_T to capture the overall test structure of test cases (Section 4.3). Afterwards, we present how we perform test case selection for a
4.1 Running Example

The core functionality of a VCS is to establish a videoconference and the Saturn supports the following two types of videoconferences: Multi-way and Multi-site. A Multi-way call in VCS products means one VCS can dial at most to only one Endpoint (EP1) and put the current call on hold to dial to another Endpoint (EP2). The VCS can then switch between EP1 and EP2, but can have only one active call at a time. Compared with a Multi-way call, a Multi-site call allows users to make calls to more than one Endpoint simultaneously. In the current VCSs, some of them, e.g., C20 only supports Multi-way calls and others, e.g., C60, C90 and SX20 support Multi-site calls. Among products supporting Multi-site call, there is also a possibility of transmitting presentations in parallel to a videoconference using VCS products. Only one conference participant can send presentation at a time and all others can receive it. The Saturn supports two protocols for videoconference: H323 and SIP. Notice that each VCS product at Cisco on average has three million lines of C++ code.

To test Saturn, a testing repository including more than 2000 test cases is developed for various functionalities. For instance, the test case “Multi-way call test—max bandwidth” is designed and implemented to test the bandwidth of Multi-way call. Notice that each product is associated with a subset of test cases from the repository since it may not consist of all functionalities. Moreover, whenever a new functionality is introduced in the product line, new test cases are added into the repository.

4.2 Feature Model for Testing (FM_T)

In this section, we present how our FM_T models various testing functionalities of a product line and the relations among testing functionalities. Moreover, we provide the statistics of the current FM_T for Saturn.

Modeling Testing Functionalities using FM_T. Testing functionalities of a product line $P$ can be represented as $FM_T = \{f_1, f_2, f_3, ..., f_{nf}\}$, where $nf$ is the total number of features for $P$. As shown in Fig. 4, each testing functionality is associated with a feature $f_i$
in FM_T. For instance, the feature Multi-way is used to test the Multi-way call during conference meetings, and the Multi-site used to test the Multi-site call. Notice that the types of features in FM_T can be mandatory, optional, alternative and or as discussed in Section 2.1. For instance, as shown in Fig. 4, the feature Call is mandatory feature since each product must support call functionality and the feature Presentation is optional because not all products support the presentation functionality (e.g., C90 supports while C20 does not). The features Multi-way and Multi-site are alternative features since one product can only choose to support either Multi-way call or Multi-site call. SIP and H323 features are or features because one product can support at least one protocol for videoconference. Moreover, based on the expertise, testing of a VCS product requires the following information:

- Testing states such as “Ready” and “Standby”. The “Ready” state tells that a system is ready to be tested and “Standby” describes that the system needs some conditions or operations to wake up and transit into the “Ready” state;
- Testing functionalities such as “Multi-way” and “Multi-site”;
- Testing parameters such as “SIP”, “H323”.

![Fig. 4. An excerpt of FM_T](image)

In order to meet the VCS testing domain, our FM_T represents testing states, testing functionalities, and testing parameters as different dimensions of features, which describe testing states of VCS products, functionalities needed to be tested and parameters needed to be configured. Hence, FM_T in our context consists of three parent features, namely, \( F_{TS} \), \( F_{TF} \) and \( F_{TP} \), respectively,
i.e., FM_T can be divided into three parts \( FM_T = \{F_{TS}, F_{TF}, F_{TP} \} \). Each part consists of a list of relevant features:

\[
F_{TS} = \{f_{ts1}, f_{ts2}, f_{ts3}, \ldots, f_{nts} \}
\]

such as the features Ready and Standby,

\[
F_{TF} = \{f_{tf1}, f_{tf2}, f_{tf3}, \ldots, f_{ntf} \}
\]

such as the features Video Call and Presentation,

\[
F_{TP} = \{f_{tp1}, f_{tp2}, f_{tp3}, \ldots, f_{ntp} \}
\]

such as the feature Protocol (Fig. 4), where \( n_{ts}, n_{tf}, n_{tp} \) are the numbers of features belonging to \( F_{TS}, F_{TF} \) and \( F_{TP} \), respectively, and \( n_{ts} + n_{tf} + n_{tp} = n_f \). Notice all the features are identified and created together with the test engineers based on the domain knowledge and system information.

**Modeling Relations using FM_T.** A set of cross-tree constraints (\( CONS \) as defined in Section 3.2) is added to the FM_T since testing functionalities may be related to each other. For instance, the Presentation feature requires the Multi-site feature since one product cannot support the presentation functionality unless it supports the Multi-site call, then the constraint \( cons_k = require (Presentation, Multi-site) \) is assigned from the source feature Presentation to the target feature Multi-site (Fig. 4). Notice that these cross-tree relations are also identified and built together with the test engineers in Cisco.

**Summary for FM_T.** Various products within the product line can be represented by different sets of features, i.e., a specific product can be configured by selecting a subset of features in FM_T. Together with test engineers of Cisco, we developed the FM_T for Saturn, which contains 134 features (44 mandatory, 38 optional, 25 alternative and 27 or) and 35 require constraints in total. In addition, the designed FM_T can represent 25 valid products even though there are only seven commercial mature products in the market (e.g., C20). Notice that such FM_T is designed and built based on the VCS testing domain knowledge and thorough discussion with test engineers. Building and revising the FM_T took around three months including twelve meetings (each meeting took on average two hours) together with test engineers at Cisco. Throughout such process, the potential inconsistencies (e.g., dead features) have been manually checked and removed accordingly based on the domain knowledge. Besides, it is important to mention that building FM_T is one-time manual effort since the functionalities of Saturn doesn’t change significantly. Also note that the FM_T have been adapted into the test case selection process for VCS product line testing.
4.3 Component Family Model for Testing (CFM_T)

In this section, we first present how to model the structure of all test cases in the repository using CFM_T followed by how to link FM_T and CFM_T using restrictions. Finally, we provide the statistics about CFM_T developed for Saturn.

Modeling Test Structure using CFM_T. Test plans are usually composed of many test cases and test engineers spend significant amount of time organizing test cases within these plans. In order to model the structure of test cases and automatically obtain relevant test cases for test plans, we proposed a CFM_T to capture the overall structure of test cases in the repository.

First of all, we investigated the test structure in the context of Saturn. Based on the domain knowledge, we found that the test structure in VCS testing is composed by test tasks and test cases. A test task is a collection of test cases that has a common test resource requirement such as "Multi-way call" task and "Multi-site call" task. Each test case is a test script with a set of parameters for execution such as required software/hardware resources, which can be run on different products.

Our CFM_T is represented as \( CFM_T = \{c_1, c_2, c_3, ..., c_n \} \) comprising of a set of components, where \( n \) is the number of components. Each component represents a test task and can be hierarchically decomposed into parts representing various test cases \( c_i = \{p_{ai1}, p_{ai2}, p_{ai3}, ..., p_{ain} \} \), where \( in \) is the number of parts belonging to \( c_i \). Fig. 5 shows two components Multi-way call and Multi-site call in the CFM_T representing two test tasks “Multi-way call” and “Multi-site call”. Each component includes a set of parts, which represent relevant test cases. Fig. 5 also shows two parts Multi-way call test—max bandwidth and Multi-site call test—max bandwidth belonging to the two components, which represent two test cases “Multi-way call test—max bandwidth” and “Multi-site call test—max bandwidth” belonging to the two test tasks (the names of two parts in CFM_T are not completely shown in Fig. 5 due to space).

Meanwhile, each part consists of a set of attributes representing different information for testing: \( A_{pa_{ij}} = \{a_{ij1}, a_{ij2}, a_{ij3}, ..., a_{ijn} \} \), where \( ijn \) is the number of attributes belonging to \( pa_{ij} \). In particular, each part \( pa_{ij} \) in our current CFM_T consists of four attributes (Fig. 5), which can be categorized as two groups: 1) Attributes for tracing, more specifically,
testID is used to identify and trace test cases between CFM_T and the repository; and 2) Attributes for measuring effectiveness (Section 3.2), i.e., fault detection capability (FDC), average execution time (AET) and execution frequency (EF). Notice that all the information for attributes is available (they can be generated from the test database in Cisco automatically) and can be used for different purposes such as multi-objective test minimization [14]. Notice that we only focus on the test case selection using CFM_T in this paper but our CFM_T can be adapted for more testing purposes via assigned attributes, e.g., minimizing the number of test cases for testing a product [14].

![An excerpt of CFM_T](image)

**Fig. 5.** An excerpt of CFM_T

**Linking FM_T and CFM_T using Restrictions.** Afterwards, restrictions are assigned to components or parts, which constrain relations between components or parts in CFM_T and features in FM_T. As discussed in Section 2.2, in our case, restrictions can be formulated as $A \rightarrow B$, where $A$ refers to a feature (test functionality) in FM_T and $B$ corresponds to a particular component (test suite) or part (test case). This means the corresponding components and parts in CFM_T cannot be included into the final selection until the relevant features is selected in FM_T. Notice that each component $c_i$ or part $p_{aij}$ can be linked with one or more features in FM_T via restrictions (i.e., Each component or part can have any number of restrictions). A component or part cannot be included into the final selection for a product unless its restrictions evaluate to true. For instance, we assigned a restriction to the part Multi-way call test—max bandwidth to link this part with the feature Multi-way in the FM_T since the test case “Multi-way call test—max bandwidth” is developed to test the bandwidth of Multi-way call, i.e., during test case selection, the test case cannot be included into the final selection unless the feature Multi-way is in the selected set of features.
Summary for CFM_T. An initial version of CFM_T was built together with test engineers at Cisco so that test engineers can get familiarized with the notations of CFM_T. Later on, we developed a tool called Import Plugin and Transformation (IPT) that can build CFM_T automatically (Section 5) in the context of Cisco. Following the test structure of Saturn, a CFM_T was built automatically using the tool IPT. In general, 143 test tasks with 2374 test cases in the repository are modeled as 143 components including 2374 parts with 9496 attributes (test ID, FDC, AET and EF) in the CFM_T. Meanwhile, 7386 restrictions are assigned to relevant components or parts in the CFM_T, which are used to link with related features in the FM_T.

4.4 Process to Select Test Cases for a Product

Test case selection for a product has the following two steps: 1) Based on the domain expertise and system information, test engineers analyze the test requirements for the product; 2) According to the analyzed requirements, test engineers select a set of relevant features in FM_T. Afterwards, related components and parts in CFM_T will be selected automatically, i.e., a set of relevant test cases in the repository will be chosen automatically.

Fig. 6. An example of test case selection process for a product

Fig. 6. shows an example of test case selection process for a product and it has the following three main parts: 1) An excerpt of FM_T; 2) An excerpt of CFM_T and 3) Two
associated test tasks including a set of test cases respectively. For FM_T, there are two alternative features, namely Multi-way and Multi-site. Each product can only support either Multi-way call or Multi-site call. In CFM_T, the component Multi-way call and Multi-site call are linked with the feature Multi-way and Multi-site in FM_T via restrictions at the same time the corresponding test tasks are associated with the related components in CFM_T. For instance, since C90 supports the Multi-site call, test engineers need to select the Multi-site feature in FM_T and then the component Multi-site call will be selected automatically via restrictions defined in CFM_T. Meanwhile, the test task “Multi-site call” will be chosen automatically from the repository for testing the functionality Multi-site call.

4.5 Consistency Between Different Artefacts

At this stage, a set of artefacts has been defined, i.e., FM_T, CFM_T and a set of relationships have also been specified between various artefacts. Being specific, FM_T and CFM_T are built and maintained (e.g., addition, deletion and modification) by a commercial tool called pure::variants (P::V) and the IPT tool we developed (Section 5). Notice that our tool IPT can automatically build CFM_T together with restrictions and the traces from CFM_T to the test case repository (Section 5) and all the relationships between different artefacts can be automatically checked for consistency by IPT and P::V [13]. Notice that IPT is developed as a plugin to P::V. More specifically, we defined three types of constraints for consistency checking, which include: 1) Cross-tree relationships between features in FM_T to ensure that the configured products are valid; 2) Consistency rules to ensure that the relationships (Restrictions) between components/parts in CFM_T and features in FM_T are correctly established; and 3) Consistency rules to ensure that the relationships (Traces) between components/parts in CFM_T and test suites/test cases are correctly established. We summarize all artifacts and relevant relationships in Table 1.

The first column indicates a relationship from a source artifact to a target artifact, the second column shows the type of relationship, whereas the third column describes the exact model elements of artifacts that are related. The last column presents whether and how such relationships can be checked automatically for correctness. For instance, there are two types of cross-tree constraints used in our case (i.e., require and mutually exclusive), which can be specified between a source feature to a target feature and P::V
tool automatically checks them to ensure that the configured product doesn’t contain any invalid features. Notice that in our case, due to the nature of the case study, we only used these two types of cross-tree relationships, however in other contexts, other types of relationships can also be used and enforced. For instance, there is a require relationship between the Presentation feature and the Multi-site feature, which will be automatically checked by P::V tool for consistency when performing feature selection in FM_T (Fig. 4). Notice that for the require relationship, a source feature and the target feature can be all the four types of features (Table 1). As for a mutually exclusive relationship, similarly, a source feature or a target feature can also be any of the four types of features (Table 1).

Table 1. Consistency between different artefacts

<table>
<thead>
<tr>
<th>Artefact to Artefact</th>
<th>Relationship Type</th>
<th>Element to Element</th>
<th>Checking</th>
</tr>
</thead>
<tbody>
<tr>
<td>FM_T ↔ FM_T</td>
<td>Cross-tree Relationships</td>
<td>Mandatory Feature ↔ Mandatory Feature</td>
<td>A (IPT and P::V)</td>
</tr>
<tr>
<td>FM_T ↔ FM_T</td>
<td>Require</td>
<td>Mandatory Feature ↔ Mandatory Feature</td>
<td>A (IPT and P::V)</td>
</tr>
<tr>
<td>FM_T ↔ FM_T</td>
<td>Mutually Exclusive</td>
<td>Mandatory Feature ↔ Mandatory Feature</td>
<td>A (IPT and P::V)</td>
</tr>
<tr>
<td>CFM_T → FM_T</td>
<td>Restrictions</td>
<td>Component → Mandatory Feature</td>
<td>A (IPT and P::V)</td>
</tr>
<tr>
<td>CFM_T → Test Case Repository</td>
<td>Traces</td>
<td>Component → Test Suite</td>
<td>A (IPT)</td>
</tr>
</tbody>
</table>

↔: The relationships between source artefact and target artefact are bidirectional. →: The relationships between source artefact and target artefact are one way. A: automated checking by the relevant tools.
Moreover, various restrictions (Section 4.3) are assigned in CFM_T (i.e., components or parts), which associate components or parts in CFM_T with the four types of features in FM_T (Table 1). Notice that restrictions can only be assigned in CFM_T, i.e., such relations are one way from CFM_T to FM_T. We further defined a set of rules for checking the consistency from CFM_T to FM_T as shown in Table 2. All these defined rules were implemented in Java as part of the implementation of IPT and are automatically checked when establishing restrictions and traces. Notice that CFM_T together with relevant restrictions is built and checked for consistency by the IPT and P::V tool (Table 1). For instance, a restriction is assigned from the part Multi-way call test—max bandwidth in CFM_T to the Multi-way feature in the FM_T. Such restriction is automatically built and checked for consistency with respect to the above-defined rules (i.e., rule 1, 2, 4, 5 and 6 in Table 2).

Table 2. Rules for consistency checking in terms of restrictions

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>One component in CFM_T must be associated with at least one feature in FM_T, while a part may not be associated with any feature in FM_T</td>
</tr>
<tr>
<td>2</td>
<td>One feature must be associated with at least one component or part</td>
</tr>
<tr>
<td>3</td>
<td>When a feature is associated with one component, all the parts belonging to the component will be associated with the feature</td>
</tr>
<tr>
<td>4</td>
<td>When a feature is associated with one part, the corresponding component for the part is not necessarily associated with the feature</td>
</tr>
<tr>
<td>5</td>
<td>The source artifact for restrictions cannot be an attribute of component/part in CFM_T</td>
</tr>
<tr>
<td>6</td>
<td>The target artifact for restrictions cannot be the features that have conflicts based on the cross-tree constraints</td>
</tr>
</tbody>
</table>

Furthermore, traces are automatically built from CFM_T to the test case repository (Table 1) with the aim at checking the consistency between components/parts in CFM_T and test suites/test cases in the repository. Notice that in our case, a test suite contains a set of relevant test cases that are developed for testing similar VCS functionalities. We also defined a set of rules for checking the consistency of traces between CFM_T and the test case repository as shown in Table 3. All these rules are also implemented in Java as part of the implementation of IPT and are automatically enforced when building CFM_T. For instance, the “Multi-way call test—max bandwidth” test case in the repository is traced to the Multi-way call test—max bandwidth part in CFM_T and at the same time all the attributes of the test case are associated with the corresponding attributes of the part Multi-way call test—max bandwidth in CFM_T. Such trace is automatically checked by IPT according to the defined rules (i.e., rule 2, 4, 5 and 7 in Table 3).
Table 3. Rules for consistency checking in terms of traces

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>One test suite in the repository must be associated with only one specific component in CFM_T</td>
</tr>
<tr>
<td>2</td>
<td>One test case in the repository must be associated with only one specific part in CFM_T</td>
</tr>
<tr>
<td>3</td>
<td>Components in CFM_T can only be associated with test suites in the repository</td>
</tr>
<tr>
<td>4</td>
<td>Parts in CFM_T can only be associated with test cases in the repository</td>
</tr>
<tr>
<td>5</td>
<td>Attributes of test cases in the repository must be associated with the corresponding attributes of parts in CFM_T</td>
</tr>
<tr>
<td>6</td>
<td>When a test suite is associated with one component in CFM_T, the test cases belonging to the test suite should be associated with the corresponding parts of the component in CFM_T</td>
</tr>
<tr>
<td>7</td>
<td>When a test case is associated with one part in CFM_T, the relevant test suite for the test case is not necessarily associated with the corresponding component for the part</td>
</tr>
</tbody>
</table>

5 Automation

In this section, we present the tool support for our proposed methodology. Fig. 7 shows the detailed architecture of tool support for test case selection having three main parts, i.e., Front-end, Test Information and Tooling. More specifically, Front-end provides user interfaces to product line modeling for testing (i.e., Feature Model) and test selection. Test Information represents all the relevant test information (e.g., FM_T, CFM_T and test case information) and Tooling presents the tool support within test selection (i.e., Import Plugin and Transformation (IPT) tool).

As shown in Fig. 7, using the commercial tool P::V [13], we first developed the FM_T to capture testing commonalities and variabilities of a product line. Afterwards, a CFM_T is built to capture the overall structure of test cases (e.g., test case name, test script) and maintain all the links to the repository of test cases and to a FM_T. Test engineers only need to perform feature selection through the Test Selection Front-end for FM_T and all relevant test cases can be obtained automatically based on the links between FM_T and CFM_T, and between CFM_T and the repository Test Cases (Fig. 7). Finally, the selected test cases can be put into the test execution engine tool for testing System Under Testing (SUT) and the execution results are returned to the execution tool for further analysis.

Notice that the process of building a CFM_T is automated in our industrial application; however, a CFM_T may need to be built manually in other contexts. In our current industrial application (i.e., Cisco), the repository updates very frequently (e.g., new test cases are developed and existing test cases are modified) thereby it is not practical to build a CFM_T manually. Meanwhile, building CFM_T with a large number of restrictions requires too much effort (Section 4.3). To address this, we developed IPT tool for automatically building CFM_T, which used Java and Eclipse plugin techniques. The input
is the information of test cases in the repository such as test ID, fault detection capability, average execution time, execution frequency and tags associated with test cases. Such information can be automatically obtained as an xml file from the repository in Cisco. Notice that test engineers insert tags into the test cases based on the domain expertise while developing test cases to identify relevant testing functionalities. Notice that the inserted tags are part of a test case in our case and using tags, we can link features in FM_T with test cases in the repository for automated selection. For instance, the test case “Multi-way call test—max bandwidth” with test id 1268 (Fig. 7) is developed for testing the bandwidth of Multi-way call so that one tag named “Multi-way” is integrated into the test case to identify that the Multi-way call is tested by such test case. Based on the tag, a restriction can be built to link the part Multi-way call test—max bandwidth in CFM_T with the feature Multi-way in FM_T. Using tags information associated with test cases, our tool can build all relevant restrictions from CFM_T to FM_T automatically.

More specifically, IPT is comprised of four components, i.e., Test Case Parser, CFM_T Element Builder, Restriction Builder and CFM_T Assembler (Fig. 7). Test Structure Parser

Fig. 7. The detailed architecture for the IPT tool
is designed to parse the input of test case information to obtain the information used to build components, parts and related restrictions for CFM\_T. Afterwards, \textit{CFM\_T Element Builder} and \textit{Restriction Builder} are developed to create the elements for CFM\_T and restrictions using the output of Test structure parser. Finally, a \textit{CFM\_T assembler} is implemented to assemble the obtained elements and restrictions together and which outputs a complete CFM\_T.

6 Evaluation

In this section, we evaluate our methodology via: 1) reporting an industrial case study to demonstrate the benefits of applying our methodology in an industrial setting (Section 6.1); and 2) reporting results of a small scale questionnaire-based survey in Cisco with the objective of investigating the adoption of FM\_T and CFM\_T (Section 6.2) including related threats to validity (Section 6.3).

6.1 Industrial Case Study

Our case study is the \textit{Saturn} product line developed in Cisco [4]. The \textit{Saturn} family consists of various hardware codecs ranging from C20 to MX300. C20 is the lowest end product with minimum hardware and has lowest performance while MX300 is the highest end product with advanced hardware and highest performance.

\textit{Saturn} family consists of 20 subsystems such as audio and video subsystems. Each subsystem can run in parallel to the subsystem implementing the core functionality that deals with establishing videoconferences. To test such product line family, a large number of test cases (more than 2000) have been developed for various products. Each test case can be scheduled and executed on different platforms. All these test cases are stored in the \textit{Saturn} repository for test cases. When a specific product comes into play, it is required to choose a subset of relevant test cases from the repository and put them into execution after scheduling.

<table>
<thead>
<tr>
<th>Product</th>
<th>Selected Features</th>
<th>Selected Test Cases</th>
<th>Percentage of Selected Test Case</th>
<th>Sel-Time</th>
<th>Percentage of Reduced Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>C20</td>
<td>17</td>
<td>238</td>
<td>10.0</td>
<td>2.5 hours</td>
<td>92</td>
</tr>
<tr>
<td>C40</td>
<td>25</td>
<td>367</td>
<td>15.5</td>
<td>3 hours</td>
<td>91</td>
</tr>
<tr>
<td>C60</td>
<td>32</td>
<td>592</td>
<td>24.9</td>
<td>4.5 hours</td>
<td>86</td>
</tr>
</tbody>
</table>
Table 4 summarizes the results of test case selection for various products in *Saturn* using our proposed methodology. The *Selected Features* column indicates the number of selected features in FM_T for each product. The *Selected Test Cases* column shows the number of selected test cases by our proposed methodology. The *Percentage of Selected* column describes the percentage of selected test cases for a product among all the test cases in the repository. The *Sel-Time* and *Percentage of Reduced Time* columns show the required time for selection and the percentage of reduced time by our proposed methodology as compared with the current manual process. Notice that as discussed in Section 3.2, the selection time (*Sel-Time*) in our case includes the time for: 1) Identifying the testing requirements; 2) Manually or automatically selecting relevant test cases for testing the identified test functionalities.

Table 5 shows the results of comparing our methodology and the manual selection approach in terms of effectiveness for seven products in *Saturn*. The *FC* column shows the feature coverage for our methodology and the manual approach for each product, i.e., how many features can be covered by the selected test cases obtained by our methodology and the manual approach. The *FPC* column indicates the feature pairwise coverage for each product, i.e., how many features pairs can be covered using our methodology and the manual approach. The *FDC* column describes the fault detection capability achieved by our methodology and the manual selection approach. Notice that due to the confidential issues, only average values are shown for each measure in terms of *FC*, *FPC* and *FDC*, respectively for the manual approach. The analysis is however performed based on the values for each product. Notice that we also report a mean value for each column, e.g., on average 0.71 (71%) can be achieved for *FPC* (feature pairwise coverage) for each product.

Table 5. Summarized results of for comparison of effectiveness

<table>
<thead>
<tr>
<th>Product</th>
<th>Our Methodology</th>
<th>Manual Approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>C90</td>
<td>43</td>
<td>81</td>
</tr>
<tr>
<td>SX20</td>
<td>49</td>
<td>80</td>
</tr>
<tr>
<td>MX200</td>
<td>55</td>
<td>77</td>
</tr>
<tr>
<td>MX300</td>
<td>63</td>
<td>75</td>
</tr>
<tr>
<td>Mean</td>
<td>40.6</td>
<td>83</td>
</tr>
</tbody>
</table>
We further performed statistical tests to check whether the differences for the obtained results for the cost/effectiveness measures (Sel-Time, FC, FPC and FDC as shown in Table 4 and Table 5) are statistically significant. As suggested in [15], the Shapiro-Wilk test was first performed to test the normality of samples to select an appropriate statistical test for checking the differences. We chose the significance level of 0.05, i.e., a sample is normally distributed if the p-value is greater than 0.05. The calculated p-values for Sel-Time, FPC and FDC turned out to be 0.54, 0.29 and 0.63, respectively, suggesting that the samples are normally distributed.

<table>
<thead>
<tr>
<th>MD</th>
<th>p-value</th>
<th>MD</th>
<th>p-value</th>
<th>MD</th>
<th>p-value</th>
<th>MD</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>-26.6</td>
<td>&lt;0.000001</td>
<td>0.2</td>
<td>&lt;0.000001</td>
<td>0.01</td>
<td>0.11</td>
<td>0.02</td>
<td>0.21</td>
</tr>
</tbody>
</table>

*Sel-Time: Selection Time; FC: Feature Coverage; FPC: Feature Pairwise Coverage; FDC: Fault Detection Capability (defined in Section 3.2). MD: Main difference (our methodology minus manual approach). p: p-value. All the p-values less than 0.05 are identified as bold.

Based on [15], for samples with normal distribution, the t-test was further chosen to determine whether there were significant differences for the cost/effectiveness between manual and our proposed approach. We chose the significance level of 0.05, i.e., there is a significant difference if a p-value is less than 0.05. Table 6 summarizes the results of the t-test for the four cost/effectiveness measures. More specifically, the MD column shows the difference in mean values between our methodology and the manual approach with the aim of indicating the direction in which the results are significant. For Sel-Time, a negative value means the our methodology requires less time as compared with the manual approach, whereas for effectiveness measures, a positive value means our methodology is more effective than the manual one and vice versa. The p-value column shows the p-values obtained by the t-test. As shown in Table 6, MD for FC is 0.2 suggesting that our methodology achieves higher feature coverage than the manual approach. In addition, the
corresponding $p$-value is less than 0.00001 suggesting that in terms of $FC$, our methodology is significantly better than the manual approach.

**Abstraction and Automation.** FM_T captures various testing functionalities within the product line in a systematic way, whereas CFM_T provides an additional layer of abstraction on top of the low level details of the test cases in the repository. This additional layer of abstraction hides implementation of test scripts, test settings for execution (test setting files), and test files capturing required software/hardware resources from test engineers (test resource files). In the current practice, test engineers are required to go through all the test scripts, test setting files, and test resource files, to select a set of relevant test cases for a product. Using our methodology, a test engineer only selects a set of relevant features in FM_T for a product and corresponding test cases will be obtained from the repository automatically, which greatly reduces the complexity of the whole test case selection process in product lines. Notice CFM_T with restrictions is hidden from test engineers and built automatically by the IPT tool.

**Reduced Selection Effort.** Through discussions with test engineers in Cisco, we have learnt that: 1) The current practice of manual test selection takes at minimum two days; 2) Typically, two test engineers are involved in test selection; and 3) There is no systematic way to determine how many of testing functionalities are covered by the selected test cases.

From the *Percentage of Selected* column in Table 4, we can see that the percentage of relevant test cases for each product is low, e.g., 10% for C20. This means that effort is reduced since test engineers do not need to go through 90% of the test cases. Even for MX300 that is the most advanced VCS in the Saturn, the percentage of relevant test cases is around 41.8%. Meanwhile, from the *Percentage of Reduced Time* column (the percentage of reduced time is calculated as: $\left(1 - \frac{\text{the selection time by our methodology}}{2 \text{ working days} \times 2 \text{ persons}}\right) \times 100\%$ where 2 working days * 2persons = 2 * 8 * 2 = 32 hours (assuming minimum time required for test case selection using the current practice), we can see that the time required for test case selection using our methodology is reduced, e.g., 92% time for selection is reduced for C20 ((1- 2.5/32) * 100\% = 92\%). In total, on average 83\% time (26.6 hours) for
selection is reduced as compared to the current manual process \((1 - 5.4 \text{ hours on average}/32) \times 100\% = 83\%\). Based on the statistical analysis (Table 6), the difference for selection time is significant between our methodology and the manual approach since the \(p\)-value is much less than 0.05 (i.e., less than 0.00001). So we can conclude that our methodology significantly reduces test selection time as compared with the current manual approach. Notice that effort and time saved is at the expense of creating FM_T and CFM_T, but as we discussed in Section 4.2, developing FM_T is one time effort and CFM_T is built automatically in our context.

**Improved Effectiveness.** As shown in Table 5, with our methodology, selecting a set of relevant features in FM_T for a product ensures that all required testing functionalities are covered at least once with the selected corresponding test cases, i.e., feature coverage (FC) is 1 for all the seven VCS products. However, in the current practice, there isn’t any way to ensure such coverage for testing functionalities and the average FC achieved by the current manual approach is only 80% on average. In addition, Table 6 shows our methodology can significantly improve the feature coverage as compared with the manual approach. Similarly, as for FPC and FDC, Table 6 shows that our methodology can improve FPC and FDC as compared with the manual approach since the MD values are greater than 0 but not significantly since the \(p\)-values are greater than 0.05. In summary, our methodology can improve the current practice for test case selection as compared with the current manual selection approach, especially for the selection time (Sel-Time) and feature coverage (FC).

**Less Reliance on Domain Expertise.** The current test select practice largely depends on domain expertise of test engineers. This means that different groups of test engineers may obtain different sets of test cases based on their understanding for the same product. Moreover, most of test engineers in Cisco have been working for years in the testing group and thus understanding of testing functionalities and test cases in the repository is inside minds of several test engineers. Therefore, the current process lacks a unified understanding of testing functionalities and test cases in the repository. Because of this, when old test engineers leave, domain expertise of test selection is lost and training new test engineers require significant amount of effort. In contrast, using our methodology, FM_T captures all domain expertise for testing (testing functionalities) in a systematic way.
since it is built together with all the test engineers. Even training new test engineers is just limited to train them FM_T notations and the test engineers do not need to understand CFM_T.

**Improved Maintenance Effort.** In the current practice, there is no systematic way to maintain the testing functionalities and test cases for testing the VCS products. The maintenance for the current practice can be summarized from three main activities, including: 1) Addition: whenever a new functionality is introduced by a new product, the corresponding test cases are developed and added into the repository; 2) Deletion: when a testing functionality is removed, the corresponding test cases are not deleted from the repository; and 3) Modification: when a functionality is modified, the affected test cases are not deleted rather new test cases in the repository are added. Moreover, the current practice has to maintain the relations among test functionalities manually, which are error-prone, time-consuming and not systematic. However, using FM_T and CFM_T, we provide a systematic way to maintain testing functionalities and test cases and maintaining them is straightforward, which is presented as below.

**For Test Functionalities (FM_T).** 1) Addition: when a new functionality is introduced to the product line, it only requires adding a new relevant feature into the FM_T and corresponding cross-tree constraints if there exist specific relations between the newly introduced functionality and the existing testing functionalities; 2) Deletion: when a testing functionality is removed, test engineers only need to remove the old relevant feature and the relevant cross-tree constraints between the old features and the existing features from the FM_T; and 3) Modification: when the current functionality for the product line or the relations between testing functionalities is modified based on some specific requirements, it only needs to refine the related feature and the corresponding cross-tree constraints in FM_T based on the modified functionality and relations.

**For Test Case Repository (CFM_T).** In case of any addition, deletion or modification of test cases (test cases are not deleted from the test case repository in our context but it may happens in other contexts), the CFM_T can be rebuilt using IPT automatically, i.e., test engineers only require to export an updated XML file for the test case repository and importing such file to the IPT tool, and the CFM_T will be updated automatically.
In summary, if existing products evolve, only the affected parts in the FM_T need to be updated manually whereas CFM_T can be updated automatically with our IPT tool. Notice that the IPT tool establishes links from test cases to CFM_T automatically (Section 5).

**Adoption in Other Contexts.** To adapt our methodology in other contexts, FM_T and CFM_T are required to be built. FM_T can be built based on the domain expertise and system information for other product lines. Notice that building FM_T is one time effort and once it is built, it doesn’t require significant changes once new features are introduced in a product line. Similarly, CFM_T can be built manually or automatically. For example, in our industrial application, the tool IPT is developed to build CFM_T with restrictions automatically. In other contexts, it may not be feasible to build CFM_T with restrictions automatically. Therefore, a CFM_T with restrictions may have to be built manually, which is also one-time effort for a product line.

**Limitations of the Methodology.** Our methodology at its current stage has several limitations. Some of these are: 1) the current FM_T may not be complete since more detailed information for a product line is required to be added as features or cross-tree constraints into FM_T. However, notice that FM_T is for a product line and it will keep on evolving as more products are introduced into the product line; 2) in our current case, the restrictions between FM_T and CFM_T are determined by the integrated tags in the test cases. So the quality of test case selection largely depends on how well test engineers add relevant tags into the corresponding test cases; and 3) our methodology cannot deal with test case selection when test cases are bound to requirements and/or components at early stage (e.g., design and development), which requires further investigation for our proposed methodology.

**6.2 Questionnaire-Based Study**

We conducted a questionnaire-based study to solicit the views of the test engineers who were directly involved in the development of FM_T and CFM_T based. The questionnaire was conducted based on the reporting template defined by Wohlin [16].

**Planning and Design.** The FM_T and CFM_T have been designed together with the test
engineers, and CFM_T can be built automatically using the IPT tool (CFM_T may be built manually in other contexts). So it is essential to solicit opinions from the industrial practitioners about their experience for the FM_T and CFM_T, which is the main objective of this questionnaire. This questionnaire consists of two parts (i.e., FM_T and CFM_T) and the questions here were either multiple choices or required responses on a five-point Likert Scale. Notice that all relevant four people from the current testing team working with us have participated and filled out the questionnaire. Among the four participants, two of them are test managers and the other two are test engineers. Moreover, three of them have been working on Saturn for more than 5 years (the other one has been working for 2 years) and all of them have been involved into the discussion of our proposed methodology for at least five meetings.

**Results and Analysis for FM_T.** The objective of this section is to solicit the views of the participants on FM_T based on questions QA1-QA5. Each question along with its objective is presented as below.

- **QA1. It is easy to understand the notations of FM_T.** The objective of QA1 was to assess the difficulty of understanding the notations of FM_T since industrial practitioners are not usually familiar with modeling notations.

- **QA2. FM_T is sufficient to represent all functionalities of a VCS product line.** QA2 was asked to determine the sufficiency of FM_T notations for capturing the variabilities for Saturn.

- **QA3. It is easy to understand and use the provided tool for building FM_T.** QA3 was designed to solicit the opinions of participants in terms of required effort for building the FM_T using a provided commercial tool P::V.

- **QA4. It is easy to build and revise a FM_T for a VCS product line.** QA4 was asked to assess the difficulty of maintaining and revising the FM_T.

- **QA5. The functionalities of a VCS product line do not change significantly.** The objective of QA5 was to confirm whether the frequency of changes in functionalities of the Saturn since the FM_T is built manually and frequent and significant changes in functionalities do not warrant the use of FM_T.
As shown in Fig. 8, for QA1, all four participants agreed. For QA2, 1 participant strongly agreed and 3 participants agreed. For QA3 and QA4, 2 participants strongly agreed and 2 participants agreed. For QA5, 2 participants strongly agreed, 1 participant agreed and 1 participant had no opinion. Based on the above results, we conclude that the test engineers have already good understanding of FM_T notations and it is agreed the notations are sufficient to model testing functionalities of Saturn. Moreover, the FM_T is easy to build and maintain. Notice that a version of FM_T has already been used by the test engineers in Cisco.

Results and Analysis for CFM_T. This section consisted of four questions QB1-QB4, which were designed to solicit the participants’ views about CFM_T. Each question together with corresponding objective is presented as below.

- **QB1.** It is easy to understand the notations of CFM_T for VCSs. The objective of QB1 was to determine the difficulty of understanding the notations of CFM_T.

- **QB2.** A CFM_T is sufficient to represent test case structure. The objective of QB2 was to assess the sufficiency using notations of CFM_T to represent test case structure of the repository.

- **QB3.** It is easy to obtain the XML file from test database for representing the overall test case structure. QB3 was designed to assess the easiness of obtaining the input (Section 5) for the IPT tool to build CFM_T with restrictions automatically.
- **QB4.** It is easy to obtain and add tags information into the XML file representing the overall test case structure. QB4 was designed to solicit the views from test engineers related with the difficulty of usage of tags since our IPT tool will automatically build and maintain restrictions between FM_T and CFM_T based on tags.

![Fig. 9. Responses to the questions related with CFM_T](image)

As shown in Fig. 9, for QB1 and QB2, 1 participant strongly agreed and 3 participants agreed. For QB3 and QB4, two participants strongly agreed, 1 participant agreed and 1 participant had no opinion. Based on the obtained results, we can conclude that test engineers find the notations of CFM_T sufficient to represent test case structure of the repository. Notice that in our current context, it may not be important for test engineers to know the notations of CFM_T since it is built automatically. However, in other contexts, a CFM_T may not be built automatically and then it would be important to know the opinions of test engineers about the notations of CFM_T. The results also show that the CFM_T can be built easily using the tool IPT and the test engineers are positive about adopting CFM_T in their current practice for test case selection.

### 6.3 Threats to Validity

In this section, we discuss four types of threats to validity for the industrial case study and questionnaire-based study and how we address them.
**Internal Validity.** *Internal validity* threats exist when the outcome of results are influenced by internal factors and are not necessarily due to the application of the treatment being studied [16]. With respect to the industrial case study, one potential threat is that fault detection capability (*FDC*) in our context is represented by the success rate of test cases rather than the actual number of detected faults, which may not be an accurate measure of fault detection capability (Section 3.2). However, due to the confidentiality issues, we are not allowed to provide exact number of failures per product as it may be misinterpreted as the quality of a VCS product. Notice that we defined *FDC* together with test engineers at Cisco based on the domain knowledge of VCS testing.

Another possible threat to *internal validity* is that the involved practitioners may not have sufficient expertise for VCS testing and thus the obtained results for questionnaire have limited value. To address such validity, we involved all the testing group that we have been collaborating with, which includes: one QA manager with 13 years’ experience for testing and two test engineers working on VCS testing for 7 and 5 years respectively, and one new test engineer with 1 year expertise. Therefore, we conclude that the testing group is an expert group for VCS testing thereby guaranteeing the value for the questionnaire.

**External Validity.** *External validity* threats exist when the outcome of results are influenced by external factors and are not necessarily due to the application of the treatment being studied [17]. One of the main construct validity threats of our industrial case study is that we only compared seven VCS products, which is a small sample for the statistical analysis and results may not be generalized. However, we need to mention that all the existing products in *Saturn* have been included and we plan to involve another product line for the industrial case study in the future.

Moreover, the main *external threat* to validity of our questionnaire-based survey is that there were only four participants and thus the results cannot be generalized. However, it is important to mention that the testing group we are working comprises of four people and all of them answered the questionnaire. Of course, to generalize our results and methodology, we need to adopt our methodology to other testing groups in Cisco. Notice that the current testing group at Cisco we are working with is already using FM_T and the IPT tool.
Conclusion Validity. Conclusion validity threats are concerned with factors that can influence the conclusion that can be drawn from the results of the experiments. The most probable conclusion validity threat in our questionnaire-based study is due to the design and execution of the questionnaire, and the number of subjects involved in the survey (four people). For our industrial case study, in order to avoid drawing wrong conclusions from the results, we chose the appropriate statistical test, i.e., t-test, since our samples meet all its assumptions.

For the design of the questionnaire, we followed the commonly used reporting template defined by Wohlin [16] with the objective of avoiding bias and informality. For the execution, we distributed the questionnaire to each individual practitioner and asked them fill that independently without communicating with others. In this way, all the obtained results for each copy of the questionnaire can reflect the views of the practitioners based on their experience of using our proposed methodology. As for the number of subjects in our survey, we admit that it is not sufficient to obtain strong conclusions based on the feedback from only four test engineers (subjects). In addition, the involved people may be trained well already since all of them participated at the whole process for our proposed methodology (e.g., designing and building FM_T). However, as we discussed in the external validity threats, the whole testing group we have been collaborating includes four people and all of them have taken part into the survey. In addition, all of them are experienced in testing and thus their opinions are valuable for the survey. In addition, based on our experience, it is super expensive for conducting a controlled experiment in an industrial setting, for instance, it is very difficult to involve fresh test engineers for training of the methodology. However, we are currently conducting another industrial case study using the methodology, which will be investigated in the future work.

Construct validity. Construct validity threats are related to the degree to which the construct being studied is affected by experiment settings. For our industrial case study, we chose the cost/effectiveness measures that are comparable across the two processes being compared, i.e., the proposed automated methodology and the current manual selection approach.
A frequently observed threat on the questionnaire-based analysis is that the designed questions may not have practical value or may misguide the practitioners. To tackle this validity, we carefully designed the questions for the questionnaire for FM_T and CFM_T from three perspectives, i.e., understandability, completeness and usability for the proposed methodology, which are the main concerns for VCS testing based on the domain knowledge and expertise. Moreover, we also conducted two questions related with tool support (e.g., QB3 and QB4 in Section 6.2) since the automation always has high priority for industry.

7 Lessons Learnt

In this section, we discuss a set of lessons learnt in terms of adapting PLE in our industrial context. Notice that all these lessons learnt are based on our experience of the industrial collaboration with Cisco.

**LL1. Be involved in industry practice.** This lesson underlines the importance of participation in industry practice for researchers to identify and solve interesting problems. There is always a gap between industry and academia about understanding of problems and such a gap can be reduced when researchers spend some time working with the industrial practitioners. We observed that in industry, it is difficult for engineers to identify and explain their problems precisely. Sometimes, the problems identified by them are engineering problems rather than research problems. Notice that this lesson learnt is not specific to PLE and is applicable to any research project with industry.

In our case, the first author initially spent two weeks working in Cisco to talk with the industrial practitioners and obtained first-hand knowledge and experience about the current testing process. Such experience helped us to identify that the efficiency of testing VCSs can be improved by employing PLE since all the products share commonalities and have some variabilities.

**LL2. Learn industry terminology and map it to the PLE terminology.** Building the knowledge of relevant terminology from industry is important to solve a research problem, especially if the industrial practitioners are not aware of the context of their problems (e.g.,
product line). Our experience with Cisco revealed that even though they are developing a product line of VCSs (i.e., Saturn), they are not aware of product line terminology and hence not aware of existing PLE solutions. Meanwhile, when working with industrial practitioners, our experience suggests learning commonly used terminology from industry and mapping them to PLE terminology. Such mapping is important from two perspectives: 1) It eases the communication between industry and academia, thereby reducing the chances of misunderstanding between researchers and practitioners. For example, in our context, when providing background of FM to the practitioners, we mapped the term feature in FM to the term test functionality in VCS testing; 2) From a researcher’s perspective, such mapping helps using appropriate terms in research papers that is commonly understood by researchers in the community.

**LL3. Involve practitioners with different job roles for wider impact in industry.** It is important to involve practitioners with different job roles in a project ranging from management to technical level. The management level practitioners can ensure that required resources are available when needed and making sure that the project is at the right track. Moreover, they help spreading the research ideas to the other groups in a company. Involving technical practitioners guarantee that all technical questions related with specific domain can be answered properly. In general, involving practitioners with various job roles in industry helps spreading the ideas at various levels across different groups in a company.

In our case, the testing team we collaborated with is comprised of: 1) The QA manager whose main responsibility is to coordinate and ensure that the collaboration goes on track. The QA manager also helped spreading the solution to the other testing groups in Cisco; 2) Three test engineers who were involved from problem identification until solution evaluation. By having multiple practitioners involved from the same group ensured that the collaboration still continues even if some of the persons involved leave.

**LL4: Build a standard-based solution relying on the existing techniques.** Practitioners are always reluctant to adopt a radically different solution than the existing solution and also a solution that has less technical and tool support. Moreover, totally new tools and techniques always face challenges to be adopted in an industrial setting since industrial
practitioners cannot get fully convinced by the new techniques unless they have been well-evaluated that requires significant amount of effort in terms of time and money. Therefore, it is always recommended to develop a solution that is based on standards and existing techniques with good tool support and good technical support available.

In our case, we relied on the standard language, i.e., feature models for modeling VCS product line and built it together with test engineers at Cisco. Moreover, we chose the component family model for modeling the test case repository, which can be well integrated with FM. Notice that FM and CFM modeling is also well supported by the commercial tool P::V and in our case, CFM_T can be built automatically by IPT. Moreover, CFM is an intermediate model and test engineers do not need to actually manually modify it.

**LL5. PLE can be adapted to any phase of software development.** This lesson learnt emphasizes that PLE doesn’t need to be adapted to all the phases of software development life cycle, i.e., requirement, design, development and testing. In our context, we adapted PLE to the testing phase for our test selection problem.

**LL6. Features in FM can be mapped to different entities of artefacts.** It is well known in the community that features in FM are commonly used to represent the characteristics of product line for product configuration. However, in our case, we observed that features in FM can be used to represent the testing functionalities for a product line, i.e., features in the FM_T are associated with the corresponding test cases in the repository through the CFM_T so that the test cases related with a specific product can be automatically obtained by selection of features in the FM_T. This means that a feature in a FM can be mapped to different entities belonging to different artefacts from software development life cycle.

**LL7. Training FM to industrial practitioners is challenging.** In our context, there were very few people in industry knowing details for FM so it was essential to provide relevant training of FM for the industrial practitioners for successfully adapting PLE. Notice that such training is also valuable for the practitioners to get convinced by the proposed methodology since it is natural that people trust what they are familiar with. However, the training for FM also poses several challenges based on our experience, which include: 1) it
is usually difficult for the industrial practitioners to distinguish to what extent testing functionalities should be reflected in FM, i.e., which test functionalities should be modeled as features in FM; and 2) the testing process is usually complex with a large number of detailed testing functionalities so it is challenging to build a complete FM for a product line.

In our case, we arranged two workshops (each one lasted on average two hours) to provide the background knowledge of FM to the practitioners of the testing group at Cisco. The main objective for such workshops is to make the industrial practitioners familiar with the concepts of FM (e.g., different types of features and cross-tree constraints). Moreover, when building the FM\_T for the Saturn product line together with the test engineers, we investigated the available documentation of VCS systems carefully for clear understanding of the domain and listed all the potential testing functionalities. Afterwards, we discussed each specific testing functionality together with test engineers to assess whether it should be modeled in the FM\_T. To make the FM\_T precise and complete, it took three months for both researchers at Certus and test engineers at Cisco to build and revise the FM\_T for the Saturn product line. Notice that the FM\_T has already adapted in the current practice at Cisco and still keeps evolving when new products are introduced, the old products removed or the current products are modified (the maintenance for the FM\_T is discussed in Section 6.1).

**LL8. Configuration of products in terms of testing means selection of test cases for a product.** Configuration of products in a product line can have different meanings at different phases of software development life cycle, i.e., requirements, design, development, and testing. For instance, typically configuration of products for development means choosing source code corresponding to a product for reuse [3]. In our case, the configuration of products means selection of relevant test cases for testing a specific product. The philosophy behind is that PLE can be adapted in industry in a customized and flexible way.

**LL9: Tool support is a pre-requisite for successful adoption in industry.** Through the collaboration with Cisco, we learned that good tool support is a key concern for the industrial practitioners when adapting a new technique in their current practice.
Sometimes, they are more eager to know the automation support of the technique than the technique itself since users (test engineers in our case) prefer to click a button for accomplishing the required job rather than learning the underline technique. Therefore, when adapting PLE, it is crucial to provide the tool support for the industrial practitioners for the successful adoption.

In our case, we carefully studied several tools that support feature modeling such as Clafer (http://www.clafer.org), FeatureIDE (http://fosd.de/fide/) and P::V (http://pure-systems.com/). The reasons we chose P::V is due to its high usability, better stability and reliability, friendly user interface and more technical support available for the commercial tool.

8 Related Works

In this section, we discuss the related works in terms of product line testing, feature modeling, consistency checking and regression testing. Since our work can be categorized into product line testing, we first present the related works for product line testing (Section 8.1) followed by the existing works for feature modeling (Section 8.2) since feature model is adapted in our work for automated test case selection. We further discuss the related works for consistency checking (Section 8.3) since it is crucial to ensure different artifacts are consistent in the context of product line. Finally, we brief describe the related works for regression testing (Section 8.4) since there are similarities for test case selection between product line testing and regression testing.

8.1 Product Line Testing

Product line testing is a relatively new, but intense field of research since product line engineering has shown significant benefits [3, 18-21]. McGregor [3] presented a set of activities, which can be used to address testing individual assets (e.g., verification of consistency between requirements and specifications) and testing artefacts (e.g., test-case derivation and test suite design) that represent complete products in the context of product line. Muccini et al. [18] discussed a set of challenges and opportunities for adapting the existing techniques (e.g., regression testing) to product line architecture testing, e.g., regression testing techniques can be extensively reused for
product line testing but the main challenge is to capture the commonalities and variabilities for a product line systematically and identify the corresponding test cases, which can be reused. Cohen et al. [19] designed a relational model (e.g., orthogonal variability model) to capture variability in product line and further defined a family of cumulative coverage criteria for testing a given product line. Cohen’s study was also considered to adapt the existing combinatorial testing approaches in product line testing. Nebut et al. [20] proposed to use extended UML use case notations to capture functional variation points for product line at requirements level, which could automatically generate behavioral models thereby generating test cases for testing a specific product. Scheidemann [21] proposed an algorithm to identify and eliminate the invalid configurations from all possible ones thereby reducing the testing effort (e.g., time) and increasing the efficiency for product line testing. However, these works only provide guidelines and suggestions for product line testing and particularly do not provide any systematic and automated process for test case selection [6].

In another our previous work [22], we proposed a product line modeling and configuration methodology using Feature Model (FM) and Component Family Model (CFM) to support model-based test case generation. More specifically, a FM was defined to capture various test functionalities of a product line and a CFM was designed to model a behavioral model repository (e.g., state machines, class diagrams). By linking FM and CFM, relevant behavioral models can be selected and configured for a product line by selecting features in FM and configuring relevant attributes in CFM. Afterwards, the selected and configured models can be used to generate test cases for testing the product. As compared with [22], the work in this paper adapts the same models (FM and CFM) but with significantly different objective i.e., CFM is designed and developed to model the test case repository for automated test case selection rather than the behavioral model repository for test case generation. Moreover, different tool is designed and developed for automation support as compared with [22]. In addition, this paper also discusses the detailed research methods used for our collaboration and shares a set of lessons learnt in terms of adapting PLE in industrial practice.
8.2 Feature Modeling

Feature model plays a key role for variability management in the context of product line [2]. However, features in feature model (FM) are only symbols without being associated with other models such as modeling behavior [23]. To address such challenge, several techniques for mapping FMs to other models have been proposed in the existing literature [23-26]. In [23], Czarnecki and Antkiewicz proposed a template-based approach to map FMs to other models such as UML activity and class models mainly for requirement analysis. In [24-26], two similar approaches called FeatureMapper and VML* were proposed to map FMs to other models (e.g., use case models, activity models, and business process models) by building a mapping model or a family of languages to specify various relationships between features in FM and their realization in the other models for supporting model-driven development. These works are similar as our work, i.e., our work can be also considered mapping FM to component family model (CFM), which is designed to model a test case repository at a higher level of abstraction. However, as compared with the existing works, our objective is significantly different, i.e., we aim at obtaining a set of relevant test cases from the repository for testing a specific product. Second, CFM is designed to model the test case repository for automated test case selection in product line, which is not used in the existing literature.

Moreover, pure::variants (P::V) provides a modeling solution for product line software development using FM and CFM [13]. More specifically, the FM captures the commonalities and variabilities of a product line, whereas the CFM represents the concrete software architecture of the product line. By linking these two models via restrictions, a software product can be configured by selecting a set of features for the product in FM. We adapted the P::V’s approach to enable the automated reuse of test cases for product line testing. More specifically, the main differences between our methodology and P::V can be summarized as follows: 1) We designed a methodology dedicated to software testing, i.e., enable the reuse of test cases across different products in a product line, whereas P::V mainly supports the variability management and configuration of software products; 2) The IPT tool automatically builds restrictions from CFM_T to FM_T, and traces from CFM_T to test cases, whereas, in principle, such restrictions and traces have to be built manually in P::V; 3) CFM_T, in our case, models the test case repository for a product line containing
effective attributes associated with test cases (e.g., fault detection capability) to support test case selection. In contrast, P::V supports modeling software architecture of the product line using CFM and attributes are used to specify additional information.

In conclusion, our contribution in this paper concerns the innovative application of FM and CFM in an industrial setting for automated test case selection. According to our knowledge, such work has not yet been explored in the literature. In addition, we also report an original experience and a set of lessons learnt on the industrial adoption of PLE.

8.3 Consistency Checking

In terms of consistency checking, there are several existing works in the context of product line with the aim of ensuring that different artifacts are consistent, e.g., feature model and UML models [23, 27-31]. Typically, consistency checking relies on consistency rules that are specified in various languages, e.g., Object Constraint Language (OCL) [32], programming languages such as Java [29] and propositional logic [28]. Once these rules are specified, they can be checked using various tools such as OCL constraint solver, e.g., UMLtoCSP [33], Constraint Satisfaction Problem (CSP) solver e.g., Minion [34], Boolean Satisfaction Problem (SAT) solver, e.g., PicoSAT ([31]) and Eclipse-based consistency checker, e.g., Incremental Consistency Checker [29]. As for example, Czarnecki et al. [32] proposed an approach to verify templates of feature model based on the defined OCL constraints. The instance of the model template is checked whether it violates the defined OCL constraints. Thaker et al. [27] focused on the safe composition of products and proposed an automated approach to support creation of software implementation based on feature models. The proposed approach mapped feature model to propositional formulas and checked the consistencies based on the formulas using SAT solver. Vierhauser et al. [29] took variability models and the code base into consideration and classified five types of consistencies within variability models and code base. According to each classified type of consistency, a set of specific constraints were further defined, which were implemented as a consistency checker using Java with aims at checking each type of consistency between different levels of artifacts, e.g., between feature model and UML models.

As compared with the existing works, our work falls into the category of defining and enforcing consistency rules using programming languages (Java in our case) since our
rules are not very complex that can be automatically checked when building CFM_T. However, the main differences between our work and the existing works can be summarized as: 1) Our goal is different, i.e., we focus on automated test case selection for a product rather than deriving code of the implementation of a product; and 2) We implemented a set of rules to ensure the consistency between CFM_T and FM_T, and CFM_T and the test case repository. This aspect is different than the rest of the related works in the literature.

8.4 Regression Testing

Regression test selection aims at identifying a set of relevant test cases when changes are made to existing software [8]. Various types of such techniques are proposed in the literature, but mostly around the following two aspects: selection based on code changes [35-37] and selection based on specification changes [38, 39]. In addition, several thorough survey papers have been published in the literature [7, 8].

Although techniques for regression test selection have been evaluated in many previous works [40-42], there is no enough evidence to prove that these techniques still work well if being adapted in the context of product line. Comparing our work with regression testing, main effort for our methodology is spent on building reasonable models for product line and the structure of test cases, and making links between them. In contrast to regression testing, where focus is on testing changed functionality of an existing software system, our work is applicable when a new product is to be tested.

In summary, our main objective is to perform test case selection automatically thereby reducing the selecting effort using FM and CFM in the context of product line. To the best of our knowledge, existing works have not covered such an objective: applying FM and CFM in product line for supporting automated selection of test cases in practice.

9 Conclusion and Future Work

This paper presented a solution to an industrial test case selection problem of a Video Conferencing Systems (VCS) product line called Saturn developed by Cisco Systems, Norway. We solved the problem by proposing a product line methodology for automated test case selection with lower cost and higher effectiveness as compared to the current
manual process. The methodology included the three main parts: 1) Feature Model for Testing (FM_T) modeling variability and commonalities of a product line; 2) Component Family Model for Testing (CFM_T) modeling the structure of a test case repository; and 3) A tool called IPT to automatically build restrictions from CFM_T to FM_T and traces from CFM_T to test cases in the test case repository. With our methodology, test engineers select features in FM_T and the corresponding test cases are retrieved automatically from the repository thus hiding the unnecessary implementation details of test cases that are not required for test selection.

We evaluated our methodology in two ways. First, we applied our methodology to the seven commercial products in Saturn and compared our methodology with the current manual process. The results showed our methodology can significantly reduce test selection time and at the same time preserves better effectiveness as compared with the current manual process. Second, we conducted a questionnaire-based study to solicit the views of our proposed methodology from test engineers at Cisco. The results showed that the test engineers are positive about adapting our methodology in their current practice for improvement.

We also presented the tool support for the methodology, which has already been adapted in Cisco’s practice for test case selection. In addition, we shared a set of lessons learnt based on our industrial collaboration with the aim to provide guidance to other practitioners. In the future, we plan to adapt our methodology to other product lines and conduct a large-scale empirical study to assess the cost/effectiveness of our proposed methodology.

References

Automated Product Line Test Case Selection: Industrial Case Study and Controlled Experiment

Shuai Wang¹,², Shaukat Ali¹, Arnaud Gotlieb¹, and Marius Liaaen³

¹Certus Software V&V Center, Simula Research Laboratory, Norway
²Department of Informatics, University of Oslo, Norway
³Cisco Systems Inc., Norway
{shuai, shaukat, arnaud}@simula.no
marliaae@cisco.com

Abstract. Automated test case selection for a new product in a product line is challenging due to several reasons. First, the variability within the product line needs to be captured in a systematic way; second, the reusable test cases from the repository are required to be identified for testing a new product. The objective of such automated process is to reduce the overall effort for selection (e.g., selection time), while achieving an acceptable level of the coverage of testing functionalities. In this paper, we propose a systematic and automated methodology using a Feature Model for Testing (FM_T) to capture commonalities and variabilities of a product line and a Component Family Model for Testing (CFM_T) to capture the overall structure of test cases in the repository. With our methodology, a test engineer does not need to manually go through the repository to select a relevant set of test cases for a new product. Instead, a test engineer only needs to select a set of relevant features using FM_T at a higher level of abstraction for a product and a set of relevant test cases will be selected automatically. We evaluated our methodology via three different ways: 1) We applied our methodology to a product line of video conferencing systems called Saturn developed by Cisco and the results show that our methodology can reduce the selection effort significantly; 2) We conducted a questionnaire-based study to solicit the views of test engineers who were involved in developing FM_T and CFM_T. The results show that test engineers are positive about adapting our methodology and models (FM_T and CFM_T) in their current practice; 3) We conducted a controlled experiment with 20 graduate students to assess the performance (i.e., cost, effectiveness and efficiency) of our automated methodology as compared with the manual approach. The results showed that our methodology is cost-effective as
compared with the manual approach and at the same time its efficiency is not affected with the increased complexity of products.

**Keywords:** Test Case Selection, Product Line, Feature Model, Component Family Model

1. **Introduction**

Product line engineering (PLE) is a systematic process to capture commonalities and variability across a set of products belonging to a product line [1, 2]. PLE has demonstrated several benefits in both academia and industry including: reducing development time and cost, speeding up product time-to-market and reducing required modeling effort for Model-based Testing (MBT) through the mechanism of reuse [3, 4].

Test case selection is important for product line testing since the number of all possible products derived from the product line is very huge and it is difficult to obtain a set of relevant test cases for a specific product [5]. Efficient test selection strategies can reduce the effort (i.e., selection time) and at the same time improve the effectiveness, e.g., coverage of testing functionalities [6, 7]. In recent years, more and more researchers have spent significant effort on fully automated strategies for test case selection, which have proven to be efficient as compared to manual strategies [8].

Our industrial partner in the context of this work is Cisco Systems, Inc, Norway [9], which develops high quality product lines of Videoconferencing Systems (VCSs) [10]. The current test case selection practice at Cisco is to select test cases manually from a repository of test cases, whenever a new VCS is to be tested. Due to the increasing complexity of functionalities and diversity of VCS products, manual selection poses several challenges [11]. First, manual test case selection consumes a significant amount of time, which reduces the efficiency of testing; Second, the manual selection process is mainly driven by the expertise of test engineers and hence it is not an objective and repeatable process (i.e., different test engineers may select different sets of test cases for the same product). Third, manual selection may result in a set of test cases that has low effectiveness, e.g., coverage of testing functionalities (i.e., all required testing functionalities may not be covered by the selected test cases), because a focus has been placed on the testing of specific functionalities. Finally, no guidelines or methodology is provided to train recently hired engineers to select test cases. This means the current
practice of test selection largely depends on the expertise of test engineers and is not scalable when more VCSs are developed and are to be tested.

To cope with the above-mentioned challenges, we propose a systematic product line modeling methodology to support automated test case selection. As shown in Fig. 1, we developed a Feature Model for Testing (FM_T) to capture commonalities and variabilities of a product line and a Component Family Model for Testing (CFM_T) to capture the overall test structure of test cases in the repository. When a new product Product\textsubscript{i} needs to be tested, test engineers are only required to perform selection through the Test Selection Front-end for FM_T and the related test cases will be chosen automatically, based on the restrictions from CFM_T to FM_T, and traces from CFM_T to the test cases in the Product Line Test Case Repository. Notice that such restrictions and traces are automatically built using a tool we developed called Import Plugin and Transformation (IPT) that implements our methodology.

We applied our methodology on a commercial VCS product line with seven products of varying complexity. The results showed that our methodology is systematic and significantly reduces the test case selection time as compared with the manual approach, while at the same time maintaining high coverage of testing functionalities. In addition, we also solicited views of test engineers involved in the development of the methodology using a questionnaire-based survey and the results showed that the test engineers are positive about integrating the proposed methodology in their current practice. To further assess the performance of our automated methodology as compared with the manual approach, we report a controlled experiment involving 20 graduate students from Beijing University of Aeronautics and Astronautics (BUAA), Beijing, China. The performance is assessed from three different perspectives, i.e., cost, effectiveness and efficiency using

![Fig. 1. An overview of the proposed methodology](image-url)
seven commercial VCS products. The results of the experiment showed that our automated methodology is significantly cost-effective as compared with the manual approach. Moreover, the efficiency of our methodology is not significantly affected with the increased complexity of VCS products.

This paper is an extension of a conference paper [12]. The main differences as compared with the conference version includes: 1) Three additional VCS products are involved for the evaluation of industrial case study; 2) a controlled experiment including 20 graduate students is carefully designed to evaluate the cost, effectiveness and efficiency for our automated methodology as compared with the manual approach; and 3) additional analyses comparing the performance of experts and graduate students with the aim at collecting additional evidence. Moreover, we need to mention that our previous work [40] is also an extension of the conference paper [12] and the main differences as compared with [40] mainly include points 2) and 3) as discussed above. In addition, we further compared them in detail in the related work section (Section 8.1).

The rest of the paper is organized as follows: Section 2 provides a background to Feature Model and Component Family Model. Section 3 describes the running example used to exemplify our methodology. Section 4 proposes our methodology using FM_T and CFM_T. Section 5 discusses the tool support. Section 6 and Section 7 present evaluations using an industrial case study and controlled experiment, respectively. Section 8 discusses the related work and Section 9 concludes the paper.

2. Background

In this section, we briefly introduce feature model (Section 2.1), followed by related background of component family model (Section 2.2).

2.1. Feature Model (FM)

Feature modeling is a hierarchical modeling approach for capturing commonalities and variabilities in product line [1, 9]. FM can be represented as a 2-tuple \((\text{features, constraints})\) with four types of features, namely \textit{mandatory}, \textit{optional}, \textit{alternative} and \textit{or}. A \textit{mandatory} feature means it must be included if its father feature is included in the current selection. The selection of an \textit{optional} feature is optional even if its father feature is
included. A father feature with a set of alternative features describes that only one of the alternative features can be included if their father feature is included. A father feature with a set of or features means at least one of the or features is included if their father feature is included. In addition, FM contains cross-tree constraints which are supplementary relations among unrelated features. There are two kinds of such constraints, namely require and mutually exclusive constraints. A require relation among two features (a source and a target) means if the source feature is included into the current selection, the targeted feature must also be included. A mutually exclusive relation has the opposite meaning, saying that if the source feature is included then the target feature cannot be included into the current selection [11].

2.2. Component Family Model (CFM)

A CFM is used to represent how products are assembled and generated in a product line by modeling relations among software architectural elements [14]. CFM can be represented as a 4-tuple (components, parts, source elements, restrictions). Components are named entities organized into a tree-like structure that can be of any depth. Each component represents one or more functional elements of the products in product line (e.g. C functions, Java classes). Parts are named and typed entities. Each part belongs to a component and contains one or more source elements. A part can be associated with given programming language features, classes or objects, but it can also be associated with other key elements. A source element is an unnamed but typed entity. Source elements are usually used to determine how the source code for the specified element is generated. Restrictions specify conditions under which a component, part or source element may be excluded from a final selection [14, 15].

3. Running Example

In this section, we present a running example that will be used to exemplify our methodology (Section 4). The running example is a simplified version of the Saturn product line of Cisco with a set of products (e.g., C20, C40, C60, C90, SX20, MX200 and MX300). Notice that C20 is the lowest end product with minimum hardware and has
lowest performance while MX300 is the most advance end product currently with advanced hardware and best performance.

The core functionality of a VCS is to establish a videoconference and the Saturn supports the following two types of videoconferences: Multi-way and Multi-site. A Multi-way call in VCS products means one VCS can dial at most to only one Endpoint (EP1) and put the current call on hold to dial to another Endpoint (EP2). The VCS can then switch between EP1 and EP2, but can have only one active call at a time. Compared with a Multi-way call, a Multi-site call allows users to make calls to more than one Endpoint simultaneously. In the current VCSs, some of them, e.g., C20 only supports Multi-way calls and others, e.g., C60 and C90 support Multi-site calls. Among products supporting Multi-site call, there is also a possibility of transmitting presentations in parallel to a videoconference using VCS products. Presentations can be sent only by one conference participant at a time and all others receive it. The Saturn supports two protocols for videoconference: H323 and SIP.

To test Saturn, a testing repository including more than 2000 test cases is developed for various functionalities. For instance, the test case “Multi-way call test—max bandwidth” is designed and implemented to test the bandwidth of Multi-way call. Notice that each product is associated with a subset of test cases from the repository since it may not consist of all functionalities. Moreover, whenever a new functionality is introduced in the product line, new test cases are added into the repository.

4. Methodology

In this section, we present our methodology that is based on Feature Model (FM) and Component Family Model (CFM) for automated test case selection. Since our context is related with product line testing, we will call our FM as FM for Testing—FM_T and CFM as CFM for Testing—CFM_T. More specifically, FM_T is first presented to capture the commonalities and variabilities of a product line (Section 4.1) followed by CFM_T to capture the overall test structure of test cases (Section 4.2). Afterwards, we present how we perform test case selection for a product (Section 4.3).
4.1. Feature Model for Testing (FM_T)

In this section, we first present how to model testing functionalities of a product line using FM_T followed by how to model relations among testing functionalities using FM_T. Finally, we provide the statistics of the current FM_T for Saturn.

4.1.1. Modeling Testing Functionalities using FM_T

Testing functionalities of a product line $P$ can be represented as $FM_T = \{f_1, f_2, f_3, \ldots, f_{nf}\}$, where $nf$ is the total number of features for $P$. As shown in Fig. 2, each testing functionality is associated with a feature $f_i$ in FM_T. For instance, the feature Multi-way is used to test the Multi-way call during conference meetings, and the Multi-site used to test the Multi-site call. Notice that the types of features in FM_T can be mandatory, optional, alternative and or as discussed in Section 2.1. For instance, as shown in Fig. 2 (Exclamation marks represent mandatory features, question marks represent optional features, double-arrow marks represent alternative features and cross-line marks represent or features), the feature Call is mandatory feature since each product must support call functionality and the feature Presentation is optional because not all products support the presentation functionality (e.g., C90 supports while C20 does not). The features Multi-way and Multi-site are alternative features since one product can only choose to support either Multi-way call or Multi-site call. SIP and H323 features are or features because one product can support at least one protocol for videoconference. Moreover, based on the expertise, testing of a VCS product requires the following information:

- Testing states such as “Ready” and “Standby”. The “Ready” state tells that a system is ready to be tested and “Standby” describes that the system needs some conditions or operations to wake up and transit into the “Ready” state;
- Testing functionalities such as “Multi-way” and “Multi-site”;
- Testing parameters such as “SIP”, “H323”.

In order to meet the VCS testing domain, our FM_T represents testing states, testing functionalities, and testing parameters as different dimensions of features, which describe testing states of VCS products, functionalities needed to be tested and parameters needed to be configured. Hence, FM_T in our context consists of three parent features, namely, Testing States ($F_{TS}$), Testing Features ($F_{TF}$) and Testing Parameters ($F_{TP}$), respectively,
i.e., FM_T can be divided into three parts $FM_T = \{F_{TS}, F_{TF}, F_{TP}\}$. Each part consists of a list of relevant features: $F_{TS} = \{f_{ts1}, f_{ts2}, f_{ts3}, ..., f_{nts}\}$ such as the features Ready and Standby, $F_{TF} = \{f_{tf1}, f_{tf2}, f_{tf3}, ..., f_{ntf}\}$ such as the features Video Call and Presentation and $F_{TP} = \{f_{tp1}, f_{tp2}, f_{tp3}, ..., f_{ntp}\}$ such as the feature Protocol (Fig. 2), where $nts, ntf, ntp$ are the numbers of features belonging to $F_{TS}, F_{TF}$ and $F_{TP}$, respectively, and $nts + ntf + ntp = nf$. Notice all the features are identified and created together with the test engineers based on the domain knowledge and system information.

4.1.2. Modeling Relations using FM_T

A set of cross-tree constraints is added to the FM_T since testing functionalities may be related to each other. All the constraints can be represented as $CONS = \{cons_1, cons_2, cons_3, ..., cons_{ncons}\}$, where $ncons$ is the number of constraints. Each $cons_i$ can be either require or mutually exclusive, i.e., $cons_i$ can be represented as $cons_i = require(f_m, f_n)$ or $cons_i = exclusive(f_m, f_n)$, where $f_m$ is the source feature and $f_n$ is the target feature (Section 2.1). For instance, the Presentation feature requires the Multi-site feature since one product cannot support the presentation functionality unless it supports the Multi-site call, then the constraint $cons_k = require(Presentation, Multi-site)$ is assigned from the source feature Presentation to the target feature Multi-site (Fig. 2). Notice that these cross-tree relations are also identified and built together with the test engineers in Cisco.
4.1.3. Summary for FM_T

The various products can be configured by performing different selections of the features in FM_T, i.e., a specific product can be represented as a subset of features. Together with test engineers of Cisco, we developed the FM_T for *Saturn*, which contains 134 features (44 mandatory, 38 optional, 25 alternative and 27 or) and 35 require constraints in total. Besides, according to our discussion with test engineers, we need to mention that building FM_T is one-time manual effort since the functionalities of *Saturn* doesn’t change significantly.

4.2. Component Family Model for Testing (CFM_T)

In this section, we first present how to model the structure of all test cases in the repository using CFM_T followed by how to link FM_T and CFM_T using restrictions. Finally, we provide the statistics about CFM_T developed for *Saturn*.

4.2.1. Modeling Test Structure using CFM_T

Test plans are usually composed of many test cases and test engineers spend significant amount of time organizing test cases within these plans. In order to model the structure of test cases and automatically obtain relevant test cases for test plans, we proposed a CFM_T to capture the overall structure of test cases in the repository.

First of all, we investigated the test structure in the context of *Saturn*. Based on the domain knowledge, we found that the test structure in VCS testing is composed by test tasks and test cases. A test task is a collection of test cases that has a common test resource requirement such as "Multi-way call” task and "Multi-site call” task. Each test case is a test script with a set of parameters for execution such as required software/hardware resources, which can be run on different products.

Our CFM_T is represented as $CFM_T = \{c_1, c_2, c_3, ..., c_n\}$ comprising of a set of components, where $n$ is the number of components. Each component represents a test task and can be hierarchically decomposed into parts representing various test cases $c_i = \{pa_{i1}, pa_{i2}, pa_{i3}, ..., pa_{in}\}$, where $in$ is the number of parts belonging to $c_i$. Fig. 3 shows two components *Multi-way call* and *Multi-site call* in the CFM_T representing two test tasks “Muti-way call” and “Multi-site call”. Each component includes a set of parts, which
represent relevant test cases. Fig. 3 also shows two parts *Multi-way call test—max bandwidth* and *Multi-site call test—max bandwidth* belonging to the two components, which represent two test cases “Multi-way call test—max bandwidth” and “Multi-site call test—max bandwidth” belonging to the two test tasks (the names of two parts in CFM_T are not completely shown in Fig. 3 due to space).

![Figure 3. An excerpt of CFM_T](image)

Meanwhile, each part consists of a set of attributes representing different information for testing: \( A_{pa_{ij}} = \{a_{ij1}, a_{ij2}, a_{ij3}, \ldots, a_{ijn}\} \), where \( ijn \) is the number of attributes belonging to \( pa_{ij} \). In particular, each part \( pa_{ij} \) in our current CFM_T consists of four attributes (Fig. 3), which can be categorized as two groups: 1) Attributes for tracing, more specifically, testID is used to identify and trace test cases between CFM_T and the repository; and 2) Attributes for test minimization, i.e., fault detection capability (*FDC*), average execution time (*AET*) which is recorded by seconds and execution frequency (*EF*) which is recorded per week. For instance, *AET* and *EF* for the test case “Multi-way call test—max bandwidth” is 53 and 37, respectively, showing the average execution time of the test case is 53 seconds and the test case is executed 37 times per week on average. Moreover, *FDC* is defined as the success rate of a test case in a week. For instance, the *FDC* of the part *Multi-way call test—max bandwidth* is 0.63 (Fig. 3), which means the test case “Multi-way call test—max bandwidth” executes successfully by 63% in a week. Using *FDC*, the number of selected test cases for testing a product obtained by our methodology can be further minimized based on their fault detection capability using different mechanisms such as genetic algorithms [16]. All the information for attributes is available (they can be...
generated from the test database in Cisco automatically) and can be used for different purposes. Notice that we only focus on the test case selection using CFM_T in this paper but our CFM_T can be adapted for more testing purposes via assigned attributes, e.g., minimizing the number of test cases for testing a product [16].

4.2.2. Linking FM_T and CFM_T using Restrictions

Afterwards, restrictions are assigned to components or parts, which constrain relations between components or parts in CFM_T and features in FM_T. Notice that each component \(c_i\) or part \(pa_{ij}\) can be linked with one or more features in FM_T via restrictions (i.e., Each component or part can have any number of restrictions). A component or part cannot be included into the final selection for a product unless its restrictions evaluate to true. For instance, we assigned a restriction to the part Multi-way call test—max bandwidth to link this part with the feature Multi-way in the FM_T since the test case “Multi-way call test—max bandwidth” is developed to test the bandwidth of Multi-way call, i.e., during test case selection, the test case cannot be included into the final selection unless the feature Multi-way is in the selection set of features.

4.2.3. Summary for CFM_T

An initial version of CFM_T was built together with test engineers at Cisco so that test engineers can get familiarized with the notations of CFM_T. Later on, we developed a tool called Import Plugin and Transformation (IPT) that can build CFM_T automatically (Section 5) in the context of Cisco. Following the test structure of Saturn, a CFM_T was built automatically using the tool IPT. In general, 143 test tasks with 2374 test cases in the repository are modeled as 143 components including 2374 parts with 9496 attributes (test ID, FDC, AEC and EF) in the CFM_T. Meanwhile, 7386 restrictions are assigned to relevant components or parts in the CFM_T, which are used to link with related features in the FM_T.

4.3. Process to Select Test Cases for a Product

Test case selection for a product has the following two steps: 1) Based on the expertise knowledge and system information, test engineers analyze the test requirements for the product; 2) According to the analyzed requirements, test engineers select a set of relevant
features in FM_T. Afterwards, related components and parts in CFM_T will be selected automatically, i.e., a set of relevant test cases in the repository will be chosen automatically.

Fig. 4 shows an example of test case selection process for a product and it has the following three main parts: 1) An excerpt of FM_T; 2) An excerpt of CFM_T and 3) Two associated test tasks including a set of test cases respectively. For FM_T, there are alternative two features, namely Multi-way and Multi-site. Each product can only support either Multi-way call or Multi-site call. In CFM_T, the component Multi-way call and Multi-site call are linked with the feature Multi-way and Multi-site in FM_T via restrictions at the same time the corresponding test tasks are associated with the related components in CFM_T. For instance, since C90 supports the Multi-site call, test engineers need to select the Multi-site feature in FM_T and then the component Multi-site call will be selected automatically via restrictions defined in CFM_T. Meanwhile, the test task “Multi-site call” will be chosen automatically from the repository for testing the functionality Multi-site call.

**Fig. 4.** An example of test case selection process for a product

5. Automation

In this section, we present the tool support for our automated methodology. Our tool is implemented as Eclipse plugin in Java.

In our methodology, a CFM_T is required to be built to maintain all the links to the repository of test cases and to a FM_T. The process of building a CFM_T can be
automated or manually. In our current industrial application (i.e., Cisco), the repository updates very frequently (e.g., new test cases are developed and existing test cases are modified) thereby it is not practical to build a CFM_T manually. Meanwhile, building CFM_T with a large number of restrictions requires too much effort (Section 4.2).

To address such problem, the tool IPT is developed shown as Fig. 5, which automatically builds a CFM_T to capture the structure of a large number of test cases in the repository. The input of IPT is test case information such as test ID, fault detection capability, average execution time, execution frequency and tags associated with test cases. Such information can be automatically obtained as an xml file from the repository in Cisco. The Test Selection Front-end interface allows a test engineer to perform selection of features as discussed in Section 4.1. Notice that tags are used to identify testing functionalities of test cases. For instance, the test case “Multi-way call test—max bandwidth” with test id 1268 (Fig. 3) is developed for testing the bandwidth of Multi-way call so that one tag named “Multi-way” is integrated into the test case to identify that the Multi-way call is tested by such test case. Based on the tag, a restriction can be built to link the part Multi-way call test—max bandwidth in CFM_T with the feature Multi-way in FM_T. Using tags information associated with test cases, our tool can build all relevant restrictions from CFM_T to FM_T automatically.

6. Evaluation for Industrial Case Study

In this section, we evaluate our methodology via: 1) reporting an industrial case study to demonstrate the benefits of applying our methodology in an industrial setting; 2) reporting results of a questionnaire-based survey in Cisco with the objective of investigating the adoption of FM_T and CFM_T.
6.1. Industrial Case Study

Our case study is the *Saturn* product line developed in Cisco [9]. The *Saturn* family consists of various hardware codecs ranging from C20 to MX300. C20 is the lowest end product with minimum hardware and has lowest performance while MX300 is the highest end product with advanced hardware and highest performance.

*Saturn* family consists of 20 subsystems such as audio and video subsystems. Each subsystem can run in parallel to the subsystem implementing the core functionality that deals with establishing videoconferences. To test such product line family, a large number of test cases (more than 2000) have been developed for various products. Each test case can be scheduled and executed on different platforms. All these test cases are stored in the *Saturn* repository for test cases. When a specific product comes into play, it is required to choose a subset of relevant test cases from the repository and put them into execution after scheduling.

Table 1. Summarized results of test case selection for various products

<table>
<thead>
<tr>
<th>Product</th>
<th>Selected Features</th>
<th>Selected Test Cases</th>
<th>Percentage of Selected Test Case</th>
<th>Sel-Time</th>
<th>Percentage of Reduced Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>C20</td>
<td>17</td>
<td>238</td>
<td>10.0</td>
<td>2.5 hours</td>
<td>92</td>
</tr>
<tr>
<td>C40</td>
<td>25</td>
<td>367</td>
<td>15.5</td>
<td>3 hours</td>
<td>91</td>
</tr>
<tr>
<td>C60</td>
<td>32</td>
<td>592</td>
<td>24.9</td>
<td>4.5 hours</td>
<td>86</td>
</tr>
<tr>
<td>C90</td>
<td>43</td>
<td>739</td>
<td>31.1</td>
<td>6 hours</td>
<td>81</td>
</tr>
<tr>
<td>SX20</td>
<td>49</td>
<td>801</td>
<td>33.7</td>
<td>6.5 hours</td>
<td>80</td>
</tr>
<tr>
<td>MX200</td>
<td>55</td>
<td>943</td>
<td>39.6</td>
<td>7.5 hours</td>
<td>77</td>
</tr>
<tr>
<td>MX300</td>
<td>63</td>
<td>994</td>
<td>41.8</td>
<td>8 hours</td>
<td>75</td>
</tr>
<tr>
<td>Mean</td>
<td>40.6</td>
<td>667.8</td>
<td>28.1</td>
<td>5.4 hours</td>
<td>83</td>
</tr>
</tbody>
</table>

Table 1 summarizes the results of test case selection for various products in *Saturn* using our automated methodology. The *Selected Features* column indicates the number of selected features in FM_T for each product. The *Selected Test Cases* column shows the number of selected test cases by our automated methodology. The *Percentage of Selected Test Case* column describes the percentage of selected test cases for a product among all the test cases in the repository. The *Sel-Time* and *Percentage of Reduced Time* columns show the required time for selection and the percentage of reduced time by our automated methodology as compared with the current manual process. The selection time (*Sel-Time*) in our case includes the time for: 1) Identifying the testing requirements; 2) Manually or automatically selecting relevant test cases for testing the identified test functionalities.
6.1.1. Abstraction and Automation

FM_T captures various testing functionalities within the product line in a systematic way, whereas CFM_T provides an additional layer of abstraction on top of the low level details of the test cases in the repository. This additional layer of abstraction hides implementation of test scripts, test settings for execution (test setting files), and test files capturing required software/hardware resources from test engineers (test resource files). In the current practice, test engineers are required to go through all the test scripts, test setting files, and test resource files, to select a set of relevant test cases for a product. Using our methodology, a test engineer only selects a set of relevant features in FM_T for a product and corresponding test cases will be obtained from the repository automatically, which greatly reduces the complexity of the whole test case selection process in product lines. Notice CFM_T with restrictions is hidden from test engineers and built automatically by the IPT tool.

6.1.2. Reduced Selection Effort and Test Coverage

Through discussions with test engineers in Cisco, we have learnt that: 1) The current practice of manual test selection takes at minimum of two days; 2) Typically, two test engineers are involved in test selection; and 3) There is no systematic way to determine how many of testing functionalities are covered by the selected test cases.

From the Percentage of Selected column in Table 1, we can see that the percentage of relevant test cases for each product is low, e.g., 10% for C20. This means that significant effort is reduced since test engineers do not need to go through 90% of the test cases. Even for C90 that is the most advanced VCS in the Saturn, the percentage of relevant test cases is around 31%. Meanwhile, from the Percentage of Reduced Time column (the percentage of reduced time is calculated as: \(1 - \frac{\text{the selection time by our methodology}}{2 \text{ working days} \times 2 \text{ persons}} \) * 100% where 2 working days * 2 persons = 2 * 8 * 2 = 32 hours (assuming minimum time required for test case selection using the current practice), we can see that the time required for test case selection using our methodology is reduced significantly, e.g., 92% time for selection is reduced for C20 ((1 - 2.5/32) * 100% = 92%). In total, 83% time (26.6 hours) for selection is reduced as compared to the current manual process ((1 - 5.4 hours on average/32) * 100% = 83%). Notice that effort and time saved is at the expense of creating FM_T and CFM_T,
but as we discussed in Section 4, developing FM_T is one time effort and CFM_T is built automatically in our context.

With our methodology, selecting a set of relevant features in FM_T for a product ensures that all required testing functionalities are covered at least once with the selected corresponding test cases. However, in the current practice, there isn’t any way to ensure such coverage for testing functionalities.

6.1.3. **Less Reliance on Domain Expertise**

The current test select practice largely depends on domain expertise of test engineers. This means that different groups of test engineers may obtain different sets of test cases based on their understanding for the same product. Moreover, most of test engineers in Cisco have been working for years in the testing group and thus understanding of testing functionalities and test cases in the repository is inside minds of several test engineers. Therefore, the current process lacks a unified understanding of testing functionalities and test cases in the repository. Because of this, when old test engineers leave, domain expertise of test selection is lost and training new test engineers require significant amount of effort. In contrast, using our methodology, FM_T captures all domain expertise for testing (testing functionalities) in a systematic way since it is built together with all the test engineers. Even training new test engineers is just limited to train them FM_T notations and the test engineers do not need to understand CFM_T.

6.1.4. **Reduced Maintenance Effort**

In the current practice, there is no systematic way to maintain the functionalities and test cases for the product line. Whenever a new functionality is introduced to the product line, the corresponding test cases are developed and added into the repository and when a testing functionality is removed, the corresponding test cases are not deleted from the repository. When a functionality is modified, the affected test cases are not deleted rather new test cases in the repository are added. Using FM_T and CFM_T, we provide a systematic way to maintain testing functionalities and test cases and maintaining them is straight forward. For FM_T, a new feature is added into the FM_T when a new functionality is introduced to the product line, an old feature is removed from the FM_T when a testing functionality is removed and the related feature is refined in case the current
functionality is modified. For CFM_T, in case of any addition, deletion, or modification of test cases, the CFM_T can be rebuilt using IPT automatically. In summary, if existing products evolve, only the affected parts in the FM_T need to be updated and CFM_T is updated automatically with our IPT tool. Notice that the links from test cases to our CFM_T is also automatically done by our IPT tool (Section 5).

6.1.5. Adoption in Other Contexts

To adapt our methodology in other contexts, FM_T and CFM_T are required to be built. FM_T can be built based on the domain expertise and system information for other product lines. Notice that building FM_T is one time effort and once it is build, it doesn’t require significant changes once new features are introduced in a product line. Similarly, CFM_T can be built manually or automatically. For example, in our industrial application, the tool IPT is developed to build CFM_T with restrictions automatically. In other contexts, it may not be feasible to build CFM_T with restrictions automatically. Therefore, a CFM_T with restrictions may have to be built manually, which is also one-time effort for a product line.

6.1.6. Limitations of the Methodology

Our methodology at its current stage has several limitations. Some of these are: 1) the current FM_T may not be complete since more detailed information for a product line is required to be added as features or cross-constraints into FM_T. However, notice that FM_T is for a product line and it will keep on evolving as more products are introduced into the product line; 2) in our current case, the restrictions between FM_T and CFM_T are determined by the integrated tags in the test cases. So the quality of test case selection largely depends on how well test engineers add relevant tags into the corresponding test cases; and 3) our methodology cannot deal with test case selection when test cases are bound to requirements and/or components at early stage (e.g., design and development), which requires further investigation for our automated methodology.

6.2. Questionnaire-Based Study

We conducted a questionnaire-based study to solicit the views of the test engineers who were directly involved in the development of FM_T and CFM_T based. The questionnaire was conducted based on the reporting template defined by Wohlin [17].
6.2.1. Planning and Design

The FM_T and CFM_T have been designed together with the test engineers, and CFM_T can be built automatically using the IPT tool (CFM_T may be built manually in other contexts). So it is essential to solicit opinions from the industrial people about their experience for the FM_T and CFM_T, which is the main objective of this questionnaire. This questionnaire consists of two parts (i.e., FM_T and CFM_T) and the questions here were either multiple choices or required responses on a five-point Likert Scale. Notice that all relevant four people from the current testing team working with us have participated and filled out the questionnaire. Among the four participants, two of them are test managers and the other two are test engineers. Moreover, three of them have been working on Saturn for more than 5 years (the other one has been working for 2 years) and all of them have been involved into the discussion of our automated methodology for at least five meetings.

6.2.2. Results and Analysis for FM_T

The objective of this section is to solicit the views of the participants on FM_T based on questions QA1-QA5 (Table 2).

<table>
<thead>
<tr>
<th>Question</th>
<th>Strongly agree</th>
<th>Agree</th>
<th>No opinion</th>
<th>Disagree</th>
<th>Strongly disagree</th>
</tr>
</thead>
<tbody>
<tr>
<td>QA1</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>QA2</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>QA3</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>QA4</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>QA5</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

* QA1: It is easy to understand the notations of FM_T. QA2: FM_T is sufficient to represent all functionalities of a VCS product line. QA3: It is easy to understand and use the provided tool for building FM_T. QA4: It is easy to build and revise a FM_T for a VCS product line. QA5: The functionalities of a VCS product line do not change significantly.

The objective of QA1 was to assess the difficulty of understanding the notations of FM_T since industrial people are not usually familiar with modeling notations. For QA1, all four participants agreed. QA2 was asked to determine the sufficiency of FM_T notations for capturing the variabilities for Saturn. For QA2, 1 participant strongly agreed and 3 participants agreed. QA3 and QA4 were designed to solicit the opinions of participants in terms of required effort for building and maintaining the FM_T using a
provided commercial tool called Pure::Variants (P::V). For QA3 and QA4, 2 participants strongly agreed and 2 participants agreed. The objective of QA5 was to confirm whether the frequency of changes in functionalities of the *Saturn* since the FM_T is built manually and frequent and significant changes in functionalities do not warrant the use of FM_T. For QA5, 2 participants strongly agreed, 1 participant agreed and 1 participant had no opinion.

Based on the above results, we conclude that the test engineers have already good understanding of FM_T notations and it is agreed the notations are sufficient to model testing functionalities of *Saturn*. Moreover, the FM_T is easy to build and maintain. Notice that a version of FM_T has already been used by the test engineers in Cisco.

6.2.3. Results and Analysis for CFM_T

This section consisted of four questions QB1-QB4 (Table 3), which were designed to solicit the participants’ views about CFM_T.

QB1 and QB2 were asked to determine if notations of CFM_T is easy to understand and if the notations are sufficient to represent test case structure of the repository. For QB1 and QB2, 1 participant strongly agreed and 3 participants agreed. QB3 and QB4 were asked to assess the easiness of obtaining the input (Section 5) for the IPT tool to build CFM_T with restrictions automatically. For QB3 and QB4, two participants strongly agreed, 1 participant agreed and 1 participant had no opinion.

<table>
<thead>
<tr>
<th>Question</th>
<th>Strongly agree</th>
<th>Agree</th>
<th>No opinion</th>
<th>Disagree</th>
<th>Strongly disagree</th>
</tr>
</thead>
<tbody>
<tr>
<td>QB1</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>QB2</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>QB3</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>QB4</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

*QB1: It is easy to understand the notations of CFM_T for VCSs. QB2: A CFM_T is sufficient to represent test case structure. QB3: It is easy to obtain the XML file from test database for representing the overall test case structure. QB4: It is easy to obtain and add tags information into the XML file representing the overall test case structure.*

Based on the above results, we can conclude that test engineers find the notations of CFM_T sufficient to represent test case structure of the repository. Notice that in our current context, it may not be important for test engineers to know the notations of CFM_T since it is built automatically. However, in other contexts, a CFM_T may not be built automatically and then it would be important to know the opinions of test engineers about the notations of CFM_T. The results also show that the CFM_T can be built easily using
the tool IPT and the test engineers are positive about adopting CFM_T in their current practice for test case selection.

6.2.4. Threats to Validity

One of the main external threats to validity of our questionnaire-based survey is that there were only four participants and thus the results cannot be generalized. However, it is important to mention that the testing group we are working comprises of four people and all of them answered the questionnaire. Of course, to generalize our results and methodology, we need to adopt our methodology to other testing groups in Cisco. Notice that our FM_T and the tool IPT are already being used by the current testing group working with us in Cisco.

7. Evaluation for Controlled Experiment

Based on the evaluation performed based on the several products of an industrial case study and subjective assessment of the methodology by test engineers from Cisco as discussed in Section 6, we conclude that our automated methodology reduces the test selection effort as compared to the current practice. To further collect such evidence about automated methodology, we further conducted a controlled experiment with trained graduate students to compare cost, effectiveness, and efficiency of our methodology with the manual approach.

In this section, we will report the controlled experiment according to the template defined by Wohlin et al. [29]. First, we report experiment planning in Section 7.1 followed by the results and discussions in Section 7.2. Finally, we discuss four types of threats to validity in Section 7.3 related to the controlled experiment.

7.1. Experiment Planning

In this section, we first present goals, research questions and hypotheses (Section 7.1.1) followed by the background of the participants involved in the experiment (Section 7.1.2). Description of the material used for the experiment is presented in Section 7.1.3 and Section 7.1.4 presents the variables used to compare our automated methodology (Automated) and the manual approach (Manual). The training of participants is discussed
in Section 7.1.5 followed by the design of experiment (Section 7.1.6), and lastly the choice of statistical tests along with the justification is presented Section 7.1.7.

7.1.1. Goal, Research Questions and Hypotheses

The goal of our controlled experiment is to compare Automated with respect to cost, effectiveness, and efficiency of test case selection with Manual. Based on the goal of our controlled experiment, we will answer the following four research questions.

**RQ1:** Does Automated significantly reduce the Cost of test case selection as compared to Manual?

We wish to compare the Cost of Automated and Manual. None of the expected differences between them can a priori be certain to be in a specific direction thus leading to the definition of the following two-tailed null hypothesis:

\[ H_0^1 : \text{The Cost of test case selection with Automated is the same as Manual.} \]

**RQ2:** Does Automated significantly improve the Effectiveness of test case selection as compared to Manual?

In this research question, we compare the Effectiveness of Automated and Manual. Once again, none of the expected differences between them can a priori be certain to be in a specific direction thus leading to the definition of the following two-tailed null hypothesis:

\[ H_0^2 : \text{The Effectiveness of test case selection with Automated is the same as Manual.} \]

**RQ3:** Does Automated significantly improve the Efficiency of test case selection as compared to Manual?

We designed this research question with the aims of comparing the Efficiency of Automated and Manual. Same as RQ1 and RQ2, we defined the following two-tailed hypothesis:

\[ H_0^3 : \text{The Efficiency of test case selection with Automated is the same as Manual.} \]

**RQ4:** How does the increasing complexity of systems affect Cost, Effectiveness and Efficiency of Automated and Manual as compared to each other?

This research question focuses on studying the impact of increasing complexity of systems on Automated and Manual as compared to each other. The following two-tailed hypothesis is defined for this research question:
$H_0^4$: The impact of the increasing complexity of systems in terms of Cost, Effectiveness, and Efficiency on Automated is the same as Manual.

**RQ5: How does the performance of Experts compare with graduate students in terms of Cost, Effectiveness and Efficiency?**

In this research question, we aim at comparing the Cost, Effectiveness and Efficiency of Automated between the experts from the industrial case study and the graduate students from the controlled experiment. The following two-tailed hypothesis is defined for this research question:

$H_0^5$: The Cost, Effectiveness and Efficiency of the Experts with Automated are the same as the graduate students.

### 7.1.2. Participants

The controlled experiment was conducted at the Beijing University of Aeronautics and Astronautics (BUAA), Beijing, China. The participants in the experiment were 20 graduate students. All the participants are enrolled in the Master of Computer Software and Theory program. All the participants have taken one graduate course called “Software Engineering” and thus we can assume that all the participants have basic background of software engineering. In addition, on average each participant has taken two modeling courses and three testing courses. All the participants have at least one-year experience in the industry either as full time job or as training. Generally speaking, all the participants have roughly the same background.

The reason to choose such group of participants was to select the students with sufficient relevant background, i.e., modeling and testing such that they can be trained to use Automated and Manual in a limited amount of time. Having the participants with relevant background and sufficient training significantly increases the success of experiment. We also need to mention that it is very expensive and challenging to involve industrial practitioners for this kind of controlled experiment and this was the case for the test engineers in Cisco and therefore we resort to conducting the experiments with well-trained graduate students with relevant educational background.

Note that all the participants were free to decide whether they would like to participate this controlled experiment and they were also informed that the results of such controlled
experiment would have no effect on their grades. All the subjects were given the specific training related to Automated and Manual as discussed in Section 7.1.5.

7.1.3. Case Study System and Material

We used the same industrial case study for the experiments as for the evaluation based on the industrial case study reported in Section 6. The product line of the Videoconference Systems is called Saturn and has seven products, which are C20, C40, C60, C90, SX20, MX200 and MX300 (Section 6). The material for the experiment is divided into three main categories: 1) First, a detailed system specification for each product serving as testing requirements used for both Automated and Manual. Notice that these specifications are well documented for each product; 2) For Manual, information about test cases in the test case repository for testing Saturn as an XML file (Section 5). Notice that such XML file is used in the manual practice at Cisco and the purpose was to simulate the manual process as close to reality as possible; 3) For Automated, FM_T for Saturn (Section 4.1) together with IPT tool.

7.1.4. Variables

In this section, we present a set of variables including two independent variables and six dependent variables that we used for our controlled experiment. Table 4 summarizes the variables based on type, name, definition, research question (s) that the variable answers. Detailed description of each variable is also presented in this section.

**Table 4. Dependent and Independent Variables**

<table>
<thead>
<tr>
<th>Variable Type</th>
<th>Variable Name</th>
<th>Variable Definition</th>
<th>Mapping to RQ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Independent</td>
<td>Approach</td>
<td>Automated and Manual</td>
<td>RQ1-RQ5</td>
</tr>
<tr>
<td></td>
<td>Complexity</td>
<td>Number of features in a product</td>
<td>RQ4</td>
</tr>
<tr>
<td></td>
<td>Test Selection Time (Sel-Time)</td>
<td>Time to select test cases for a product based on testing functionalities</td>
<td>RQ1, RQ3, RQ4, RQ5</td>
</tr>
<tr>
<td>Dependent</td>
<td>Number of Covered Features (NCF)</td>
<td>Number of features covered by selected test cases for a product</td>
<td>RQ2-RQ5</td>
</tr>
<tr>
<td></td>
<td>Feature Coverage (FC)</td>
<td>Percentage of relevant features covered by selected test cases for a product. ( FC = \frac{NCF}{Total\ Number\ of\ Feature\ for\ the\ Product} )</td>
<td>RQ2-RQ5</td>
</tr>
<tr>
<td></td>
<td>Number of Selected Test Cases (NST)</td>
<td>Number of test cases selected for a product</td>
<td>RQ2-RQ5</td>
</tr>
<tr>
<td></td>
<td>Percentage of Selected Test Cases (PST)</td>
<td>Percentage of relevant test cases selected for a product. ( PST = \frac{(NST)}{(Total\ Number\ of\ Test\ Cases)} \times 100% )</td>
<td>RQ2-RQ5</td>
</tr>
<tr>
<td></td>
<td>NCF Efficiency</td>
<td>Number of features covered per unit of selection</td>
<td>RQ3-RQ5</td>
</tr>
</tbody>
</table>
### 7.1.5. Training

The first author of the paper trained all the participants spending overall two hours per approach, i.e., Manual and Automated. Training covered the following topics: 1) Introduction to the specifications of seven VCS products; 2) Introduction to feature modeling, i.e., notations and usage for FM_T; 3) Presentation of building CFM_T using our IPT tool; 4) Demonstration of Automated. Notice that we used dummy examples to explain the approach; 5) Presentation of the test case repository and the XML file capturing the structure of test cases in the repository; and 6) Demonstration of Manual.

After the training two groups of students were created randomly, Group 1 and Group 2, where Group 1 was assigned to use Automated and Group 2 to use Manual. As we discussed in Section 7.1.2, the participants have roughly the equivalent background and thus after equivalent training we assume that all participants had equivalent training for each approach.

### 7.1.6. Experiment Design

Table 5 summarizes the design of our experiment comprising of seven rounds (Round Column), where each round corresponds to test selection for one product in Saturn (Product Column). Each product has different complexity, which is measured in terms of the number of features. Notice that the products are ordered in terms of increasing complexity (# Features Column) to further study the learning curve. The columns four and five represent two pieces of information, i.e., the approach (Automated/Manual) used by a group and the number of data points collected for each approach per product. The maximum time for each round was set for 8.5 hours. Our design is a standard between-subjects design where different groups of subjects are compared when using Automated and Manual [29]. As shown in Table 5, Group 1 used Automated and the participants in this group were provided with FM_T for Saturn (Section 4.1), specifications of the seven VCS products, the IPT tool and the test case repository extracted as a XML file. The tasks for Group 1 can be summarized as: 1) build the CFM_T (Section 4.2) using the IPT tool;
and 2) analyze the detailed specification for each VCS product and select relevant features in FM_T. The participants in Group 2 used Manual were provided with specifications for the seven VCS products and the XML file that represent the test case repository. Tasks for Group 2 include: 1) analyze the specification for each VCS product; and 2) go through the entire XML file for test case repository and select the corresponding test cases. In addition, recall that the number of test cases in the repository developed for Saturn is 2374 (Section 4.2.3).

<table>
<thead>
<tr>
<th>Round</th>
<th>Product</th>
<th># Features</th>
<th>Group 1 (# Data Points)</th>
<th>Group 2 (# Data Points)</th>
<th>Maximum Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>C20</td>
<td>17</td>
<td>Automated (10)</td>
<td>Manual (10)</td>
<td>8.5 hours</td>
</tr>
<tr>
<td>2</td>
<td>C40</td>
<td>25</td>
<td>Automated (10)</td>
<td>Manual (10)</td>
<td>8.5 hours</td>
</tr>
<tr>
<td>3</td>
<td>C60</td>
<td>32</td>
<td>Automated (10)</td>
<td>Manual (10)</td>
<td>8.5 hours</td>
</tr>
<tr>
<td>4</td>
<td>C90</td>
<td>43</td>
<td>Automated (10)</td>
<td>Manual (10)</td>
<td>8.5 hours</td>
</tr>
<tr>
<td>5</td>
<td>SX20</td>
<td>49</td>
<td>Automated (10)</td>
<td>Manual (10)</td>
<td>8.5 hours</td>
</tr>
<tr>
<td>6</td>
<td>MX200</td>
<td>55</td>
<td>Automated (10)</td>
<td>Manual (10)</td>
<td>8.5 hours</td>
</tr>
<tr>
<td>7</td>
<td>MX300</td>
<td>63</td>
<td>Automated (10)</td>
<td>Manual (10)</td>
<td>8.5 hours</td>
</tr>
</tbody>
</table>

7.1.7. Overview of Statistical Tests

In this section, we provide the statistical analyses we conducted to analyze the experiment data together with the justification for the statistical tests [30, 38, 39]. The goal of selecting appropriate statistical tests is to provide sufficient evidence to determine whether we can accept or reject the null hypotheses stated in Section 7.1.1. To answer RQ1 and RQ2, we first report mean difference (MD) between Automated and Manual indicating the direction in which the results are significant. We further check whether the differences between Automated and Manual are statistically significant such that we can reject the null hypotheses [30]. As a first step to test the normality of distributions (i.e., Sel-Time, NCF, FC, NST, PST, Eff_F and Eff_TC) of the obtained samples, we used the Shapiro-Wilk test. We choose the significance level of 0.05, i.e., a sample is normally distributed if the p-value is greater than 0.05. The results of the Shapiro-Wilk test showed that the distributions do not strongly depart from normality, i.e., all the obtained samples for dependent variables are normal distributed. Based on these results, we chose the t-test as this test requires a distribution from a normal distribution. The significant level for t-test was chosen as 0.05 to compare Automated and Manual for the dependent variables discussed in Section 7.2.

In addition, we also double-checked the results of the t-test with a Wilcoxon Signed-Rank test, which is an equivalent, non-parametric test [30]. We report the results for both
tests and the results of the tests turned out to be consistent. We also report the mean differences between Automated and Manual indicating the direction in which the result is statistically significant.

To answer RQ4, we choose the Spearman’s rank correlation coefficient (ρ) to measure the relations between the results of dependent variables and complexity of VCS products [30]. The value of ρ ranges from -1 to 1, i.e., there is a positive correlation if ρ is closer to 1 and a negative correlation when ρ is closer to -1. A ρ close to 0 shows that there is no correlation between the two sets of data. Moreover, we also report significance of correlation using Prob>|p|, a value lower than 0.05 means the correlation is statistically significant.

Moreover, to address RQ5, one-sample t-test is performed to compare the results obtained by experts and graduate students when using Automated. The reason is that we compare results of a sample of students with exactly one value obtained for the industrial case study (Section 6).

7.2. Results and Discussions

In this section, we analyze and present the results of our experiment, i.e., the results for the four research questions are presented in Section 7.2.1 to Section 7.2.4, respectively. Afterwards, we provide an overall discussion in Section 7.2.4 and concluding remarks in Section 7.2.5 based on the obtained results and analysis for the controlled experiment.

7.2.1. Results and Analysis for RQ1

Recall that the purpose of RQ1 is to compare Automated and Manual in terms of Cost, i.e., time to select test cases (Sel-Time in Section 7.1.4). Table 6 shows the descriptive statistics including min, max, mean, median and standard derivation for the results of Sel-Time taken by Automated (Group 1) and Manual (Group 2) on the seven VCS products. Looking at the mean values in Table 6, we can observe that the participants working with Manual couldn’t manage to finish test case selection in the given amount of time, i.e., 8.5 hours. In contrast for Automated, the participants took on average 2.7 hours for C20 and maximum 8.1 hours on average for MX300. Based on the results, we can conclude that Automated took less time than Manual.

Table 6. Descriptive statistics for Cost, i.e., Sel-Time
To determine if the differences between *Automated* and *Manual* are statistically significant, the results of *t*-test and Wilcoxon tests are reported in Table 7. All the mean difference values are negative and *p*-values are less than 0.05 suggesting that *Automated* took significantly less time than *Manual*. Based on these results, we can conclude that Cost of test case selection measured as *Sel-Time* for *Automated* is significantly less than *Manual* and thus we can reject the null hypothesis $H_0$.

### Table 7. Two tailed *t*-test and Wilcoxon test for selection time (*Sel-Time*)

| Product | Mean Difference (MD) (Automated - Manual) | DF | *t*-value | *p*-value | Prob>|Z|
|---------|------------------------------------------|----|-----------|-----------|------------|
| C20     | -5.8                                     | 9  | -66.059   | <0.0001   | 0.002      |
| C40     | -5.2                                     | 9  | -81.054   | <0.0001   | 0.002      |
| C60     | -3.9                                     | 9  | -70.333   | <0.0001   | 0.002      |
| C90     | -2.2                                     | 9  | -31.728   | <0.0001   | 0.002      |
| SX20    | -2.1                                     | 9  | -39.735   | <0.0001   | 0.002      |
| MX200   | -0.9                                     | 9  | -10.819   | <0.0001   | 0.002      |
| MX300   | -0.4                                     | 9  | -9.775    | <0.0001   | 0.002      |

### 7.2.2. Results and Analysis for RQ2

RQ2 aims at assessing the Effectiveness (measured as *NCF, FC, NST* and *PST* defined in Section 7.1.4) as *Automated* as compared with *Manual*. Below we present the results based on each of the measure.

Table 8 reports the descriptive statistics based on the number of features (*NCF*) for *Automated* and *Manual*. For instance, C40 has 25 features and *Automated* on average can cover on average 23 features with median value and standard derivation as 23, 0.82, respectively. In contrast, *Manual* can cover on average 12.5 features with the median value of 13 and the standard derivation as 0.71. In general, we can see from Table 8 that for all the VCS products, *Automated* achieved higher *NCF* than *Manual* suggesting that *Automated* has higher Effectiveness in terms of *NCF* as compared with *Manual*. The results for statistical tests are reported in Table 9, where we can see that for all the products mean differences are positive and all the *p*-values are less than 0.05 suggesting that *Automated*
significantly achieved better Effectiveness measured as NCF. Based on the results, we can conclude that for all the VCS products, Automated can achieve significantly higher NCF than Manual.

Table 8. Descriptive statistics for Effectiveness, i.e., NCF

<table>
<thead>
<tr>
<th>Product</th>
<th>C20</th>
<th>C40</th>
<th>C60</th>
<th>C90</th>
<th>SX20</th>
<th>MX200</th>
<th>MX300</th>
</tr>
</thead>
<tbody>
<tr>
<td>G1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Min</td>
<td>15</td>
<td>7</td>
<td>2</td>
<td>11</td>
<td>29</td>
<td>41</td>
<td>61</td>
</tr>
<tr>
<td>Max</td>
<td>17</td>
<td>10</td>
<td>24</td>
<td>13</td>
<td>31</td>
<td>42</td>
<td>54</td>
</tr>
<tr>
<td>Mean</td>
<td>16.3</td>
<td>8.7</td>
<td>12.5</td>
<td>13</td>
<td>14</td>
<td>21.3</td>
<td>15.8</td>
</tr>
<tr>
<td>Std.</td>
<td>0.82</td>
<td>1.16</td>
<td>0.87</td>
<td>0.79</td>
<td>0.79</td>
<td>0.84</td>
<td>0.82</td>
</tr>
</tbody>
</table>

* G1: group 1; G2: group 2; Med.: Median; Std.: standard deviation

Table 9. Two tailed t-test and Wilcoxon test for number of covered features (NCF)

<table>
<thead>
<tr>
<th>Product</th>
<th>C20</th>
<th>C40</th>
<th>C60</th>
<th>C90</th>
<th>SX20</th>
<th>MX200</th>
<th>MX300</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean Difference (MD) (Automated - Manual)</td>
<td>7.6</td>
<td>0.5</td>
<td>15.8</td>
<td>20</td>
<td>23.4</td>
<td>31.5</td>
<td>37.3</td>
</tr>
<tr>
<td>DF</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>t-value</td>
<td>22.357</td>
<td>28.175</td>
<td>40.645</td>
<td>84.868</td>
<td>54.817</td>
<td>78.478</td>
<td>49.024</td>
</tr>
<tr>
<td>p-value</td>
<td>&lt;0.0001</td>
<td>&lt;0.0001</td>
<td>&lt;0.0001</td>
<td>&lt;0.0001</td>
<td>&lt;0.0001</td>
<td>&lt;0.0001</td>
<td>&lt;0.0001</td>
</tr>
<tr>
<td>Prob&gt;</td>
<td>Z</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 10. Descriptive statistics for Effectiveness, i.e., FC

<table>
<thead>
<tr>
<th>Product</th>
<th>C20</th>
<th>C40</th>
<th>C60</th>
<th>C90</th>
<th>SX20</th>
<th>MX200</th>
<th>MX300</th>
</tr>
</thead>
<tbody>
<tr>
<td>G1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Min</td>
<td>0.51</td>
<td>0.52</td>
<td>0.5</td>
<td>0.51</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
</tr>
<tr>
<td>Max</td>
<td>1</td>
<td>0.51</td>
<td>0.52</td>
<td>0.98</td>
<td>0.98</td>
<td>0.98</td>
<td>0.98</td>
</tr>
<tr>
<td>Mean</td>
<td>0.55</td>
<td>0.52</td>
<td>0.51</td>
<td>0.98</td>
<td>0.98</td>
<td>0.98</td>
<td>0.98</td>
</tr>
<tr>
<td>Std.</td>
<td>0.04</td>
<td>0.03</td>
<td>0.03</td>
<td>0.02</td>
<td>0.02</td>
<td>0.02</td>
<td>0.02</td>
</tr>
</tbody>
</table>

* G1: group 1; G2: group 2; Med.: Median; Std.: standard deviation

Table 11. Two tailed t-test and Wilcoxon test for feature coverage (FC)

<table>
<thead>
<tr>
<th>Product</th>
<th>C20</th>
<th>C40</th>
<th>C60</th>
<th>C90</th>
<th>SX20</th>
<th>MX200</th>
<th>MX300</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean Difference (MD) (Automated - Manual)</td>
<td>0.446</td>
<td>0.42</td>
<td>0.494</td>
<td>0.463</td>
<td>0.478</td>
<td>0.578</td>
<td>0.591</td>
</tr>
<tr>
<td>DF</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>t-value</td>
<td>21.867</td>
<td>28.175</td>
<td>42.360</td>
<td>103.244</td>
<td>55.988</td>
<td>83.044</td>
<td>46.902</td>
</tr>
<tr>
<td>p-value</td>
<td>&lt;0.0001</td>
<td>&lt;0.0001</td>
<td>&lt;0.0001</td>
<td>&lt;0.0001</td>
<td>&lt;0.0001</td>
<td>&lt;0.0001</td>
<td>&lt;0.0001</td>
</tr>
<tr>
<td>Prob&gt;</td>
<td>Z</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 10 shows the descriptive statistics for feature coverage (FC) for Automated and Manual. Based on the results, we observed that the FC values achieved by Automated are close to 1, i.e., feature coverage is closer to 100%. The observation is consistent with the industrial case study (Section 6.1). Moreover, we can also observe that FC obtained by Manual is low, for example, the highest mean value for FC is 0.51 while the lowest value is 0.37. The obtained results show that within the limited time constraint (e.g., 8.5 hours),
Manual has low FC achieved by the selected test cases. The results for statistical tests are reported in Table 11, where can see that all the MD values are positive and p-values are less than 0.05 suggesting that Automated achieved significantly better FC than Manual. The same results are observed when the observations are combined from all the products. In summary, we can conclude that Automated has significantly better Effectiveness in terms of FC when compared with Manual.

Table 12 shows the descriptive statistics for the number of selected test cases (NST) for Automated and Manual. From Table 12, we can observe that the NST for Automated is higher than Manual. For instance, for SX20, Automated obtained on average 801 test cases with high coverage (i.e., FC=0.96), while Manual obtained on average 188 test cases with low coverage (i.e., FC=0.49). Therefore, we can conclude that Automated can obtain more test cases with higher FC as compared with Manual. Moreover, we observed that NST with Manual is similar for all the seven products indicating that within the time constraint, the participants had roughly the same test case selection skills. The results for statistical tests are reported in Table 13, where all MD values are positive and all p-values are less than 0.05 suggesting that Automated selected significantly more test cases as compared with Manual.

Table 12. Descriptive statistics for Effectiveness, i.e., NST*

<table>
<thead>
<tr>
<th>Product</th>
<th>C20</th>
<th>C40</th>
<th>C60</th>
<th>C90</th>
<th>SX20</th>
<th>MX200</th>
<th>MX300</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min</td>
<td>224</td>
<td>172</td>
<td>328</td>
<td>358</td>
<td>747</td>
<td>156</td>
<td>206</td>
</tr>
<tr>
<td>Max</td>
<td>260</td>
<td>172</td>
<td>387</td>
<td>358</td>
<td>747</td>
<td>192</td>
<td>237</td>
</tr>
<tr>
<td>Mean</td>
<td>237</td>
<td>157</td>
<td>348</td>
<td>348</td>
<td>736</td>
<td>183</td>
<td>224</td>
</tr>
<tr>
<td>Std.</td>
<td>12.2</td>
<td>11.1</td>
<td>26.5</td>
<td>9.3</td>
<td>9.9</td>
<td>12.5</td>
<td>12.8</td>
</tr>
</tbody>
</table>

* G1: group 1; G2: group 2; Med.: Median; Std.: standard deviation

Table 13. Two tailed t-test and Wilcoxon test for number of selected test cases (NST)

| Product | Mean Difference (MD) (Automated - Manual) | DF | t-value | p-value | Prob>|Z| |
|---------|-----------------------------------------|----|---------|---------|------|
| C20     | 79.9                                   | 9  | 12.742  | <0.0001 | 0.002|
| C40     | 180.7                                  | 9  | 18.522  | <0.0001 | 0.002|
| C60     | 411.4                                  | 9  | 84.39   | <0.0001 | 0.002|
| C90     | 554.3                                  | 9  | 124.149 | <0.0001 | 0.002|
| SX20    | 613.2                                  | 9  | 164.565 | <0.0001 | 0.002|
| MX200   | 745                                    | 9  | 180.336 | <0.0001 | 0.002|
| MX300   | 786.7                                  | 9  | 178.3   | <0.0001 | 0.002|

Table 14 reports the descriptive statistics for PST with Automated and Manual. For instance, for C90, Automated covered on average 31% of all the test cases with median value and standard derivation as 31 and 0.38. On the contrary, Manual can select 7.7% of
all the test cases on average with the median value of 7.7 and the standard derivation as 0.5. From Table 14, we conclude that for all the products, Automated can cover more test cases than Manual. The results of statistical tests are presented in Table 15, where all the MD values are positive and all p-values are less than 0.05 suggesting that Automated can cover significantly more test cases as compared with Manual. Based on the results, we can conclude that Automated achieved significantly higher Effectiveness in terms of PST than Manual.

### Table 14. Descriptive statistics for Effectiveness, i.e., PST*

<table>
<thead>
<tr>
<th></th>
<th>C20</th>
<th>C40</th>
<th>C60</th>
<th>C90</th>
<th>SX20</th>
<th>MX200</th>
<th>MX300</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min</td>
<td>9.4</td>
<td>6.1</td>
<td>13.3</td>
<td>6.5</td>
<td>24.2</td>
<td>39.2</td>
<td>41.5</td>
</tr>
<tr>
<td>Max</td>
<td>11</td>
<td>7.2</td>
<td>16.3</td>
<td>7.1</td>
<td>25</td>
<td>39.2</td>
<td>43.2</td>
</tr>
<tr>
<td>Mean</td>
<td>10</td>
<td>6.6</td>
<td>14.7</td>
<td>7</td>
<td>24.6</td>
<td>33.7</td>
<td>39.6</td>
</tr>
<tr>
<td>Med.</td>
<td>11</td>
<td>7</td>
<td>14.3</td>
<td>7</td>
<td>24.6</td>
<td>33.7</td>
<td>39.6</td>
</tr>
<tr>
<td>Std.</td>
<td>0.5</td>
<td>0.5</td>
<td>1.12</td>
<td>0.38</td>
<td>0.4</td>
<td>0.51</td>
<td>0.42</td>
</tr>
</tbody>
</table>

*G1: group 1; G2: group 2; Med.: Median; Std.: standard deviation

### Table 15. Two tailed t-test and Wilcoxon test for percentage of selected test cases (PST)

| Product | Mean Difference (MD) (Automated - Manual) | DF | t-value | p-value | Prob>|Z| |
|---------|------------------------------------------|----|---------|---------|------|
| C20     | 3.4                                      | 9  | 12.555  | <0.0001 | 0.002|
| C40     | 7.61                                     | 9  | 18.488  | <0.0001 | 0.002|
| C60     | 17.34                                    | 9  | 84.566  | <0.0001 | 0.002|
| C90     | 23.34                                    | 9  | 123.317 | <0.0001 | 0.002|
| SX20    | 25.83                                    | 9  | 160.157 | <0.0001 | 0.002|
| MX200   | 31.38                                    | 9  | 186.206 | <0.0001 | 0.002|
| MX300   | 33.12                                    | 9  | 181.527 | <0.0001 | 0.002|

Based on the all four Effectiveness measures, we can conclude that Automated has significantly higher Effectiveness than Manual and thus we can reject the null hypothesis $H_0^2$.

### 7.2.3. Results and Analysis for RQ3

This research question focuses on evaluating the Efficiency of Automated and Manual. This research question aims at studying the trade-off between Cost and Effectiveness; for example, Cost may increase significantly at the same time the Effectiveness increases. Recall that for Efficiency, we defined two dependent variables, i.e., efficiency for features (Eff_F) and efficiency for test cases (Eff_TC) and the results based on these measures are presented below:

### Table 16. Descriptive statistics for Effectiveness, i.e., Eff_F*
Table 16 reports the descriptive statistics for $Eff_F$. For instance, for C40, **Automated** covers on average seven features per hour, whereas **Manual** covers on average 1.5. According to the obtained results, we can conclude that **Automated** has higher efficiency than **Manual** in terms of $Eff_F$. Table 17 shows the results of statistical tests for $Eff_F$, where we can see that all the MD values are positive and all the $p$-values are less than 0.05 suggesting that **Automated** has significantly higher Efficiency than **Manual**.

Table 17. Two tailed $t$-test and Wilcoxon test for efficiency for features ($Eff_F$)

| Product | Mean Difference (MD) (Automated - Manual) | DF | $t$-value | $p$-value | Prob>|Z|
|---------|----------------------------------------|----|------------|-----------|-----|
| C20     | 5.16                                   | 9  | 24.712     | <0.0001   | 0.002|
| C40     | 5.54                                   | 9  | 34.93      | <0.0001   | 0.002|
| C60     | 4.89                                   | 9  | 44.869     | <0.0001   | 0.002|
| C90     | 2.2                                    | 9  | 28.943     | <0.0001   | 0.002|
| SX20    | 4.58                                   | 9  | 56.281     | <0.0001   | 0.002|
| MX200   | 4.4                                    | 9  | 62.929     | <0.0001   | 0.002|
| MX300   | 4.74                                   | 9  | 50.786     | <0.0001   | 0.002|

Table 18 shows the descriptive statistics Efficiency measured as $Eff_TC$ for example, for MX300, on average 125 test cases can be obtained **Automated** per hour and 26 with **Manual**. From Table 18, we can conclude that for each product, **Automated** can obtain higher efficiency in terms of $Eff_TC$ as compared with **Manual**. The results of statistical tests are shown in Table 19, we can observe that all the MD values are positive and all the $p$-values are less than 0.05 suggesting that **Automated** has significantly higher Efficiency as compared to **Manual** in terms of $Eff_TC$.

Based on the results of the both Efficiency measures, we can conclude that **Automated** has significantly higher Efficiency than **Manual** and thus the null hypothesis $H_0$ can also be rejected.

Table 18. Descriptive statistics for Effectiveness, i.e., $Eff_TC^*$

<table>
<thead>
<tr>
<th></th>
<th>C20</th>
<th>C40</th>
<th>C60</th>
<th>C90</th>
<th>SX20</th>
<th>MX200</th>
<th>MX300</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>G1</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Min</td>
<td>72</td>
<td>17</td>
<td>90</td>
<td>18</td>
<td>120</td>
<td>19</td>
<td>109</td>
</tr>
<tr>
<td>Max</td>
<td>100</td>
<td>20</td>
<td>124</td>
<td>21</td>
<td>136</td>
<td>23</td>
<td>122</td>
</tr>
<tr>
<td>Mean</td>
<td>90</td>
<td>19</td>
<td>106</td>
<td>20</td>
<td>129</td>
<td>21</td>
<td>117</td>
</tr>
<tr>
<td><strong>G2</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Min</td>
<td>72</td>
<td>17</td>
<td>90</td>
<td>18</td>
<td>120</td>
<td>19</td>
<td>109</td>
</tr>
<tr>
<td>Max</td>
<td>100</td>
<td>20</td>
<td>124</td>
<td>21</td>
<td>136</td>
<td>23</td>
<td>122</td>
</tr>
<tr>
<td>Mean</td>
<td>90</td>
<td>19</td>
<td>106</td>
<td>20</td>
<td>129</td>
<td>21</td>
<td>117</td>
</tr>
<tr>
<td><strong>G1</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Min</td>
<td>10</td>
<td>2</td>
<td>11.9</td>
<td>1.25</td>
<td>5.1</td>
<td>1.4</td>
<td>4.6</td>
</tr>
<tr>
<td>Max</td>
<td>100</td>
<td>20</td>
<td>124</td>
<td>21</td>
<td>136</td>
<td>23</td>
<td>122</td>
</tr>
<tr>
<td>Mean</td>
<td>90</td>
<td>19</td>
<td>106</td>
<td>20</td>
<td>129</td>
<td>21</td>
<td>117</td>
</tr>
</tbody>
</table>

*G1: group 1; G2: group 2; Med.: Median; Std.: standard deviation*
Table 19. Two tailed t-test and Wilcoxon test for efficiency for test cases (Eff_TC)

| Product | Mean Difference (MD) (Automated - Manual) | DF | t-value | p-value | Prob>|Z| |
|---|---|---|---|---|---|
| C20 | 71.1 | 9 | 22.084 | <0.0001 | 0.002 |
| C40 | 85.9 | 9 | 22.01 | <0.0001 | 0.002 |
| C60 | 107.9 | 9 | 74.087 | <0.0001 | 0.002 |
| C90 | 95 | 9 | 63.412 | <0.0001 | 0.002 |
| SX20 | 103 | 9 | 109.248 | <0.0001 | 0.002 |
| MX200 | 100.7 | 9 | 66.224 | <0.0001 | 0.002 |
| MX300 | 98.2 | 9 | 107.145 | <0.0001 | 0.002 |

7.2.4. Results and Analysis for RQ4

Recall that the goal for RQ4 is to assess how the complexity of products (Complexity) can influence the performance of Automated and Manual in terms of Cost (i.e., Sel-Time), Effectiveness (i.e., NCF, FC, NST and PST) and Efficiency (i.e., Eff_F and Eff_TC). Notice that Manual was terminated within the maximum allocated time, i.e., 8.5 hours and thus we cannot obtain the actual selection time (Sel-Time) taken by Manual. Therefore, we assume that the selection time taken by Manual is 8.5 hours for all the VCS products and assess the cost, effectiveness and efficiency for Manual with the increasing complexity of products in this section.

To address RQ4, we calculated the spearman’s rank correlation between the mean values for each dependent variable (i.e., Sel-Time NCF, FC, NST, PST, FC Eff_F and Eff_TC) and Complexity. Recall that the numbers of features for all the seven VCS products are 17, 25, 32, 43, 49, 55, 63, respectively as shown in Table 20, which also shows that all the mean values for each dependent variable for both approaches. Notice that for each dependent variable, two numbers are shown in each cell of the table split by a slash. The first number shows the value obtained by Automated whereas the second number shows the value achieved by Manual.

Table 20. Mean values of each dependent variable for Automated and Manual with varying complexity (different number of features)

<table>
<thead>
<tr>
<th>Product</th>
<th>Number of Feature</th>
<th>Sel-Time</th>
<th>NCF</th>
<th>FC</th>
<th>NST</th>
<th>PST</th>
<th>Eff_F</th>
<th>Eff_TC</th>
</tr>
</thead>
<tbody>
<tr>
<td>C20</td>
<td>17</td>
<td>2.7/8.5</td>
<td>16.3/8.7</td>
<td>0.96/0.51</td>
<td>237/57</td>
<td>10/6.6</td>
<td>6.2/1</td>
<td>90/19</td>
</tr>
<tr>
<td>C40</td>
<td>25</td>
<td>3.8/8.5</td>
<td>23/12.5</td>
<td>0.92/0.5</td>
<td>348/167</td>
<td>14/7.7</td>
<td>7/1.5</td>
<td>106/20</td>
</tr>
<tr>
<td>C60</td>
<td>32</td>
<td>4.6/8.5</td>
<td>30/14</td>
<td>0.93/0.44</td>
<td>586/175</td>
<td>24/7.7</td>
<td>6.5/1.6</td>
<td>129/21</td>
</tr>
<tr>
<td>C90</td>
<td>43</td>
<td>6.3/8.5</td>
<td>41.3/21.3</td>
<td>0.96/0.5</td>
<td>756/182</td>
<td>31/7.7</td>
<td>6.5/2.5</td>
<td>117/22</td>
</tr>
<tr>
<td>SX20</td>
<td>49</td>
<td>6.3/8.5</td>
<td>47.24</td>
<td>0.96/0.49</td>
<td>801/188</td>
<td>33/7.7</td>
<td>7.4/2.8</td>
<td>125/22</td>
</tr>
<tr>
<td>MX200</td>
<td>55</td>
<td>7.6/8.5</td>
<td>52.3/20.8</td>
<td>0.95/0.38</td>
<td>940/195</td>
<td>39/6.8</td>
<td>6.9/2.5</td>
<td>124/23</td>
</tr>
<tr>
<td>MX300</td>
<td>63</td>
<td>8.1/8.5</td>
<td>60.4/23.1</td>
<td>0.96/0.37</td>
<td>1011/224</td>
<td>42/6.9</td>
<td>7.5/2.7</td>
<td>125/26</td>
</tr>
</tbody>
</table>
Table 21. Spearman’s correlation analysis for Automated and Manual with the increasing complexity

| Dependent Variables | Spearman’s $\rho$ | $\text{Prob}>|\rho|$ |
|---------------------|-------------------|-----------------|
| Sel-Time            | 0.991/null        | <0.001/null     |
| NCF                 | 1/0.775           | <0.001/0.06     |
| FC                  | 0.296-0.865       | 0.52/0.012      |
| NST                 | 1/1               | <0.001/<0.001   |
| PST                 | 1/1               | <0.001/<0.001   |
| Eff_F               | 0.703-0.865       | 0.08/0.10       |
| Eff_TC              | 0.577/0.463       | 0.18/0.13       |

Table 21 summarizes the results for Automated and Manual. Similarly as in Table 20, each cell includes two numbers split by a slash. The first number shows the results for spearman’s test related with Automated while the second number shows the relevant results for Manual. Notice, that we cannot get the value for Spearman’s $\rho$ for Manual in terms of Sel-Time since the selection time for each product is the same (i.e., 8.5 hours). From Table 21, we can observe that for Automated, there is a positive trend for all the variables with Complexity since all the $\rho$ values are greater than 0. Moreover, there is a significant positive trend for Sel-Time, NCF, NST and PST since all the $\text{Prob}>|\rho|$ are less than 0.001 indicating that the values for these variables significantly increase with the increase in the number of features. On the contrast, for Manual, there is also a positive trend for all the variables except for FC with the increasing complexity since all the values for spearman $\rho$ is greater than 0. In addition, there is a significant positive trend for NST and PST since the values for $\text{Prob}>|\rho|$ are less than 0.001. However, for FC, there is a significant negative trend with the growth of complexity, which shows that FC significantly decreases when the complexity increases. Notice that in terms of efficiency for features and test cases (Eff_F and Eff_TC), the positive trend is not statistically significant for both Automated and Manual. Based on the results, we can conclude that Automated can preserve high efficiency even with varying complexity of products. Therefore, we can reject the null hypothesis $H_0^4$ since Automated can preserve high performance (cost, effectiveness and efficiency) as the increasing complexity of VCS products while Manual significantly reduces the effectiveness in terms of FC.

7.2.5. Results and Analysis for RQ5

To assess the performance of students and the experts, i.e., test engineers in terms of using Automated, we further compared the results of the industrial case study reported in Section 6.1 with the results of students reported in this controlled experiment. We used the
same statistical tests as the ones we used for the controlled experiment except that in this case we used one-sample tests since we are comparing the results of a group of students with one data point from the industrial case study. The statistical tests are performed based for the seven products in terms of all the dependent variables (i.e., *Sel-Time*, *NCF*, *FC*, *NST*, *PST*, *Eff_F* and *Eff_TC*). Notice that for the results of experts, we use one-value sample based on Table 1, e.g., for C20, the one-value samples for each variable are 2.5 hours for *Sel-Time*, 17 features for *NCF*, 1 for *FC*, 238 test cases for *NST*, 10% for *PST*, 6.8 (17/2.5=6.8) for *Eff_F* and 95 for *Eff_TC* (238/2.5=95), respectively. The results of this comparison showed that experts outperformed graduate students since the value for mean difference (MD calculated by Mean (Expert)-Mean (Graduate Students)) was less than 0 in terms of *Sel-Time* and the other values for MD are greater than 0 for *NCF*, *FC*, *NST*, *PST*, *Eff_T* and *Eff_TC*. Notice that we do not report all the MD values in the paper to keep the paper more readable. In addition, there were no significant differences for the results between experts and graduate students since all the *p*-values were greater than 0.05 when using *Automated*. Notice that since there weren’t any significant differences we do not provide all the values for mean differences and *p*-value in this paper since they are similar for all the seven VCS products. Such results provide the evidence that *Automated* is easy to use and thus we can accept the null hypothesis $H_0$. 

7.2.6. Overall Discussion

In this section, we provide an overall discussion based on the results of RQs.

Based on the experiment results discussed above, we can conclude that for all the VCS products, our methodology (*Automated*) can significantly reduce the Cost test case selection time (*Sel-Time*), while achieves significantly higher effectiveness measured as four different variables, which are: the Number of Covered Features (*NCF*), Feature Coverage (*FC*), Number of Selected Test Cases (*NST*), and Percentage of Selected Test Cases (*PST*) as compared with the manual approach (*Manual*). In addition, *Automated* significantly improves the Efficiency of test case selection as compared to *Manual* based on two measures that study the effect of Cost and Effectiveness at the same time. We also combined the observations from all the products together and report the overall values for min, max, mean, median and standard deviation for the descriptive statistics as shown in Table 22. From the table, we can observe that *Automated* can outperform *Manual* in terms
of all the dependent variables, which is similar as what we observed for each product. For instance, Automated obtained on average 680 test cases (NST) achieving 0.95 for FC, whereas Manual obtained on average 184 that can only achieve 0.46 for FC.

<table>
<thead>
<tr>
<th>Table 22. Overall descriptive statistics for cost, effectiveness and efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Min</td>
</tr>
<tr>
<td>2.4</td>
</tr>
<tr>
<td>Max</td>
</tr>
<tr>
<td>Mean</td>
</tr>
<tr>
<td>Med.</td>
</tr>
<tr>
<td>Std.</td>
</tr>
</tbody>
</table>

* G1: group 1; G2: group 2; Med.: Median; Std.: standard deviation

The better performance of our proposed methodology (Automated) can be explained as follows: 1) our methodology relies on models and thus abstracts out the unnecessary details of test case implementation that are not required for test case selection; 2) We use standardized models (e.g., feature model) capturing the domain knowledge thus providing unified meaning to everyone and at the same times reduces the bias in the test case selection process. Notice that in our experiment we set the maximum time for each round to 8.5 hours due to practical reasons and in all the rounds participants working with the manual approach stopped at 8.5 hours. This can be explained from the fact that the manual approach requires going through the entire test case repository via the XML files, which is time consuming, error-prone, and tedious job.

When studying the relationship of increasing complexity of the products measured as the number of features on the Cost, Effectiveness, and Efficiency, we observed that as the complexity of a product increases, cost and effectiveness increases for our Automated methodology. The increase in Cost can be explained by the fact that with the increased complexity one needs to deal with more testing requirements leading. In case of Effectiveness, our automated methodology is systematic and automated based on the models and is not significantly affected by increased complexity. However, in terms of Efficiency measured as $Eff_F$ and $Eff_{TC}$, there was no significant impact with the increase in complexity suggesting that irrespective of complexity our automated methodology is efficient.
7.2.7. Concluding Remarks

Based on the above results and discussions, we summarize the answers for each research question as shown in Table 23.

<table>
<thead>
<tr>
<th>Research Question</th>
<th>Concluding Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>RQ1 (Cost)</td>
<td>Our automated methodology can significantly reduce the test case selection time (Sel-Time) as compared with the manual approach.</td>
</tr>
<tr>
<td>RQ2 (Effectiveness)</td>
<td>Our automated methodology can significantly improve the effectiveness in terms of number of covered features (NCF), feature coverage (FC), number of selected test cases (NST), and percentage of selected test cases (PST) when compared with the manual approach.</td>
</tr>
<tr>
<td>RQ3 (Efficiency)</td>
<td>Our automated methodology can significantly improve the efficiency of test case selection measured as efficiency for covering features (Eff_F) and efficiency for selecting test cases (Eff_TC) as compared with the manual approach.</td>
</tr>
<tr>
<td>RQ4 (Complexity)</td>
<td>Even with the increase in the complexity of products, the efficiency of our automated methodology is not significantly affected.</td>
</tr>
<tr>
<td>RQ 5 (Expert vs Graduate Students)</td>
<td>When using our automated methodology, there is no significant difference between the results obtained by experts and graduate students in terms of Sel-Time, NCF, FC, NST, PST, Eff_F and Eff_TC.</td>
</tr>
</tbody>
</table>

7.3. Threats to Validity

Below, we discuss the threats to validity of our experiments based on the principles discussed in [29].

7.3.1. Conclusion validity

*Conclusion validity* threats are concerned with factors that can influence the conclusion that can be drawn from the results of the experiments. As with most controlled experiments in software engineering, our main conclusion validity is due to the sample size on which we base our analysis. To tackle this issue, we used 10 participants per product per approach, which is a sufficient number to perform statistical analysis. Moreover, we used seven commercial products and thus obtained 70 data points per approach to further to maximize the number of observations within given time constraints. The other concern is that the cost, effectiveness, efficiency of selected test cases can be measured from various perspectives, which depends on different industrial contexts. To deal with this, we chose one cost measure, four effectiveness measures and two efficiency measures as seven dependent variables based on our experience of working with the industrial collaboration.
7.3.2. **Internal validity**

*Internal validity* threats exist when the outcome of results are influenced by internal factors and are not necessarily due to the application of the treatment being studied. Our controlled experiment applied between-subjects design based on [29] when comparing two groups using different approaches. More specifically, our experiment design consists of seven rounds and there were two separate groups denoted *Group 1* and *Group 2*. Given the number of the subjects, this led, respectively, to ten subjects in each group. In each round for each VCS product, *Group 1* was asked to perform test case selection using our automated methodology while *Group 2* was asked to select test cases manually. In addition, during the training sessions (Section 7.1.5), each subject was equally trained to understand the two approaches followed by given an assignment after the training sessions, for them to practice before the actual experiment tasks. Notice that the assignment was marked by the first author of this paper and grades were used to form blocks (i.e., groups of students of equivalent skills). The experiment groups were then formed through randomization and blocking to obtain two comparable groups with similar proportions of students from each skill block.

Through our experiment design, we have tried to minimize the chances of other factors being confounded with our primary independent variable: the treatment using our automated methodology as compared with the manual approach. In addition, we avoided any biased assignment of subjects to groups by using blocking based on assignments marks.

7.3.3. **Construct Validity**

*Construct Validity* threats are related to the degree to which the construct being studied (i.e., cost/effectiveness in our context) is affected by experiment settings, which includes one cost measure (*Sel-Time*), four effectiveness measures (*NC, FC, NST* and *PST*) and two efficiency measures (*Eff_F* and *Eff_TC*). The main threat for construct validity is that we cannot investigate all types of existing measures in terms of cost, effectiveness and efficiency, e.g., fault detection capability for the selected test cases [8]. However, as we previously discussed, the measures as dependent variables were defined together with test
engineers based on the industrial experience. More other measures can be evaluated using our methodology in the future.

7.3.4. External validity

External validity threats are typically the most common threats in controlled experiments. One potential threat is whether the material used for the experiment is representative of industrial practice. Due to time constraints, case studies and tasks used for the controlled experiment are usually small and this often tends to minimize the differences among treatments. To deal with this, we reused the seven commercial VCS products of varying complexity (ranging from 17 to 63 features) for the controlled experiment. Another external validity threat is due to the lack of representations of software professionals by the experiment subjects. Regarding to this threat, we compared the results obtained by experts (test engineers) and the involved subjects in our experiment (graduate students) when using our methodology. The results show that there is no significant difference for the results obtained by the experts and graduate students.

In addition, we need to mention that many practitioners have very little knowledge in terms of feature modeling, which anyway requires extensive training. Notice that we chose a group of experienced graduate students with an advanced educational background. In addition, some studies [31-33] have been reported on the performance, for various tasks, of trained software engineering graduate students when compared with professional developers. These differences were not statistically significant when compared to junior and intermediate developers. Such studies show that there is no evidence that graduate students with software engineering background and good training for the experiment cannot be used as subjects in place of software professionals.

8. Related Works

In this section, we discuss the related works in terms of product line testing, regression testing. Since our work falls into the category for product line testing and thus we first discuss related works for this in Section 8.1. In addition, we discuss the related works with respect to regression testing (Section 8.2) since product line testing and regression testing shares a lot of similarities, which can be reused as guidelines.
8.1. Product Line Testing

Product line testing is a relatively new, but intense field of research since product line engineering has shown significant benefits [5, 10-20]. McGregor [5] presented a set of activities, which can be used to address test individual assets (e.g., verification of consistency between requirements and specifications) and testing artefacts (e.g., test-case derivation and test suite design) that represent complete products in the context of product line. Muccini [10] discussed a set of challenges and opportunities for adapting the existing techniques (e.g., regression testing) to product line architecture testing, e.g., regression testing techniques can be extensively reused for product line testing but the main challenge is to capture the commonalities and variabilities for a product line systematically and identify the corresponding test cases, which can be reused. However, these works only provide guidelines and suggestions and do not provide any systematic and automated test case selection process [6].

In our previous work [40], we proposed a systematical automated test case selection methodology using Feature Model (FM) and Component Family Model (CFM). Moreover, in [40], we reported our research method used for the industrial collaboration and shared a set of lessons learnt related with adapting product line engineering (PLE) in practice. As compared with [40], the similarity for both works is that the same methodology for test case selection is applied to the industrial case study. However, there are two main differences between these two works including: 1) In this paper, besides the industrial case study, we conducted a controlled experiment including 20 graduate students for assessing the cost, effectiveness and efficiency when using our automated methodology; and 2) The results between experts and graduate students were further compared to provide additional evidence for the conclusion extracted from the controlled experiment.

Another our previous work proposed a product line modeling and configuration methodology by adapting FM and CFM to support model-based test case generation [35]. Similarly as this paper, a FM was defined to model a product line within various testing functionalities and a CFM was designed to model a repository developed for behavioral models (e.g., state machines, class diagrams). The output for this methodology is a set of configured behavioral models for testing a specific product by performing feature selection and in FM and attribute configuration in CFM. The configured behavioral models can be
further taken as input for test case generation. As compared with [35], we uses the same models (FM and CFM) in this paper but the objective is largely different, i.e., CFM designed in this paper focuses on modeling the test case repository for automated test case selection rather than the behavioral model repository for test case generation. Moreover, in this paper, we report a carefully-designed controlled experiment which involved 20 graduate students with aims at assessing the cost-effectiveness of our test case selection methodology as compared with the manual approach.

8.2. Regression Testing

Regression Testing is an attractive topic, which has been studied by a large number of researchers since the last decades [8, 34, 36, 37]. In the context of regression testing, there are three main research topics, which includes test selection, test minimization and test prioritization. More specifically, test case selection aims at identifying a set of relevant test cases when changes are made to existing software with aim to test a modified version of systems or programs while test minimization eliminates the redundant test cases from the existing test suite for the current systems or programs in order to reduce the cost of testing (e.g., time). Test prioritization orders the test cases in the existing test suite to test the current systems or programs with the objective of achieving the pre-defined criteria (e.g., maximum number of executing test cases) in a given time budget.

Our work in this paper is related to the topic of test selection and various types of such techniques are proposed in the literature, but mostly around the following two aspects: selection based on code changes [21-23] and selection based on specification changes [24, 25]. In addition, several thorough survey papers have been published in the literature [7, 8]. Although techniques for regression test selection have been evaluated in many previous works [26-28], there is no enough evidence to prove that these techniques still work well if being adapted in the context of product line. In our process, effort is spent on building reasonable models for product line and the structure of test cases, and making links between them. In contrast to regression testing, where focus is on testing changed functionality of an existing software system, our work is applicable when a new product is to be tested.
Our main objective is to perform test case selection automatically thereby reducing the selecting effort using FM and CFM in the context of product line. To the best of our knowledge, existing works have not covered such an objective: applying FM and CFM in product line for supporting automated selection of test cases in practice.

9. Conclusion

In this paper, we proposed a product line modeling methodology for automated test case selection with the aims of reducing selection effort at the same time covering all required test functionalities. The methodology consists of the following main parts: 1) defining a Feature Model for Testing (FM_T) to model a product line for testing; 2) defining a Component Family Model for Testing (CFM_T) to model the structure of test cases in the repository; and 3) linking CFM_T and FM_T via restrictions. With our methodology, test engineers only need to perform selection of features in FM_T and the related test cases can be chosen automatically from the repository.

We evaluated our methodology in three ways. First, we applied our methodology to the Saturn product line of Videoconferencing Systems developed by Cisco Systems, Inc, Norway and performed test case selection for its seven products. The results showed that the effort such as selection time can be reduced significantly at the same time all required testing functionalities can be covered for testing a product as compared with the current manual process at Cisco. Second, we conducted a questionnaire-based study to solicit the views of our automated methodology from test engineers at Cisco. The results showed that the test engineers are very positive about adapting our methodology in their current practice. Third, we report the results of a carefully designed controlled experiment with the aim of evaluating the performance of our automated methodology as compared with manual approach in terms of cost, effectiveness and efficiency. The results show that our automated methodology can significantly reduce the cost, while preserving high effectiveness as compared with the manual approach. In addition, as the increasing complexity of VCS products, the efficiency of our methodology is not significantly impacted, which indicates that our automated methodology can solve a wider range of problems with varying complexity.
References

9. www.cisco.com


Cost-Effective Test Suite Minimization in Product Lines Using Search Techniques

Shuai Wang¹,², Shaukat Ali¹ and Arnaud Gotlieb¹

¹Certus Software V&V Center, Simula Research Laboratory, Norway
²Department of Informatics, University of Oslo, Norway
{shuai, shaukat, arnaud}@simula.no

Abstract. Cost-effective testing of a product in a product line requires obtaining a set of relevant test cases from the entire test suite via test selection and minimization techniques. In this paper, we particularly focus on test minimization for product lines, which identifies and eliminates redundant test cases from test suites in order to reduce the total number of test cases to execute, thereby improving the efficiency of testing. However, such minimization may result in the minimized test suite with low test coverage, low fault revealing capability, low priority test cases, and require more time than the allowed testing budget (e.g., time) as compared to the original test suite. To deal with the above issues, we formulated the minimization problem as a search problem and defined a fitness function considering various optimization objectives based on the above issues. To assess the performance of our fitness function, we conducted an extensive empirical evaluation by investigating the fitness function with three weight-based Genetic Algorithms (GAs) and seven multi-objective search algorithms using an industrial case study and 500 artificial problems inspired from the industrial case study. The results show that Random-Weighted Genetic Algorithm (RWGA) significantly outperforms the other algorithms since RWGA can balance all the objectives together by dynamically updating weights during each generation. Based on the results of our empirical evaluation, we also implemented a tool called TEst Minimization using Search Algorithms (TEMSA) to support test minimization using various search algorithms in the context of product lines.

Keywords: Product Line; Search Algorithm; Test Suite Minimization; Feature Pairwise Coverage; Fault Detection Capability; Overall Execution Time
1. Introduction

Product Line Engineering (PLE) has proven to be a cost-effective way to develop similar products by exploiting and managing commonalities and variabilities among a large number of products [1, 2]. PLE has shown promising benefits in both academia and industry such as reducing development time and costs, speeding up product time-to-market and improving the quality of products of a product line family [1]. Cost-effective testing of a new product in a product line can be achieved by systematic application of model-based PLE approaches [2], for example, based on feature models [1]. Feature models have become the de-facto standard for capturing commonalities and variabilities in product lines and can be used for systematic selection of test cases for a new product in a product line.

Fig. 1. An overview of cost-effective testing of products in a product line

An overall view of our product line testing project is presented in Fig. 1. The project has two main activities: 1) Test selection for a product in a product line; 2) Test minimization for the selected test cases following the first activity. Test selection was focused in our previous works [3, 4], where we proposed a methodology for automated selection of test cases for a new product using Feature Model (FM) and Component Family Model (CFM) [5]. FM was used to capture commonalities and variabilities of features of a product line, where CFM was used to model the structure of test cases in a test case repository as shown in Fig. 1. Two types of traceability links were established: traceability links (traces) from CFM to actual test cases in the repository; 2) Links between FM and CFM using pre-defined restrictions. To test a new product (e.g., Product₂, as shown in Fig. 1), a test engineer only requires performing simple selection of relevant features in the
FM and the corresponding test cases can be automatically obtained from the test case repository.

Based on our experience of working with Cisco, we discovered that even after test selection, the number of test cases is still large (more than 2000 requiring 3-4 days to execute per product) and many of the test cases are redundant (e.g., several test cases cover the same functionality). On top of that, new test cases for product lines are continuously added further increasing the number of test cases. Executing all these test cases is practically impossible within the allocated testing time and thus warrants efficient and sophisticated test minimization techniques [6-10]. However, such minimization may reduce cost (e.g., execution time), but on the other hand may also reduce effectiveness (e.g., fault detection capability). This means an efficient technique must achieve a desirable balance among cost and effectiveness measures, which is a multi-objective optimization problem. A number of such objectives have been studied for multi-objective test optimization in regression testing in the existing literature such as execution time, test coverage and fault detection capability [7, 11].

Multi-objective search algorithms are well adapted for the type of problem we are solving in this paper [12, 13]. Moreover, Weight-based Genetic Algorithms (GAs) are also known to solve multi-objective optimization problems by assigning a set of weights to each objective (Konak et al., 2006). In our previous work (Wang et al., 2013c), we defined a fitness function based on the following cost/effectiveness measures based on our industrial collaboration and the existing literature [7, 8, 10]: 1) Cost: minimizing the number of test cases thereby indirectly reducing execution cost referred as Test Minimization Percentage (TMP); 2) Effectiveness: High Feature Pairwise Coverage (FPC) and Fault Detection Capability (FDC). In (Wang et al., 2013c), we compared three weight-based genetic algorithms (GAs) and showed that Random-Weighted GA (RWGA) had significantly better performance than the other algorithms.

Further investigation at Cisco revealed the following additional cost and effectiveness measures: 1) Cost: overall execution cost of the minimized test cases; 2) Effectiveness: Priority of test cases. Notice these two additional measures have also been investigated in the existing literatures [7, 8, 10]. Moreover, we also decided to compare weight-based algorithms with other types of algorithms such as Fast Non-dominated Sorting Genetic Algorithm (NSGA-II) [13, 14]. Based on these observations, this paper extends our
existing work [6] by including: 1) Four effectiveness measures, i.e., Test Minimization Percentage (TMP), Pairwise Coverage (FPC), Fault Detection Capability (FDC) and Average Execution Frequency (AEF) and one cost measure, i.e., Overall Execution Time (OET); 2) Multi-objective fitness function considering all the proposed cost/effectiveness measures; and 3) Empirical evaluation for the proposed fitness function in conjunction with three weight-based multi-objective search algorithms and seven other multi-objective search algorithms on an industrial case study and assessing the scalability using 500 artificial problems of varying complexity; 4) A tool called TEst Minimization with Search Algorithms (TEMSA) based on [15].

The results of our empirical evaluation showed that: 1) Based on the industrial case study, all the selected search algorithms except Improved Strength Pareto Evolutionary Algorithm (SPEA2) are cost-effective for solving the test minimization problem and RWGA achieves the best performance; 2) For the 500 artificial problems, the results are consistent with the industrial case study, i.e., RWGA outperforms all the other search algorithms. Furthermore, RWGA managed to solve the problems of varying complexity.

The rest of the paper is organized as follows. Section 2 provides a brief introduction of the selected search algorithms. Section 3 presents a formal representation of our problem, definitions and functions for the cost/effectiveness measures and fitness function used by all the algorithms. Section 4 describes our industrial case study and designed artificial problems followed by empirical evaluation of the applied algorithms in Section 5. Section 6 discusses the results and analysis for the experiments and Section 7 presents an overall discussion based on the obtained results. In Section 8, the tool support is discussed. Section 9 addresses the threats to validity of our empirical study and related work is discussed in Section 10. Section 11 concludes the paper and presents future work.

2. Description of the Selected Algorithms

In this section, we provide a brief description of the ten algorithms that were selected for our experiments. These algorithms are classified into four working mechanisms to guide the search [16], which are: Evolutionary Algorithms (EAs), Swarm Algorithms, Hybrid Algorithms and Stochastic Algorithms (Table 1). The column Minimization Before shows whether the listed algorithms have been adapted for solving the test minimization problem in the existing literature. Notice that one of our aims is to evaluate a representative
set of the existing search algorithms rather than limiting us to the commonly used algorithms (e.g., NSGA-II and SPEA2). To achieve this goal, we chose ten search algorithms systematically based on the categories classified in [16]. Before getting into more details for each algorithm (Section 2.2 to Section 2.5), we first provide a brief overview for search algorithms (Section 2.1).

<table>
<thead>
<tr>
<th>Different Mechanisms</th>
<th>Algorithms</th>
<th>Minimization Before?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Evolutionary Algorithms (EAs)</td>
<td>GAs</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Weight-Based GA</td>
<td>WBGA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WBGA-MO</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RWGA</td>
</tr>
<tr>
<td></td>
<td>Sorting-Based GA</td>
<td>NSGA-II</td>
</tr>
<tr>
<td></td>
<td>Cellular-Based GA</td>
<td>MOCell</td>
</tr>
<tr>
<td></td>
<td>Strength Pareto EA</td>
<td>SPEA2</td>
</tr>
<tr>
<td>Swarm Algorithm</td>
<td>Particle Swarm Optimization</td>
<td>SMPSO</td>
</tr>
<tr>
<td>Hybrid Algorithm</td>
<td>Cellular genetic algorithm + differential evolution</td>
<td>CellDE</td>
</tr>
<tr>
<td>Stochastic Algorithm</td>
<td>Random Search</td>
<td>RS</td>
</tr>
</tbody>
</table>

2.1. Overview of Search Algorithms

Generally speaking, search algorithms aim at mimicking natural phenomenon, for example, natural evolution process or behaviors of animals (e.g., bird flocking) to search optimal solutions for various optimization problems [16]. To apply search algorithms, fitness function (objective function) should be defined mathematically to guide the search with aims at assessing the capability of the solutions.

Moreover, for genetic algorithms (GAs), genes, individuals (also called chromosomes) should be carefully defined beforehand. More specifically, individuals refer to potential solutions with respect to the optimization problem and each individual (solution) is composed of a set of genes, which present units for the solution, e.g., test cases for the test suite in our case. The set of individuals handled by GAs is represented as population. Similarly, for swarm algorithm (Section 2.3), the potential solutions are named as particles and the set of the solutions handled by algorithms refers to swarm.

After properly defining fitness function (objective function), the search algorithms will be repeatedly run by applying various operators (e.g., selection) to produce the best individual (solution) until the terminating condition is satisfied (e.g., number of fitness
Three operators are usually adapted by various search algorithms, namely selection, crossover (also called recombination) and mutation [16]. More specifically, 1) Selection operator corresponds to the operation of selecting the solutions (e.g., individuals for evolutionary algorithms) with the best values based on the defined fitness function. Notice that the Selection operator can be applied to all the search algorithms; 2) Crossover operator is mainly applied in various GAs and refers to the operation of selecting two individuals (usually called parent individuals) and exchanging their genes partially. The genes that will be switched between the individuals are usually determined at random. New individuals obtained by crossover operator are usually called offspring; and 3) Mutation operator is also mostly adapted in GAs and refers to updating the individuals by changing the properties of genes randomly based on the predefined mutation rate (i.e., probability of changing the properties of genes).

2.2. Evolutionary Algorithms (EAs)

EAs are inspired by biological evolution, which was first proposed by Darwin to describe how natural selection and mutation occurs on various genes of species [16]. Since EAs are well known in search-based software engineering [12], we choose seven of them, which are further classified into GAs, Strength Pareto EA and Evolution Strategies. We describe each of them in detail as below.

The working theory of these three weight-based GAs is to assign a particular weight to each objective function for converting the multi-objective problem to a single objective problem with a scalar objective function [13]. The difference of these algorithms can be summarized as the different mechanisms of assigning weights for each objective. Weight-Based GA (WBGA) assigns fixed weights for each objective defined for each objective beforehand, which can be provided by users based on the domain knowledge and expertise. The pre-determined weight for each objective is used during all generations of the algorithm. WBGA for Multi-objective Optimization (WBGA-MO) uses a weight pool including a set of predefined weights and each solution can choose weights for each objective randomly at each generation. The assigned weight for each objective changes during each generation, but is still limited to the ones in the weight pool. Notice that such weight pool can also be pre-provided by users. RWGA randomly assigns normalized weights to multiple objective functions for each solution when selecting fittest individuals.
at each generation, i.e., assigning weights to objective functions is dynamic and there is no need to set fixed weights for different objectives beforehand.

NSGA-II is based on the Pareto dominance theory, which outputs a set of non-dominated solutions for multiple objectives [17]. The population is sorted into several non-dominated fronts using a ranking algorithm first. Then, individual solutions are selected from these non-dominated fronts by calculating the crowd distance, which is used to measure the distance between the individual solutions and the rest of the solutions in the population [13]. If two individual solutions are in the same non-dominated front, the solution with a higher value of crowd distance will be selected.

Multi-objective Cellular (MOCell) is based on the cellular model of GAs (cGAs) with an assumption that an individual only interact with its neighbors during the search process (neighborhood) [18]. Moreover, MOCell stores a set of obtained non-dominated individual solutions in an external archive. After each generation, MOCell replaces a fixed number of solutions chosen randomly from the population by selecting the same number of solutions from the archive until the termination conditions are met. Such replacement only takes place when the newly generated solutions from the population are worse than the solutions in the archive.

Improved Strength Pareto Evolutionary Algorithm (SPEA2) is also based on Pareto dominance theory [19]. The fitness value for each solution is calculated by summing up a strength raw fitness based on the defined objective functions and density estimation, where density estimation measures the distance between a solution and its nearest neighbors for maximizing the diversity of the obtained solutions. SPEA2 first stores a fixed number of best solutions into an archive by applying selection, crossover and mutation operators. Then, a new population is created by combining the solutions from the archive and the non-dominated solutions of the original solution. If the combined non-dominated solutions are more than the maximum size of population, the solution with minimum distance to any other solutions is selected by applying a truncation operator to calculate the distances to the its neighborhood.

Pareto Archived Evolution Strategy (PAES) is inspired by evolution theory [20]. By applying dynamic crossover, mutation operators, PAES aims at maximizing the suitability for the selected solutions. Moreover, PAES stores the obtained non-dominated solutions into an archive and newly generated solutions can be added into such archive if they are
better than the existing solutions by calculating objective functions.

2.3. **Swarm Algorithm**

Particle Swarm Optimization (PSO) is a biological metaheuristic inspired by the social foraging behavior of animals such as bird flocking [16].

Speed-constrained Multi-objective Particle Swarm Optimization (SMPSO) algorithm is based on PSO theory [20, 21]. Similar as NSGA-II, SMPSO selects best solutions by calculating crowding distance and also stores the selected individual solutions in an archive. SMPSO takes advantage of mutation operators to accelerate the speed of convergence and adapts velocity constriction mechanism to avoid the explosion of swarms. According to [20, 21], the performance of SMPSO is considered as the best one among the existing PSO algorithms.

2.4. **Hybrid Algorithm**

Differential Evolution (DE) algorithm is considered as another kind of EA, which generates solutions by applying recombination, mutation, and selection operators [16]. More specifically, DE calculates the weighted difference between two randomly selected solutions and integrate the obtained parts into a third solution for generating a new solution.

CellDE is a hybrid metaheuristic algorithm using MOCell as a search engine and replacing the typical selection, crossover and mutation operators for GAs with the recombination mechanism of DE [22]. It is considered that CellDE takes the advantage of cellular GA and DE with good diversity and convergence and the performance of CellDE improves significantly as compared with separate ones [22].

2.5. **Stochastic Algorithm**

Random Search (RS) is a stochastic algorithm and is commonly used as a baseline for evaluation [16]. RS is adapted to randomly generate solutions from the entire search space during each generation. Notice each newly generated solution is independent of the obtained solutions.

3. **Problem Representation and Fitness Function**

To guide the search towards an optimal solution using multi-objective search algorithms, it is essential to define a fitness function, as is the case for any search algorithm. In Section
3.1, we present a formal representation of the search problem, followed by definitions and functions for the cost/effectiveness measures in Section 3.3.2. In Section 3.3.3, the overall fitness function is defined.

3.1. Problem Representation

To precisely define our test minimization problem, the following definitions are presented first.

3.1.1. Basic Concepts

Let $P = \{p_1, p_2, p_3 \ldots p_{np}\}$ be a product line with a set of products, where $np$ is the number of products in $P$.

$F = \{f_1, f_2, f_3 \ldots f_{nf}\}$ is a feature model with a set of features to represent $P$ [1], where $nf$ is the number of features, i.e., functionalities need to be tested for $P$.

$TS = \{t_1, t_2, t_3 \ldots t_{nt}\}$ is a test suite including a large number of test cases ($nt$) for testing $P$.

$F_{pi} = \{f_1, f_2, f_3 \ldots f_{nf_{pi}}\}$ is a subset of $F$ to represent a specific product $p_i$ which is to be tested [1], where $f_i$ can be any feature in $F (f_i \in F)$ and $nf_{pi}$ is the number of features of $F_{pi} (1 \leq nf_{pi} \leq nf)$.

$TS_{pi} = \{t_1, t_2, t_3 \ldots t_{nt_{pi}}\}$ is a subset of the test suite $TS$ for testing the $p_i$ product, where $t_i$ can be any test case in $TS (t_i \in TS)$ and $1 \leq nt_{pi} \leq nt$. Notice in our case, $TS_{pi}$ is obtained by our previous test case selection methodology proposed in [3, 4]. Notice that the proposed approach in this paper can be applied separately (without the test case selection methodology) for multi-objective test minimization problems in other contexts.

$S_{pi} = \{s_{p_1}, s_{p_2}, s_{p_3} \ldots s_{ns_{pi}}\}$ is a set of potential solutions based on $TS_{pi}$ for $p_i$, where $ns_{pi}$ is the total number of solutions for testing $p_i$, which can be measured as $2^{nt_{pi}} - 1$. Notice that each potential solution includes a set of test cases from $TS_{pi}$. As the number of test cases increases, the potential solutions increase exponentially. Suppose we have 1000 test cases to test the product line $P$ and using the methodology proposed by [3, 4], 100 test cases are obtained to test the product $p_i$. Then there will be $2^{100} - 1$ potential solutions for testing $p_i$, whose solution space is huge.

3.1.2. Cost/Effectiveness Measures

$Cost = \{cost_1, cost_2, \ldots, cost_{ncost}\}$ is a set of cost measures for minimization, e.g.,
execution time, cost of a test minimization algorithm.

Cost \((s_i, CostMeasure)\) is a function, which returns the cost of a solution \(s_i\) from \(S_{p_i}\) based on a cost measure \((CostMeasure)\) from Cost.

\[\text{Effect} = \{\text{effect}_1, \text{effect}_2, \ldots, \text{effect}_{n_{\text{effect}}}\}\]

is a set of effectiveness measures for minimization, e.g., test minimization percentage, feature pairwise coverage.

\[\text{Effect}(s_i, EffectivenessMeasure)\]

is a function, which returns the effectiveness of a solution \(s_i\) from \(S_{p_i}\) based on an effectiveness measure \((EffectivenessMeasure)\) from \(\text{Effect}\).

Each solution \(s_i\) in \(S_{p_i}\) is comprised of \(n_{ts_i}\) test cases from \(TS_{p_i}\): \(\{t_{s_{i1}}, t_{s_{i2}}, t_{s_{i3}} \ldots t_{n_{ts_i}}\}\), where \(1 \leq n_{ts_i} \leq n_{tp_i}\) and has a set of values for cost measures \(Cost(s_i, CostMeasure)\) and effectiveness measures \(Effect(s_i, EffectivenessMeasure)\), which are defined in the next section.

### 3.1.3. Problem Representation

Our optimization problem can be represented as:

**Problem:** Search for a solution \(s_k\) \((s_k\) is comprised of \(\{t_{s_k1}, t_{s_k2}, t_{s_k3} \ldots t_{n_{ts_k}}\}\) from \(TS_{p_i}\), where \(1 \leq n_{ts_k} \leq n_{tp_i}\) from \(S_{p_i}\) for testing the product \(p_i\) to achieve the following two objectives (i.e., maximum effectiveness and minimum cost):

\[
\forall s_l \in S_{p_i} \cap s_l \neq s_k:
\sum^{n_{\text{effect}}} \text{Effect}(s_k, \text{effect}_j) \geq \sum^{n_{\text{effect}}} \text{Effect}(s_l, \text{effect}_j)
\]

\[
\sum^{n_{\text{cost}}} \text{Cost}(s_k, \text{cost}_j) \leq \sum^{n_{\text{cost}}} \text{Cost}(s_l, \text{cost}_j)
\]

### 3.2. Definitions and Functions for Effectiveness/Cost Measures

In this section, based on the above-mentioned problem, we provide mathematical definitions and functions of relevant effectiveness and cost measures in our context.

#### 3.2.1. Effectiveness Measures

In our context, effectiveness measures \(\text{Effect}\) includes four elements, i.e., test minimization percentage (\(\text{TMP}\)), feature pairwise coverage (\(\text{FPC}\)), fault detection capability (\(\text{FDC}\)) and average execution frequency (\(\text{AEF}\)). We define them in detail below:

##### 3.2.1.1. Test Minimization Percentage (\(\text{TMP}\))

**Definition 1.** \(\text{TMP}\) is to measure the amount of reduction for the number of test cases.  

**Objective Function 1.** \(\text{TMP}\) is calculated as follows.
\[ TMP_{s_k} = \left(1 - \frac{nt_{s_k}}{nt_{p_i}}\right) \times 100\% \]

As discussed in Section 3.1, \( nt_{s_k} \) is the number of test cases for the specific solution \( s_k \) obtained by search algorithms for testing the product \( p_i \), where \( 1 \leq nt_{s_k} \leq nt_{p_i} \). \( nt_{p_i} \) is the number of test cases in the test suite for testing product \( p_i \). \( TMP \) value ranges from 0 to 1 and a higher value of \( TMP \) shows more test cases are reduced in the minimized test suite.

### 3.2.1.2. Feature Pairwise Coverage (FPC)

**Definition 2.** \( FPC \) is to measure how much pairwise coverage can be achieved by a chosen solution [23].

We chose this type of coverage based on our domain knowledge, discussion with test engineers, and history data about faults since a higher percentage of detected faults are mainly due to the interactions between features. Notice that other types of coverage (e.g., code coverage) can also be introduced into the fitness function based on other cases. \( FPC \) in our case is designed to compute the capability of covering feature pairs by a chosen solution.

**Objective Function 2.** The function for measuring \( FPC \) is shown as below.

\[ FPC_{s_k} = \frac{Num_{FP_{s_k}}}{Num_{FP_{p_i}}} \]

\( Num_{FP_{s_k}} \) is the number of feature pairs covered in the test cases for the solution \( s_k \) measured as follows.

\[ Num_{FP_{s_k}} = \sum_{i=1}^{nt_{s_k}} Num_{FP_{tc_i}} \]

\( nt_{s_k} \) is the number of test cases for the solution \( s_k \), where \( 1 \leq nt_{s_k} \leq nt_{p_i} \). \( Num_{FP_{tc_i}} \) is the number of unduplicated feature pairs covered by the test case \( i (tc_i) \). The feature pairs covered by \( tc_i \) can be computed as: \( Num_{FP_{tc_i}} = C_2^{\text{size}(F_{tc_i})} \). \( \text{size} (F_{tc_i}) \) is the number of features tested by test case \( tc_i \). For instance, test case \( tc_i \) is used to test three features. Then, the feature pairs covered by test case \( tc_i \) are \( C_2^3 = 3 \times 2 / 2 = 3 \). Notice that if some feature pairs are repeated ones compared with the feature pairs covered by the previous test cases, repeated pairs will be removed when computing \( Num_{FP_{s_k}} \). Meanwhile, feature pairs only refer to the valid feature pairs that are not violated against
the constraints defined among various features. In our case, there are no specific constraints that are defined on the features. But if such constraints exist in our contexts, the invalid feature pairs will be eliminated when calculating $Num_{FP_{sk}}$ and $Num_{FP_{pt}}$.

$Num_{FP_{pi}}$ is all number of feature pairs for testing the product $p_i$ measured as:

$$Num_{FP_{pi}} = C_2^{\text{size}(F_{pi})} = nt_{p_i} * (nt_{p_i} - 1)/2.$$  $F_{pi}$ is the set of features representing the product $p_i$ including $nt_{p_i}$ features. For instance, if $p_i$ is represented by ten features, all feature pairs covered by $p_i$ are $C_2^{10} = 45$. Notice $FPC$ is calculated for a chosen test solution ranging from 0 to 1 and a higher value shows better feature pairwise coverage.

**3.2.1.3. Fault Detection Capability (FDC)**

*Definition 3.* $FDC$ is to measure the fault detection capability of a selected test solution for a product.

In our context, fault detection refers to the rate of successful execution for a test case in a given time, e.g., a week in our case. More specifically, the execution of a test case can be defined as a *success* if it can detect faults in a given time (a week in our case) and as a *fail* if it does not detect any fault.

*Objective Function 3.* $FDC$ is measured using the following function.

$$FDC_{sk} = \frac{\sum_{i=1}^{nt_{sk}} SucR_{tc_i}}{nt_{sk}}$$

$s_k$ refers to a solution for a product including a set of test cases and $nt_{sk}$ is the number of test cases for the solution $s_k$, where $1 \leq nt_{sk} \leq nt_{p_i}$. $SucR_{tc_i}$ is the successful rate of execution for test case $tc_i$, which can be measured as below.

$$SucR_{tc_i} = \frac{NumSuc_{tc_i}}{NumSuc_{tc_i} + NumFail_{tc_i}}$$

$NumSuc_{tc_i}$ is the number of *success* for test case $tc_i$ and $NumFail_{tc_i}$ is the number of *fail* for test case $tc_i$. For instance, a test case is usually executed 1000 times per week in Cisco. So if it executes successfully for 800 times, the $SucR$ is $800/1000 = 0.8$.

Note that $FDC$ value also ranges from 0 to 1 and a higher value represents better fault detection capability.

**3.2.1.4. Average Execution Frequency (AEF)**

*Definition 4.* $AEF$ is to measure the average execution frequency of the solution $s_k$ based
on the execution frequency of each included test cases during a given time, e.g., a week.

In our context, we found that a test suite may have high priority if it is executed very frequently during the given time based on the domain knowledge. Therefore, the minimized test suite should preferably consist of test cases with high priority, which can be determined by the average execution frequency. This is also our main objective to seek a solution with higher average execution frequency from the solution space.

**Objective Function 4.** The function to measure $AEF$ is presented as below.

$$AEF_{s_k} = \frac{\sum_{i=1}^{nt_{s_k}} EF_{tc_i}}{nt_{s_k}}$$

$nt_{s_k}$ is the number of test cases for the solution $s_k$, where $1 \leq nt_{s_k} \leq nt_{p_i}$. $EF_{tc_i}$ is the execution frequency for the test case $tc_i$ in a given week. For instance, suppose the solution $s_k$ consists of three test cases: $tc_1$, $tc_2$ and $tc_3$. These three test cases are executed 5 times, 6 times and 7 times, respectively. Then $AEF$ for the solution $s_k$ will be 6 (i.e., $(5+6+7)/3 = 6$). Notice that a higher value of $AEF$ represents higher average execution frequency thereby higher priority.

### 3.2.2. Cost Measures

Cost measures $Cost$ includes one element in our context, i.e., overall execution time (OET), which is defined as below.

**Definition 5.** OET is to measure the overall execution time for the solution $s_k$ consisting of a set of test cases.

**Objective Function 5.** OET can be measured using the function as below.

$$OET_{s_k} = \sum_{i=1}^{nt_{s_k}} AET_{tc_i}$$

$nt_{s_k}$ is the number of test cases for the solution $s_k$, where $1 \leq nt_{s_k} \leq nt_{p_i}$. $AET_{tc_i}$ is the average execution time for test case $tc_i$ in the solution $s_k$, which can be calculated as below.

$$AET_{tc_i} = \frac{\sum_{i=1}^{EF_{tc_i}} ET_{tc_i}}{EF_{tc_i}}$$

$EF_{tc_i}$ is the execution frequency of the test case $tc_i$ in a given week and $ET_{tc_i}$ is the overall time for each execution. For instance, the frequency of the test case $tc_i$ is three in a
given week and the time for each execution is 15 minutes, 20 minutes and 25 minutes. Then the \( AET_{t_c} = \frac{15 + 20 + 25}{3} = 20 \) minutes. Notice that a higher value of \( OET_{s_k} \) shows more time is required for the solution \( s_k \) to execute and our goal is to keep such time within the allowed time budget.

In summary, our test minimization problem is formulated as a five-objective optimization problem, which can be solved using search algorithms.

### 3.3. Fitness Function

Based on the defined effectiveness measures and cost measures in the previous section, our fitness function is presented as below. Notice that a lower value of fitness function represents better performance for a solution.

\[
\text{Fitness} = f(TMP, FPC, FDC, OET, AEF)
\]

Notice that the obtained values by different objective functions do not stay at the same comparable level. For instance, \( FDC \) ranges from 0 to 1 and the value of \( OET \) can be 1.5 (hours). So we first normalize the obtained values of all the five functions using the following normalization function [24, 25].

\[
nor(x) = \frac{x}{x + 1}
\]

Where \( x \) can be any value obtained for each objective (i.e., \( TMP, FPC, FDC, OET \) and \( AEF \)), such as 0.56 for \( FDC \) and 0.5 hour for \( OET \). Therefore, the specific fitness function for the weight-based GAs (i.e., WBGA, WBGA-MO and RWGA) can be formulated as below, which ranges from 0 to 1, where lower the value is, better the fitness function is.

\[
\text{Fitness}_w = 1 - w_1 \ast nor(TMP) - w_2 \ast nor(FPC) - w_3 \ast nor(FDC) - w_4 \ast (1 - nor(OET)) - w_5 \ast nor(AEF)
\]

\( w_1, w_2, w_3, w_4 \) and \( w_5 \) are the weights assigned to \( TMP, FPC, FDC, OET \) and \( AEF \), respectively satisfying the following constraint: \( \sum_{i=1}^{5} w_i = 1 \). Using this way, multi-objective optimization problem is converted to a single objective problem with a scalar objective function, which is a classical approach and is efficient to be solved using GAs [13]. Notice that RWGA assigns weights to objectives dynamically during each generation and it is not required to set fixed weights to different objectives before.

The other search algorithms are based on different mechanisms and there is no need to assign particular weights to each objective function. All the cost/effectiveness measures
including five objectives with functions have been clearly defined in Section 3.2. Notice that using NSGA-II, MOCeCell, SPEA2, PAES, SMPSO, CellDE, a set of individual solutions can be obtained after a specific number of fitness generations.

4. Industrial Case Study and Artificial Problems

First, we used an industrial case study of Videoconferencing System (VCSs) product line called *Saturn* provided by Cisco Systems, Norway to evaluate our fitness function [26]. *Saturn* has several VCS products including C20 and C90 and has more than 2000 test cases. Notice that not all test cases are relevant for each product and new test cases are developed for *Saturn* every day. Based on our collaboration with Cisco, we observed that the current testing practice is performed in two ways: 1) Executing all the test cases for a product; 2) Randomly selecting a subset of test cases for a product. Therefore, random search could be considered as an equivalent case, though not a very accurate measure and require additional experiments involving real testers, which can be performed in the future. The following paper will evaluate the performance of the selected search algorithms as compared with RS.

**Table 2. Four Products in *Saturn***

<table>
<thead>
<tr>
<th>Product</th>
<th>#Features</th>
<th># Test Cases</th>
</tr>
</thead>
<tbody>
<tr>
<td>C20</td>
<td>17</td>
<td>138</td>
</tr>
<tr>
<td>C40</td>
<td>25</td>
<td>167</td>
</tr>
<tr>
<td>C60</td>
<td>32</td>
<td>192</td>
</tr>
<tr>
<td>C90</td>
<td>43</td>
<td>239</td>
</tr>
</tbody>
</table>

We chose four products C20, C40, C60 and C90 from *Saturn*. There are 169 features in *Saturn* and each product includes a subset of these features. Meanwhile, each feature can be tested by at least one test case (usually more than one). More details are shown in Table 2. For instance, C20 has 17 features and 138 test cases are used to test these features. Each test case $t_c_i$ has a successful rate for execution ($SucR_{t_c_i}$), an average execution time ($AET_{t_c_i}$) and an execution frequency ($EF_{t_c_i}$) (Section 3.2). In general, for *Saturn*, each feature is associated with 5-10 test cases and each test case $t_c_i$ is associated with 1-5 features with $SucR_{t_c_i}$ ranging from 50% to 95%, $AET_{t_c_i}$ ranging from 2 minutes to 60 minutes and $EF_{t_c_i}$ ranging from 1 time per week to 50 times per week.

Moreover, we defined 500 artificial problems to empirically evaluate whether the fitness function defined in Section 3 can address our test minimization problem even with a
large number of features and test cases. With this objective in mind, we first created a feature repository including 1200 features and a test case repository of 60,000 test cases. For each test case $tc_i$, three key attributes are assigned randomly (inspired by our industrial case study but with expansion for generality), i.e., $SucR_{tc_i}$ ranges from 0% to 100%, $AET_{tc_i}$ ranges from 1 minutes to 100 minutes and $EF_{tc_i}$ ranges from 1 time to 100 times per week. Moreover, we created artificial problems with the increasing number of features and test cases, i.e., we used a range of 10 to 1000 with an increment of 10 for features number and each feature can be associated with test cases ranging from 5 to 45 with an increment of 10. So that 100*5 = 500 artificial problems were obtained in this way. Based on that, each artificial problem consists of a set of features associated with a test suite including a set of test cases and the goal is to minimize the test suite using the selected multi-objective algorithms.

5. Empirical Evaluation

This section first presents experiment design including: 1) research questions required to be addressed, 2) selected criteria for the algorithms and parameter settings, and 3) evaluation mechanisms for all the search algorithms. Moreover, we also describe relevant statistical tests used for our experiments followed by experiment execution.

5.1. Experiments Design

The goal of our experiments is to evaluate the fitness function in conjunction with the ten search algorithms (i.e., WGBA, WBGA-MO, RWGA, NSGA-II, MOCell, SPEA2, PAES, SMPSO, CellDE and RS) in terms of addressing the optimization problem: minimizing the test suite for testing a product with high $TMP$, $FPC$, $FDC$ and $AEF$, and $OET$ within allocated time budget.

5.1.1. Research Questions

We will answer the following research questions with our experiments:

$RQ1$: Are the selected multi-objective search algorithms cost/effective to solve our test minimization problem?

$RQ2$: Are the selected multi-objective search algorithms cost/effective as compared to RS?

$RQ3$: Which one achieves the best performance among the selected multi-objective search algorithms?
**RQ4**: How does the increment of the number of features and test cases impact the performance of the selected multi-objective search algorithms?

### 5.1.2. Selected Criteria for the Algorithms and Parameter Settings

In our experiments, we first compared three weight-based multi-objective GAs and RS, i.e., WGBA with two set of fixed weights based on the domain knowledge and expertise (WBGA_W1 (W1 = (0.2, 0.2, 0.2, 0.2, 0.2)), WBGA_W2 (W2 = (0.1, 0.2, 0.2, 0.4, 0.1)), WBGA-MO and RWGA. For all of them, we used a standard one-point crossover with a rate of 0.9 and mutation of a variable is done with the standard probability 1/n, where n is the number of variables (defaulted standard parameter settings from jMetal [15]). Meanwhile, the size of population and maximum number of fitness evaluation are set as 100 and 25000, respectively (all these standard settings performed well in our previous work [6]). Finally, RS was used as the comparison baseline to assess the difficulty of the addressed minimization problems [27].

Moreover, we compared the best weight-based GA with the other six selected search algorithms, i.e., NSGA-II, MOCCell, SPEA2, PAES, SMPSO, CellDE (RS is still used as a baseline for assessing the difficulty of the problems [27]). For all the other algorithms, we also used the default suggested parameter settings from jMetal [15] as shown in Table 3. Notice that different settings may lead to different performance for genetic algorithms, but standard settings usually perform well [28].

<table>
<thead>
<tr>
<th>Table 3. Parameterization for the Selected Search Algorithms (n is the number of variables)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Parameterization used in NSGA-II and SPEA2</strong></td>
</tr>
<tr>
<td>Population Size</td>
</tr>
<tr>
<td>Selection of Parents</td>
</tr>
<tr>
<td>Recombination</td>
</tr>
<tr>
<td>Mutation</td>
</tr>
<tr>
<td><strong>Parameterization used in MOCCell</strong></td>
</tr>
<tr>
<td>Population Size</td>
</tr>
<tr>
<td>Neighborhood</td>
</tr>
<tr>
<td>Selection of Parents</td>
</tr>
<tr>
<td>Recombination</td>
</tr>
<tr>
<td>Mutation</td>
</tr>
<tr>
<td>Archive Size</td>
</tr>
<tr>
<td><strong>Parameterization used in PAES</strong></td>
</tr>
<tr>
<td>Mutation</td>
</tr>
<tr>
<td>Archive Size</td>
</tr>
<tr>
<td><strong>Parameterization used in SMPSO</strong></td>
</tr>
<tr>
<td>Swarm Size</td>
</tr>
<tr>
<td>Mutation</td>
</tr>
<tr>
<td>Archive Size</td>
</tr>
<tr>
<td><strong>Parameterization used in CellDE</strong></td>
</tr>
<tr>
<td>Population Size</td>
</tr>
<tr>
<td>Neighborhood</td>
</tr>
<tr>
<td>Selection of Parents</td>
</tr>
<tr>
<td>Recombination</td>
</tr>
<tr>
<td>Archive Size</td>
</tr>
</tbody>
</table>
5.1.3. Evaluation Mechanisms

To address RQ1, a set of thresholds for $TM, FPC, FDC$ and $OET$ are selected showing the minimum acceptable values for a particular context. Notice that these thresholds are set through the domain analysis, discussion with test engineers and a test manager at Cisco, and test execution history. Meanwhile, there is no specific threshold provided for $AEF$ since the test engineers think that it is not meaningful for them to propose such threshold. In our context, these thresholds values are: $TMP:=0.8; FPC:=0.8; FDC:=0.85$ and $OET:=2$ (hours). Notice the purpose of these pre-defined thresholds is to assess the performance of the selected search algorithms and the obtained solutions are independent of the availability of these threshold values. In other contexts, experts can provide such thresholds if required.

As discussed in Section 2, NSGA-II, MOCell, SPEA2, PAES, SMPSO, CellIDE may produce more than one solutions. Our evaluation mechanism is to choose the best solution with the highest value of objective functions for $TM, FPC, FDC$ and $OET$ and compare the values of objective functions with provided thresholds separately. For instance, for $TMP$ using NSGA-II, the highest value of objective function for the best solution is first chosen from the obtained ones. The value of the objective function is then compared with the threshold for $TMP$. Notice that for each objective, the chosen solution may be different so that it is essential to seek a mechanism to evaluate the overall performance of obtained solutions by the algorithms considering all the objectives together. Our mechanism is to calculate the average value to combine the five provided thresholds and the values of objective functions for the solutions obtained by all the search algorithms. The formula for combining is shown as below and the combined value is called as Overall Fitness Value ($OFV$).

$$OFV = \frac{(nor (TMP) + nor (FPC) + nor (FDC) + (1 - nor (OET)) + nor (AEF))}{5}$$

Afterwards, we can evaluate the overall performance for all the selected search algorithms using the values of $OFV$. Such mechanism makes sense since we define the same objective functions, and use the same formula to combine the values of objective functions for all the algorithms. Notice that higher the $OFV$ value is, better the performance of a specific algorithm is.

To answer RQ2, RQ3 and RQ4, the same mechanisms are adapted as above when
comparing all the algorithms. Notice that for the industrial case study, we first compare and analyze the results obtained by the selected algorithms for each objective separately and then evaluate the OFV considering the five objectives together. As for artificial problems, we only evaluate the values of OFV to assess the performance and scalability for the ten algorithms. Moreover, for artificial problems, Mean Fitness Value for each problem ($MFV_{i,j}$, $i$ refers to the number of features and $j$ means the number of associated test cases, i.e., $10 \leq i \leq 1000$ with an increment of 10 and $5 \leq j \leq 45$ with an increment of 10) is defined to measure the mean overall fitness value for a certain number of runs $nr$ as below (in our case, $nr = 100$). OFV$_{i,j,r}$ is the fitness value of the 25000$^{th}$ generation after the $r^{th}$ run with $i$ features and $j$ associated test cases.

$$MFV_{i,j} = \frac{\sum_{r=1}^{nr} OFV_{i,j,r}}{nr}$$

Moreover, Mean Fitness Value for Features ($MFV_F$) is defined for RQ4 to measure how the increment of the number of features impacts the performance of the selected search algorithms as below.

$$MFV_F = \frac{\sum_{j=5}^{45} \sum_{i=10}^{1000} MFV_{i,j}}{5}$$

Where $10 \leq i \leq 1000$ with an increment of 10, i.e., a set of hundred $MFV_F$ values can be obtained for various numbers of features.

Similarly, Mean Fitness Value for associated Test Cases ($MFV_TC$) is also defined for RQ4 to measure how the increment of the number of associated test cases impacts the performance of the selected multi-objective search algorithms as follows.

$$MFV_TC = \frac{\sum_{i=10}^{1000} \sum_{j=5}^{j+10} MFV_{i,j}}{100}$$

Where $5 \leq i \leq 45$ with an increment of 10, i.e., a set of five $MFV_TC$ values can be obtained for different number of associated test cases.

5.2. Statistical Tests

To compare the obtained result and given thresholds, the Vargha and Delaney statistics, Kruskal–Wallis test and Mann-Whitney U test are used based on the guidelines for reporting statistical tests for randomized algorithms [28]. The Vargha and Delaney statistics is used to calculate $\hat{A}_{12}$, which is a non-parametric effect size measure. In our context, $\hat{A}_{12}$ is used to compare the probability of yielding higher values for each objective
function and overall fitness value (OFV) for two algorithms A and B. If $\hat{A}_{12}$ is 0.5, the two algorithms are equivalent. If $\hat{A}_{12}$ is greater than 0.5, A has higher chances to obtain better solution than B. To determine statistical significance of results, first the Kruskal–Wallis test together with the Bonferroni Correction is performed to determine if there are significant differences among all the algorithms [29]. Notice that the Bonferroni Correction is used for adjusting $p$-value obtained by the Kruskal–Wallis test since the Kruskal–Wallis test is used to compare multiple sets of results (samples) obtained by various search algorithms in our case. If significant differences are observed, we further perform pair-wise comparisons of the algorithms using the Mann-Whitney U test [29]. Therefore, when comparing the performance of the selected search algorithms (RQ2 and RQ3), we first perform the Kruskal–Wallis test to assess whether there exists significant differences for all the samples obtained by various algorithms. If significant differences are observed, the Mann-Whitney U test is further performed together with Vargha and Delaney statistics for pair-wise comparison of the algorithms. We choose the significance level of 0.05, i.e., there is a significant difference if $p$-value is less than 0.05. Based on the above description, we follow the following convention in the paper: An algorithm A has better performance than algorithm B, if the $\hat{A}_{12}$ value is greater than 0.5 and have significantly better performance than B if $p$-value is less than 0.05.

Moreover, to address RQ4, we choose the Spearman’s rank correlation coefficient ($\rho$) to measure the relations between the MFV of the algorithms and different number of features and test cases [30]. The value of $\rho$ ranges from -1 to 1, i.e., there is a positive correlation if $\rho$ is equal to 1 and a negative correlation when $\rho$ is -1. A $\rho$ close to 0 shows that there is no correlation between the two sets of data. Moreover, we also report significance of correlation using $\text{Prob}>|\rho|$, a value lower than 0.05 means the correlation is statistically significant.

5.3. Experiment Execution

According to the guidelines in [28], each algorithm must be run for at least 100 times to account for random variation inherited in search algorithms. All the selected algorithms are run up to 25000 generations each time and we collected the optimal solution including the final value of fitness function for the 25000th generation. We ran our experiments on two PCs with Intel Core i7 2.3GHz with 4 GB of RAM, running Microsoft Windows 7.
operating system. In summary, it took 204.5 hours (eight and half days) to run the algorithms for all the problems.

6. Results and Analysis

In this section, we present the results and analysis of the experiments for the individual research question including two parts: 1) comparing the performance of the three weight-based GAs since all of them are based on weight theory; and 2) comparing the performance of the best weight-based GA and the other selected search algorithms, which are classified into different mechanisms to assess which one can achieve the best performance for our minimization problem. Recall that RQ1-RQ3 address questions how the selected algorithms perform in terms of cost/effectiveness for our minimization problem (as compared with RS), while RQ4 tackles the question corresponded with the impact on the performance of these algorithms with the growth of features and associated test cases (Section 5.1.1).

6.1. Results and Analysis for the Selected Weight-Based GAs

In this section, we present the results and analysis for the selected weight-based GAs (i.e., WBGA_W1, WBGA_W2, WBGA-MO and RWGA) using our industrial case study and 500 artificial problems. Furthermore, we also report a timing analysis, where we compare time taken by all the weight-based GAs as compared with RS.

6.1.1. Results and Analysis for Industrial Case Study

We first discuss the results and analysis for the industrial case study for addressing the proposed research questions.

6.1.1.1. Results and Analysis for Research Question 1

Table 4 shows the results, when the performance of the selected weight-based algorithms (WBGA_W1, WBGA_W2, WBGA-MO and RWGA) are compared with the threshold values for each objective (i.e., TMP, FPC, FDC, OET) for our industrial case study, while Table 5 describes the results for the OFV (i.e., overall fitness value) obtained by the selected algorithms. In these two tables, we did one sample Mann Whitney test since we compared the results by different algorithms with one set of fixed values for TMP, FPC, FDC, OET and OFV. Notice that RS is a baseline for assessing the difficulty of the addressed problems.
Based on the obtained results, we can answer RQ1 as follows:

For all the objectives, WBGA-MO and RWGA have higher probability to obtain better results when compared with the given thresholds, i.e., they have higher probability to be used when required since all of the $\hat{A}_{12}$ values are greater than 0.5 for TMP, FPC and FDC and all of the $\hat{A}_{12}$ values are less than 0.5 for OET. Moreover, there are no significant differences (all $p$-values are greater than 0.05) when comparing with the given thresholds as shown in Table 4.

For WBGA_W1 and WBGA_W2, the results do not stay stable. For some products and objectives (e.g., for TMP in C40 product), WBGA_W1 and WBGA_W2 have more chance to achieve better results than the given thresholds since the $\hat{A}_{12}$ values are greater than 0.5 (Table 4). But for the other products and objectives (e.g., for FDC in C90 product), WBGA_W1 and WBGA_W2 have less chance to obtain the better results as compared with the given thresholds since the $\hat{A}_{12}$ values are much less than 0.5 (Table 4). Since the only difference between WBGA_W1 and WBGA_W2 is the sets of weights, different weights can result in total different results when using WBGA. So providing an appropriate set of weights before using WBGA is essential to obtain the expected results.

<table>
<thead>
<tr>
<th>Products</th>
<th>Algorithms</th>
<th>Objectives with the Given Thresholds</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>TMP: 0.8</td>
</tr>
<tr>
<td>C20</td>
<td>WBGA_W1</td>
<td>0.45</td>
</tr>
<tr>
<td></td>
<td>WBGA_W2</td>
<td>0.55</td>
</tr>
<tr>
<td></td>
<td>WBGA-MO</td>
<td>0.61</td>
</tr>
<tr>
<td></td>
<td>RWGA</td>
<td>0.69</td>
</tr>
<tr>
<td></td>
<td>RS</td>
<td>0.22</td>
</tr>
<tr>
<td>C40</td>
<td>WBGA_W1</td>
<td>0.55</td>
</tr>
<tr>
<td></td>
<td>WBGA_W2</td>
<td>0.68</td>
</tr>
<tr>
<td></td>
<td>WBGA-MO</td>
<td>0.57</td>
</tr>
<tr>
<td></td>
<td>RWGA</td>
<td>0.68</td>
</tr>
<tr>
<td></td>
<td>RS</td>
<td>0.35</td>
</tr>
<tr>
<td></td>
<td>WBGA_W1</td>
<td>0.61</td>
</tr>
<tr>
<td></td>
<td>WBGA_W2</td>
<td>0.59</td>
</tr>
<tr>
<td></td>
<td>WBGA-MO</td>
<td>0.56</td>
</tr>
<tr>
<td></td>
<td>RWGA</td>
<td>0.67</td>
</tr>
<tr>
<td></td>
<td>RS</td>
<td>0.39</td>
</tr>
<tr>
<td>C60</td>
<td>WBGA_W1</td>
<td>0.64</td>
</tr>
<tr>
<td></td>
<td>WBGA_W2</td>
<td>0.66</td>
</tr>
<tr>
<td></td>
<td>WBGA-MO</td>
<td>0.67</td>
</tr>
<tr>
<td></td>
<td>RWGA</td>
<td>0.70</td>
</tr>
<tr>
<td></td>
<td>RS</td>
<td>0.22</td>
</tr>
</tbody>
</table>

* $\hat{A}_{12}$, p: $p$-value. All $p$-values less than 0.05 are identified as bold.

Table 5. Results for Comparing the Weight-based Algorithms with the Given Thresholds for OFV*

<table>
<thead>
<tr>
<th>Pair of Algorithms</th>
<th>C20 A</th>
<th>C20 p</th>
<th>C40 A</th>
<th>C40 p</th>
<th>C60 A</th>
<th>C60 p</th>
<th>C90 A</th>
<th>C90 p</th>
</tr>
</thead>
<tbody>
<tr>
<td>WBGA_W1 vs. Thresholds</td>
<td>0.51</td>
<td>0.23</td>
<td>0.52</td>
<td>0.04</td>
<td>0.35</td>
<td>0.04</td>
<td>0.40</td>
<td>0.72</td>
</tr>
</tbody>
</table>
Based on the results, we can see that RS always has less probability to be used in practice when compared with the given thresholds since all the $\hat{A}_{12}$ values are less than 0.5 for TMP, FPC and FDC and all of the $\hat{A}_{12}$ values are greater than 0.5 for OET. Moreover, there are significant differences when comparing with the given thresholds since most of the $p$-values are less than 0.05 as shown in Table 4.

When comparing the results for OFV considering all the objectives together, WGBA-MO and RWGA have higher probability to achieve better OFV than the given thresholds since all the $\hat{A}_{12}$ values are greater than 0.5 and there are almost no significant differences (most of the $p$-values are greater than 0.05) as shown in Table 5. As for WBGA (WBGA_\text{W1} and WBGA_\text{W2}), the results do not stay stable, i.e., some of them have higher probability to obtain better OFV as compared with the thresholds while some others have less probability to achieve better OFV than the thresholds (Table 5). However, RS always has less probability to obtain better OFV than the thresholds since all the $\hat{A}_{12}$ values are greater than 0.5 and all the $p$-values are less than 0.05 (Table 5).

In summary, weight-based GAs can assist to achieve the given thresholds. However, for WBGA (i.e., WGBA_\text{W1} and WGBA_\text{W2}), determining appropriate weights is challenging as it mostly dependent on domain-expertise and expert-guessing of what should be the most appropriate values, which is the main reason why WBGA with various weights can achieve different weights. But WGBA-MO and RWGA are not dependent on the pre-defined weights thus the obtained results are more stable.

### 6.1.1.2. Results and Analysis for Research Question 2 and 3

For the Saturn products, the Kruskal–Wallis test (together with the Bonferroni Correction) was first performed for all the samples obtained by weight-based GAs and RS with respect to each objective and OFV (considering all the objective together, Section 5.1.3) to determine whether there are significant differences between the performances of the algorithms. We obtained the following p-values: 0.017 for TMP, 0.008 for FPC, 0.020 for FDC, 0.012 for OET and 0.010 for OFV, which shows that there exist significant differences between the performance of the weight-based GAs and RS since all the $p$-values are less than 0.05. Therefore, the Vargha and Delaney statistics and the Mann-
Whitney U test were further performed to compare each pair of weight-based GAs as shown in Table 6 and Table 7. In total, we have $C_2^5 = 10$ pairs to compare for the three selected search algorithms. The obtained results are first compared for each objective (Table 6) and then compared for OFV considering all the objectives (Table 7).

**Table 6. Results for Comparing the Weight-based Algorithms for Each Objective**

<table>
<thead>
<tr>
<th>Products</th>
<th>Pair of Algorithms</th>
<th>Objectives</th>
<th>C20</th>
<th>C40</th>
<th>C60</th>
</tr>
</thead>
<tbody>
<tr>
<td>WBGA $W_1$ vs. WBGA $W_2$</td>
<td>0.57</td>
<td>0.29</td>
<td>0.58</td>
<td>0.15</td>
<td>0.45</td>
</tr>
<tr>
<td>WBGA $W_1$ vs. WBGA-MO</td>
<td>0.35</td>
<td>0.05</td>
<td>0.41</td>
<td>0.38</td>
<td>0.40</td>
</tr>
<tr>
<td>WBGA $W_1$ vs. RWGA</td>
<td>0.22</td>
<td>0.02</td>
<td>0.40</td>
<td>0.25</td>
<td>0.27</td>
</tr>
<tr>
<td>WBGA $W_1$ vs. RS</td>
<td>0.66</td>
<td>0.04</td>
<td>0.58</td>
<td>0.17</td>
<td>0.69</td>
</tr>
<tr>
<td>WBGA $W_1$ vs. WBGA-MO</td>
<td>0.54</td>
<td>0.12</td>
<td>0.47</td>
<td>0.53</td>
<td>0.40</td>
</tr>
<tr>
<td>WBGA $W_1$ vs. RWGA</td>
<td>0.28</td>
<td>0.02</td>
<td>0.32</td>
<td>0.02</td>
<td>0.41</td>
</tr>
<tr>
<td>WBGA $W_2$ vs. RS</td>
<td>0.65</td>
<td>0.02</td>
<td>0.66</td>
<td>0.02</td>
<td>0.57</td>
</tr>
<tr>
<td>WBGA $W_2$ vs. RWGA</td>
<td>0.30</td>
<td>0.01</td>
<td>0.32</td>
<td>0.02</td>
<td>0.34</td>
</tr>
<tr>
<td>WBGA-MO vs. RS</td>
<td>0.67</td>
<td>0.02</td>
<td>0.70</td>
<td>0.01</td>
<td>0.38</td>
</tr>
<tr>
<td>WBGA-MO vs. RWGA</td>
<td>0.61</td>
<td>0.06</td>
<td>0.75</td>
<td>0.01</td>
<td>0.59</td>
</tr>
<tr>
<td>RWGA vs. WBGA</td>
<td>0.52</td>
<td>0.44</td>
<td>0.47</td>
<td>0.16</td>
<td>0.51</td>
</tr>
<tr>
<td>RWGA vs. RS</td>
<td>0.53</td>
<td>0.43</td>
<td>0.58</td>
<td>0.15</td>
<td>0.46</td>
</tr>
<tr>
<td>RWGA vs. RWGA</td>
<td>0.52</td>
<td>0.46</td>
<td>0.36</td>
<td>0.19</td>
<td>0.29</td>
</tr>
<tr>
<td>RWGA vs. WBGA</td>
<td>0.35</td>
<td>0.04</td>
<td>0.17</td>
<td>0.01</td>
<td>0.25</td>
</tr>
<tr>
<td>RWGA vs. WBGA-MO</td>
<td>0.59</td>
<td>0.27</td>
<td>0.65</td>
<td>0.04</td>
<td>0.68</td>
</tr>
<tr>
<td>RWGA vs. WBGA-MO</td>
<td>0.57</td>
<td>0.09</td>
<td>0.45</td>
<td>0.22</td>
<td>0.47</td>
</tr>
<tr>
<td>RWGA vs. RS</td>
<td>0.30</td>
<td>0.03</td>
<td>0.23</td>
<td>0.01</td>
<td>0.29</td>
</tr>
<tr>
<td>RWGA vs. RWGA</td>
<td>0.60</td>
<td>0.04</td>
<td>0.59</td>
<td>0.04</td>
<td>0.61</td>
</tr>
<tr>
<td>RWGA vs. RS</td>
<td>0.39</td>
<td>0.14</td>
<td>0.29</td>
<td>0.02</td>
<td>0.30</td>
</tr>
<tr>
<td>RWGA vs. RWGA</td>
<td>0.60</td>
<td>0.03</td>
<td>0.68</td>
<td>0.02</td>
<td>0.35</td>
</tr>
<tr>
<td>RWGA vs. RS</td>
<td>0.59</td>
<td>0.03</td>
<td>0.62</td>
<td>0.02</td>
<td>0.74</td>
</tr>
<tr>
<td>RWGA vs. RWGA</td>
<td>0.48</td>
<td>0.39</td>
<td>0.57</td>
<td>0.16</td>
<td>0.55</td>
</tr>
<tr>
<td>RWGA vs. RS</td>
<td>0.40</td>
<td>0.15</td>
<td>0.31</td>
<td>0.07</td>
<td>0.41</td>
</tr>
<tr>
<td>RWGA vs. RWGA</td>
<td>0.26</td>
<td>0.03</td>
<td>0.36</td>
<td>0.12</td>
<td>0.38</td>
</tr>
<tr>
<td>RWGA vs. RS</td>
<td>0.57</td>
<td>0.10</td>
<td>0.67</td>
<td>0.03</td>
<td>0.59</td>
</tr>
<tr>
<td>RWGA vs. RWGA</td>
<td>0.44</td>
<td>0.20</td>
<td>0.46</td>
<td>0.32</td>
<td>0.40</td>
</tr>
<tr>
<td>RWGA vs. RS</td>
<td>0.37</td>
<td>0.05</td>
<td>0.40</td>
<td>0.13</td>
<td>0.31</td>
</tr>
<tr>
<td>RWGA vs. RWGA</td>
<td>0.62</td>
<td>0.03</td>
<td>0.65</td>
<td>0.02</td>
<td>0.68</td>
</tr>
<tr>
<td>RWGA-MO vs. RS</td>
<td>0.36</td>
<td>0.03</td>
<td>0.39</td>
<td>0.05</td>
<td>0.30</td>
</tr>
<tr>
<td>RWGA-MO vs. RWGA</td>
<td>0.64</td>
<td>0.03</td>
<td>0.67</td>
<td>0.02</td>
<td>0.70</td>
</tr>
<tr>
<td>RWGA-MO vs. RS</td>
<td>0.69</td>
<td>0.01</td>
<td>0.71</td>
<td>0.01</td>
<td>0.77</td>
</tr>
</tbody>
</table>

*All $p$-values less than 0.05 are identified as bold.

**Table 7. Results for Comparing the Weight-based Algorithms for OFV**

<table>
<thead>
<tr>
<th>Pair of Algorithms</th>
<th>C20</th>
<th>C40</th>
<th>C60</th>
<th>C90</th>
</tr>
</thead>
<tbody>
<tr>
<td>WBGA $W_1$ vs. WBGA $W_2$</td>
<td>0.41</td>
<td>0.23</td>
<td>0.42</td>
<td>0.38</td>
</tr>
<tr>
<td>WBGA $W_1$ vs. WBGA-MO</td>
<td>0.36</td>
<td>0.29</td>
<td>0.32</td>
<td>0.05</td>
</tr>
<tr>
<td>WBGA $W_1$ vs. RWGA</td>
<td>0.18</td>
<td>0.01</td>
<td>0.27</td>
<td>0.03</td>
</tr>
<tr>
<td>WBGA $W_1$ vs. RS</td>
<td>0.60</td>
<td>0.04</td>
<td>0.69</td>
<td>0.01</td>
</tr>
<tr>
<td>WBGA $W_1$ vs. WBGA-MO</td>
<td>0.39</td>
<td>0.04</td>
<td>0.40</td>
<td>0.12</td>
</tr>
<tr>
<td>WBGA $W_1$ vs. RWGA</td>
<td>0.33</td>
<td>0.02</td>
<td>0.36</td>
<td>0.02</td>
</tr>
<tr>
<td>WBGA $W_1$ vs. RS</td>
<td>0.60</td>
<td>0.04</td>
<td>0.59</td>
<td>0.04</td>
</tr>
<tr>
<td>WBGA-MO vs. RS</td>
<td>0.39</td>
<td>0.04</td>
<td>0.32</td>
<td>0.02</td>
</tr>
<tr>
<td>WBGA-MO vs. RS</td>
<td>0.60</td>
<td>0.03</td>
<td>0.69</td>
<td>0.02</td>
</tr>
</tbody>
</table>

147
Based on the obtained results, we can answer RQ2 and RQ 3 as follows:

**RQ2:** For all the objectives (i.e., TMP, FPC, FDC and OET), the results show that all the selected weight-based GAs have significantly better performance than RS since all the $\hat{A}_{12}$ values for TMP, FPC, and FDC are greater than 0.5, and all the $\hat{A}_{12}$ values for OET are less than 0.5, and most of the $p$-values are less than 0.05 (Table 6). When comparing the OFV results considering all objectives for the algorithms and RS, we can see that all of the selected weight-based GAs have significantly better performance than RS since all the $\hat{A}_{12}$ values are greater than 0.5 and all the $p$-values are less than 0.05 (Table 7).

**RQ3:** For all the objectives (i.e., TMP, FPC, FDC and OET), the results show that RWGA significantly outperforms WBGA$_{W1}$, WBGA$_{W2}$ and WBGA-MO since all the $\hat{A}_{12}$ values for TMP, FPC, and FDC are greater than 0.5, and all the $\hat{A}_{12}$ values for OET are less than 0.5, and most of the $p$-values are less than 0.05 (Table 6). Similarly, when comparing OFV, the performance of RWGA is also significantly better than the other weight-based GAs since all the $\hat{A}_{12}$ values are greater than 0.5 and all the $p$-values are less than 0.05 (Table 7).

In general, we can conclude that RWGA achieves the best performance among the selected weight-based GAs for all the objectives and overall fitness value (OFV) when considering all the objectives together.

### 6.1.2. Results and Analysis for Artificial Problems

To answer RQ2 and RQ3, we first conducted the Kruskal–Wallis test (together with the Bonferroni Correction) for all the obtained results and we obtained a $p$-value of 0.014 (less than 0.05) showing that there are significant differences between the results obtained by the selected weight-based GAs and RS. Moreover, we further compared the selected weight-based algorithms with RS and then compared each pair of them based on mean fitness value (MFV) obtained after 25000 generations for each algorithm and each of the problems. Recall that each problem was repeated for 100 times to account for random variation (i.e., MFV is a mean for 500 values of OFV after 100 runs). Notice that we answer RQ4 in terms of scalability for all the ten search algorithms in Section 6.2.2.2.

Table 8 summarizes the results for Vargha and Delaney statistics without/with Whitney U test for the 500 artificial. Two numbers are shown in each cell of the table split by a
slash. The first number in the column A>B shows the number of problems out of 500 for which an algorithm A has better performance than B (\( \hat{A}_{12} > 0.5 \)), A<B means vice versa (\( \hat{A}_{12} < 0.5 \)), and A=B means the number of problems for which A has equivalent performance as B (\( \hat{A}_{12} = 0.5 \)). The second number after “/” in the column A>B means the number of problems out of 500 for which an algorithm A has significantly better performance than B (\( \hat{A}_{12} > 0.5 \) \&\& \( p < 0.05 \)), A<B means vice versa (\( \hat{A}_{12} < 0.5 \) \&\& \( p < 0.05 \)), and A=B means the number of problems for which there are no significant differences in performance between A and B (\( p \geq 0.05 \)).

Table 8. Results for the Vargha and Delaney Statistics-artificial problems (without/with Whitney U test)

<table>
<thead>
<tr>
<th>RQ</th>
<th>Pair of Algorithms (A vs. B)</th>
<th>A&gt;B</th>
<th>A&lt;B</th>
<th>A=B</th>
</tr>
</thead>
<tbody>
<tr>
<td>RQ2</td>
<td>WBGA_W1 vs. RS</td>
<td>367/354</td>
<td>133/102</td>
<td>0/54</td>
</tr>
<tr>
<td></td>
<td>WBGA_W2 vs. RS</td>
<td>392/371</td>
<td>108/96</td>
<td>0/33</td>
</tr>
<tr>
<td></td>
<td>WBGA_MO vs. RS</td>
<td>445/426</td>
<td>55/45</td>
<td>0/29</td>
</tr>
<tr>
<td></td>
<td>RWGA vs. RS</td>
<td>48/480</td>
<td>11/0</td>
<td>0/20</td>
</tr>
<tr>
<td>RQ3</td>
<td>WBGA_W1 vs. WBGA_W2</td>
<td>226/202</td>
<td>274/264</td>
<td>0/34</td>
</tr>
<tr>
<td></td>
<td>WBGA_W1 vs. WBGA_MO</td>
<td>198/170</td>
<td>302/287</td>
<td>0/43</td>
</tr>
<tr>
<td></td>
<td>WBGA_W1 vs. RWGA</td>
<td>119/93</td>
<td>381/371</td>
<td>0/36</td>
</tr>
<tr>
<td></td>
<td>WBGA_W2 vs. WBGA_MO</td>
<td>211/198</td>
<td>289/264</td>
<td>0/38</td>
</tr>
<tr>
<td></td>
<td>WBGA_W2 vs. RWGA</td>
<td>134/122</td>
<td>366/354</td>
<td>0/24</td>
</tr>
<tr>
<td></td>
<td>WBGA_MO vs. RWGA</td>
<td>192/174</td>
<td>308/295</td>
<td>0/31</td>
</tr>
</tbody>
</table>

6.1.2.1. Results and Analysis for Research Question 2 and 3

To answer RQ2 and RQ3, we compared WBGA_W1, WBGA_W2, WBGA-MO, RWGA and RS based on the mean fitness values for each problem (\( MFV_{i,j} \), where 10 \( \leq i \leq 1000 \) with an increment of 10 and 5 \( \leq j \leq 45 \) with an increment of 10, i.e., there are 500 \( MFV \) values for the 500 artificial problems).

RQ2: All the selected weight-based GAs significantly outperformed RS for finding optimal solutions in most of the problems. For instance, WBGA-MO has better performance than RS for 445 problems and the performance is significantly better for 426 problems. WBGA-MO has worse performance than RS for 55 problems (45 of them were significantly worse). There were no significant differences between WBGA-MO and RS for 29 problems (TABLE VIII).

RQ3: RWGA achieved the best performance among all the other algorithms in most of the problems (the performance of RWGA is better than the other algorithms for 351.7 problems on average \( \frac{381+366+308}{3} = 351.7 \) and significantly better for 340 problems on average \( \frac{371+354+295}{3} = 340 \)). WBGA_W1 and WBGA_W2 have almost equivalent
performance, whose performance stays at the second level. Moreover, WBGA-MO achieved better performance than WBGA_W1 and WBGA_W2 (WBGA-MO outperformed WBGA_W1 and WBGA_W2 for 302 and 289 problems respectively and significantly outperformed for 287 and 264 problems respectively).

6.1.3. Timing Analysis for Weight-based GAs

In addition, we report the average time taken by the three weight-based GAs per run (i.e., WBGA, WBGA-MO and RWGA) as compared with RS (shown in Table 9). In addition, the Kruskal–Wallis test was conducted to evaluate whether there are significant differences among the running time of the weight-based GAs as compared with RS (100 run in our case). The p-value obtained by the Kruskal–Wallis test is 0.39, which shows that there are no significant differences among the running time of the weight-based GAs and RS. Based on the results, we can conclude that the selected weight-based GAs took similar running time as compared with RS in terms of finding an optimal solution for our test minimization problem.

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>WBGA_W1</th>
<th>WBGA_W2</th>
<th>WBGA-MO</th>
<th>RWGA</th>
<th>RS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time (s)</td>
<td>2.01</td>
<td>2.11</td>
<td>1.94</td>
<td>1.37</td>
<td>1.13</td>
</tr>
</tbody>
</table>

6.2. Results and Analysis for the Best Weight-Based GA (RWGA) and the Other Selected Search Algorithms

This section presents the results and related analysis for our industrial case study and the designed artificial problems using the best weight-based GA (RWGA) and the other selected algorithms.

6.2.1. Results and Analysis for Industrial Case Study

In this section, we summarize the results in the following sections for the industrial case study.

6.2.1.1. Analysis for Research Question 1

We did one sample Mann Whitney test since we compared the results by different algorithms with one set of fixed values for TMP, FPC, FDC, OET and OFV (i.e., overall fitness value). The Table 10 and Table 11 show that the performance of algorithms as
compared with the threshold values for each objective (i.e., TMP, FPC, FDC, OET and OFV). Based on the results, we answer RQ1 as follows.

For TMP, all the selected algorithms except RS have higher probability to achieve better results than the given threshold for TMP since most of the $\hat{A}_{12}$ values are greater than 0.5 and there are no significant differences between the results and the threshold (all the $p$-values are greater than 0.05). However, all the values of $\hat{A}_{12}$ obtained by RS are less than 0.5 and all the $p$-values are less than 0.05, i.e., RS has significantly less probability to obtain better results than the given threshold (Table 10).

### Table 10. Results for Comparing the Eight Algorithms with the Given Thresholds for Each Objective*

<table>
<thead>
<tr>
<th>Products</th>
<th>Algorithms</th>
<th>$\text{TMP} : 0.8$</th>
<th>$\text{FPC} : 0.8$</th>
<th>$\text{FDC} : 0.85$</th>
<th>$\text{OET} : 2h$</th>
</tr>
</thead>
<tbody>
<tr>
<td>C20</td>
<td>RWGA</td>
<td>0.69</td>
<td>0.19</td>
<td>0.76</td>
<td>0.19</td>
</tr>
<tr>
<td></td>
<td>NSGA-II</td>
<td>0.72</td>
<td>0.23</td>
<td>0.65</td>
<td>0.33</td>
</tr>
<tr>
<td></td>
<td>MOCell</td>
<td>0.65</td>
<td>0.32</td>
<td>0.61</td>
<td>0.34</td>
</tr>
<tr>
<td></td>
<td>SPEA2</td>
<td>0.59</td>
<td>0.18</td>
<td>0.38</td>
<td>0.04</td>
</tr>
<tr>
<td></td>
<td>PAES</td>
<td>0.68</td>
<td>0.24</td>
<td>0.54</td>
<td>0.27</td>
</tr>
<tr>
<td></td>
<td>SMPSO</td>
<td>0.57</td>
<td>0.32</td>
<td>0.62</td>
<td>0.31</td>
</tr>
<tr>
<td></td>
<td>CellDE</td>
<td>0.63</td>
<td>0.35</td>
<td>0.68</td>
<td>0.34</td>
</tr>
<tr>
<td></td>
<td>RS</td>
<td>0.22</td>
<td>0.04</td>
<td>0.34</td>
<td>0.02</td>
</tr>
<tr>
<td>C40</td>
<td>RWGA</td>
<td>0.68</td>
<td>0.08</td>
<td>0.65</td>
<td>0.10</td>
</tr>
<tr>
<td></td>
<td>NSGA-II</td>
<td>0.55</td>
<td>0.22</td>
<td>0.65</td>
<td>0.18</td>
</tr>
<tr>
<td></td>
<td>MOCell</td>
<td>0.72</td>
<td>0.31</td>
<td>0.58</td>
<td>0.41</td>
</tr>
<tr>
<td></td>
<td>SPEA2</td>
<td>0.57</td>
<td>0.25</td>
<td>0.31</td>
<td>0.04</td>
</tr>
<tr>
<td></td>
<td>PAES</td>
<td>0.61</td>
<td>0.34</td>
<td>0.62</td>
<td>0.19</td>
</tr>
<tr>
<td></td>
<td>SMPSO</td>
<td>0.65</td>
<td>0.17</td>
<td>0.57</td>
<td>0.23</td>
</tr>
<tr>
<td></td>
<td>CellDE</td>
<td>0.72</td>
<td>0.19</td>
<td>0.61</td>
<td>0.31</td>
</tr>
<tr>
<td>C60</td>
<td>RWGA</td>
<td>0.67</td>
<td>0.13</td>
<td>0.71</td>
<td>0.17</td>
</tr>
<tr>
<td></td>
<td>NSGA-II</td>
<td>0.61</td>
<td>0.32</td>
<td>0.55</td>
<td>0.23</td>
</tr>
<tr>
<td></td>
<td>MOCell</td>
<td>0.54</td>
<td>0.22</td>
<td>0.63</td>
<td>0.18</td>
</tr>
<tr>
<td></td>
<td>SPEA2</td>
<td>0.65</td>
<td>0.34</td>
<td>0.49</td>
<td>0.17</td>
</tr>
<tr>
<td></td>
<td>PAES</td>
<td>0.47</td>
<td>0.41</td>
<td>0.57</td>
<td>0.23</td>
</tr>
<tr>
<td></td>
<td>SMPSO</td>
<td>0.49</td>
<td>0.25</td>
<td>0.63</td>
<td>0.29</td>
</tr>
<tr>
<td></td>
<td>CellDE</td>
<td>0.54</td>
<td>0.18</td>
<td>0.59</td>
<td>0.32</td>
</tr>
<tr>
<td></td>
<td>RS</td>
<td>0.39</td>
<td>0.04</td>
<td>0.32</td>
<td>0.02</td>
</tr>
<tr>
<td>C90</td>
<td>RWGA</td>
<td>0.70</td>
<td>0.12</td>
<td>0.64</td>
<td>0.17</td>
</tr>
<tr>
<td></td>
<td>NSGA-II</td>
<td>0.58</td>
<td>0.25</td>
<td>0.69</td>
<td>0.17</td>
</tr>
<tr>
<td></td>
<td>MOCell</td>
<td>0.61</td>
<td>0.27</td>
<td>0.54</td>
<td>0.21</td>
</tr>
<tr>
<td></td>
<td>SPEA2</td>
<td>0.48</td>
<td>0.14</td>
<td>0.47</td>
<td>0.09</td>
</tr>
<tr>
<td></td>
<td>PAES</td>
<td>0.54</td>
<td>0.11</td>
<td>0.49</td>
<td>0.15</td>
</tr>
<tr>
<td></td>
<td>SMPSO</td>
<td>0.63</td>
<td>0.35</td>
<td>0.55</td>
<td>0.23</td>
</tr>
<tr>
<td></td>
<td>CellDE</td>
<td>0.68</td>
<td>0.29</td>
<td>0.59</td>
<td>0.17</td>
</tr>
<tr>
<td></td>
<td>RS</td>
<td>0.22</td>
<td>0.02</td>
<td>0.34</td>
<td>0.01</td>
</tr>
</tbody>
</table>

*A: $\hat{A}_{12}$; $p$: $p$-value. All $p$-values less than 0.05 are identified as bold.

### Table 11. Results for Comparing the Eight Algorithms with the Given Thresholds for OVF*

<table>
<thead>
<tr>
<th>Pair of Algorithms</th>
<th>C20</th>
<th>C40</th>
<th>C60</th>
<th>C90</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$A$</td>
<td>$p$</td>
<td>$A$</td>
<td>$p$</td>
</tr>
<tr>
<td>RWGA vs. Thresholds</td>
<td>0.62</td>
<td>0.32</td>
<td>0.59</td>
<td>0.18</td>
</tr>
<tr>
<td>NSGA-II vs. Thresholds</td>
<td>0.71</td>
<td>0.38</td>
<td>0.62</td>
<td>0.21</td>
</tr>
<tr>
<td>MOCell vs. Thresholds</td>
<td>0.58</td>
<td>0.24</td>
<td>0.62</td>
<td>0.19</td>
</tr>
<tr>
<td>SPEA2 vs. Thresholds</td>
<td>0.38</td>
<td>0.11</td>
<td>0.31</td>
<td>0.04</td>
</tr>
<tr>
<td>PAES vs. Thresholds</td>
<td>0.55</td>
<td>0.23</td>
<td>0.58</td>
<td>0.16</td>
</tr>
</tbody>
</table>
As for FPC and FDC, all the selected algorithms except SPEA2 and RS have higher chances to obtain better results than the given thresholds when required since almost all the values of $\hat{A}_{12}$ are greater than 0.5 and there are no significant differences (all the p-values are greater than 0.05). For SPEA2, all the $\hat{A}_{12}$ values are less than 0.5 and half of the p-values are less than 0.05, i.e., SPEA2 has significantly less probability to yield better results than the given thresholds. RS also has significantly less chance to be used for achieving an acceptable level of FPC and FDC since all the values of $\hat{A}_{12}$ are less than 0.5 and most of the p-values are less than 0.05 (Table 10).

For OET, most of the $\hat{A}_{12}$ values achieved by the selected algorithms except RS are less than 0.5 and more than half of the p-values are less than 0.05, i.e., they have significantly higher chances to meet the time budget within the given threshold (2 hours). Moreover, all the $\hat{A}_{12}$ values obtained by RS are greater than 0.5 and most of the p-values are less than 0.05, i.e., the whole execution time achieved by RS has higher chance to be significantly more than 2 hours (Table 10).

When comparing results for OFV considering all the objectives, the selected algorithms except SPEA2 and RS have higher probability to obtain better OFV when compared with the given thresholds since all the $\hat{A}_{12}$ values are greater than 0.5 and there are no significant differences (all the p-values are greater than 0.05). As for SPEA2, the obtained OFV has significantly less chance to be equivalent as the given thresholds (most of the $\hat{A}_{12}$ values are less than 0.5 and half of the p-values are less than 0.05). For RS, the obtained OFV has also higher chance to be significantly less than the thresholds since all the $\hat{A}_{12}$ values are less than 0.5 and all the p-values are less than 0.05 (Table 11).

We also report the number of solutions that satisfy all the given thresholds (i.e., 0.8 for TMP, 0.8 for FPC, 0.85 for FDC and 2h for OET) obtained by each selected algorithm as shown in Table 12. Based on the results, we observe that weight-based GAs can manage to find one solution which balances all the objectives together (i.e., maximum of the OFV value) while the Pareto-based algorithms can obtain a set of non-dominated solutions, e.g., eleven non-dominated solutions are obtained by using NSGA-II. Notice RS can not manage to find a solution which can satisfy the required thresholds for all the objectives.

<table>
<thead>
<tr>
<th>SMPSO vs. Thresholds</th>
<th>0.65</th>
<th>0.27</th>
<th>0.72</th>
<th>0.34</th>
<th>0.69</th>
<th>0.27</th>
<th>0.74</th>
<th>0.25</th>
</tr>
</thead>
<tbody>
<tr>
<td>CellDE vs. Thresholds</td>
<td>0.59</td>
<td>0.18</td>
<td>0.63</td>
<td>0.22</td>
<td>0.74</td>
<td>0.39</td>
<td>0.69</td>
<td>0.35</td>
</tr>
<tr>
<td>RS vs. Thresholds</td>
<td>0.28</td>
<td>0.02</td>
<td>0.33</td>
<td>0.02</td>
<td>0.39</td>
<td>0.01</td>
<td>0.27</td>
<td>0.01</td>
</tr>
</tbody>
</table>

* All p-values less than 0.05 are identified as bold.

Table 12. Number of solutions obtained by each search algorithm
6.2.1.2. Analysis for Research Question 2 and 3

Similarly as Section 6.1.1.2, for the Saturn products, all the p-values obtained by the Kruskal–Wallis test (together with the Bonferroni Correction) are less than 0.05 for each objective and OFV (0.009 for TMP, 0.031 for FPC, 0.015 for FDC, 0.024 for OET and 0.016 for OFV), showing that there are significant differences between the performance of the selected search algorithms for each objective and OFV. Therefore, we further performed the Vargha and Delaney statistics and the Mann-Whitney U test to compare each pair of the eight search algorithms as shown in Table 13 (including four sub tables related with each product respectively) and Table 14. In total, we have \( C_2^8 = 28 \) pairs to compare for the eight selected search algorithms. The obtained results are first compared for each objective and then compared for OFV considering all the objectives (shown as Table 13 and Table 14, respectively). Based on the results, we answer the RQ2 and RQ3 as follows.

RQ2: For TMP, FPC, FDC and OET, we first compared the performance of the selected algorithms with RS. The results show that all the selected search algorithms have significantly better performance than RS for TMP and OET since all the \( \hat{A}_{12} \) values for TMP are greater than 0.5, and all the \( \hat{A}_{12} \) values for OET are less than 0.5, and most of the p-values are less than 0.05. For FPC and FDC, all the selected algorithms except SPEA2 achieve significantly better performance than RS since almost all the \( \hat{A}_{12} \) values are greater than 0.5 and most of the p-values are less than 0.05. SPEA2 has almost equivalent performance as RS for FPC and FDC since most of the \( \hat{A}_{12} \) values are close to 0.5 and the p-values are greater than 0.05 (Table 13).

Table 13. Results for Comparing the Eight Algorithms for Each Objective*

<table>
<thead>
<tr>
<th>Pair of Algorithms</th>
<th>( \text{TMP: } 0.8 )</th>
<th>( \text{FPC: } 0.8 )</th>
<th>( \text{FDC: } 0.85 )</th>
<th>( \text{OET: } 2h )</th>
</tr>
</thead>
<tbody>
<tr>
<td>RWGA vs. NSGA-II</td>
<td>0.43</td>
<td>0.11</td>
<td>0.34</td>
<td>0.04</td>
</tr>
<tr>
<td>RWGA vs. MOCell</td>
<td>0.51</td>
<td>0.12</td>
<td>0.37</td>
<td>0.04</td>
</tr>
<tr>
<td>RWGA vs. SPEA2</td>
<td>0.48</td>
<td>0.11</td>
<td>0.57</td>
<td>0.07</td>
</tr>
<tr>
<td>RWGA vs. PAES</td>
<td>0.47</td>
<td>0.21</td>
<td>0.45</td>
<td>0.18</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>( A )</th>
<th>( p )</th>
<th>( A )</th>
<th>( p )</th>
<th>( A )</th>
<th>( p )</th>
</tr>
</thead>
<tbody>
<tr>
<td>RWGA vs. NSGA-II</td>
<td>0.43</td>
<td>0.11</td>
<td>0.34</td>
<td>0.04</td>
<td>0.40</td>
<td>0.08</td>
</tr>
<tr>
<td>RWGA vs. MOCell</td>
<td>0.51</td>
<td>0.12</td>
<td>0.37</td>
<td>0.04</td>
<td>0.36</td>
<td>0.04</td>
</tr>
<tr>
<td>RWGA vs. SPEA2</td>
<td>0.48</td>
<td>0.11</td>
<td>0.57</td>
<td>0.07</td>
<td>0.29</td>
<td>0.03</td>
</tr>
<tr>
<td>RWGA vs. PAES</td>
<td>0.47</td>
<td>0.21</td>
<td>0.45</td>
<td>0.18</td>
<td>0.54</td>
<td>0.25</td>
</tr>
<tr>
<td>Pair of Algorithms</td>
<td>Objectives</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>--------------------</td>
<td>------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>TPC</strong>: 0.8</td>
<td><strong>FPC</strong>: 0.8</td>
<td><strong>FDC</strong>: 0.85</td>
<td><strong>OET</strong>: 2h</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RWGA vs. NSGA-II</td>
<td>0.48</td>
<td>0.21</td>
<td>0.46</td>
<td>0.17</td>
<td>0.38</td>
<td>0.03</td>
</tr>
<tr>
<td>RWGA vs. MOSCell</td>
<td>0.49</td>
<td>0.12</td>
<td>0.41</td>
<td>0.07</td>
<td>0.32</td>
<td>0.03</td>
</tr>
<tr>
<td>RWGA vs. SPEA2</td>
<td>0.52</td>
<td>0.23</td>
<td>0.52</td>
<td>0.18</td>
<td>0.31</td>
<td>0.03</td>
</tr>
<tr>
<td>RWGA vs. PAES</td>
<td>0.49</td>
<td>0.16</td>
<td>0.53</td>
<td>0.20</td>
<td>0.52</td>
<td>0.21</td>
</tr>
<tr>
<td>RWGA vs. SMPSO</td>
<td>0.47</td>
<td>0.19</td>
<td>0.48</td>
<td>0.27</td>
<td>0.45</td>
<td>0.29</td>
</tr>
<tr>
<td>RWGA vs. CellIDE</td>
<td>0.32</td>
<td>0.02</td>
<td>0.47</td>
<td>0.14</td>
<td>0.55</td>
<td>0.20</td>
</tr>
<tr>
<td>RWGA vs. RS</td>
<td>0.82</td>
<td>0.01</td>
<td>0.76</td>
<td>0.02</td>
<td>0.69</td>
<td>0.02</td>
</tr>
<tr>
<td>NSGA-II vs. MOSCell</td>
<td>0.48</td>
<td>0.20</td>
<td>0.41</td>
<td>0.16</td>
<td>0.69</td>
<td>0.02</td>
</tr>
<tr>
<td>NSGA-II vs. SPEA2</td>
<td>0.51</td>
<td>0.11</td>
<td>0.53</td>
<td>0.01</td>
<td>0.73</td>
<td>0.01</td>
</tr>
<tr>
<td>NSGA-II vs. PAES</td>
<td>0.52</td>
<td>0.16</td>
<td>0.55</td>
<td>0.18</td>
<td>0.69</td>
<td>0.02</td>
</tr>
<tr>
<td>NSGA-II vs. SMPSO</td>
<td>0.49</td>
<td>0.20</td>
<td>0.52</td>
<td>0.20</td>
<td>0.64</td>
<td>0.03</td>
</tr>
<tr>
<td>NSGA-II vs. CellIDE</td>
<td>0.29</td>
<td>0.02</td>
<td>0.49</td>
<td>0.09</td>
<td>0.72</td>
<td>0.01</td>
</tr>
<tr>
<td>NSGA-II vs. RS</td>
<td>0.89</td>
<td>0.02</td>
<td>0.81</td>
<td>0.01</td>
<td>0.77</td>
<td>0.01</td>
</tr>
<tr>
<td>MOSCell vs. SPEA2</td>
<td>0.54</td>
<td>0.19</td>
<td>0.77</td>
<td>0.01</td>
<td>0.62</td>
<td>0.06</td>
</tr>
<tr>
<td>MOSCell vs. PAES</td>
<td>0.49</td>
<td>0.09</td>
<td>0.73</td>
<td>0.01</td>
<td>0.56</td>
<td>0.09</td>
</tr>
<tr>
<td>MOSCell vs. SMPSO</td>
<td>0.51</td>
<td>0.23</td>
<td>0.69</td>
<td>0.02</td>
<td>0.53</td>
<td>0.18</td>
</tr>
<tr>
<td>MOSCell vs. CellIDE</td>
<td>0.29</td>
<td>0.01</td>
<td>0.68</td>
<td>0.03</td>
<td>0.54</td>
<td>0.27</td>
</tr>
<tr>
<td>MOSCell vs. RS</td>
<td>0.73</td>
<td>0.02</td>
<td>0.76</td>
<td>0.01</td>
<td>0.73</td>
<td>0.01</td>
</tr>
<tr>
<td>SPEA2 vs. PAES</td>
<td>0.43</td>
<td>0.22</td>
<td>0.31</td>
<td>0.01</td>
<td>0.29</td>
<td>0.01</td>
</tr>
<tr>
<td>SPEA2 vs. SMPSO</td>
<td>0.49</td>
<td>0.31</td>
<td>0.36</td>
<td>0.02</td>
<td>0.36</td>
<td>0.04</td>
</tr>
<tr>
<td>SPEA2 vs. CellIDE</td>
<td>0.32</td>
<td>0.01</td>
<td>0.28</td>
<td>0.01</td>
<td>0.35</td>
<td>0.02</td>
</tr>
<tr>
<td>SPEA2 vs. RS</td>
<td>0.73</td>
<td>0.01</td>
<td>0.55</td>
<td>0.13</td>
<td>0.47</td>
<td>0.22</td>
</tr>
<tr>
<td>PAES vs. SMPSO</td>
<td>0.41</td>
<td>0.13</td>
<td>0.52</td>
<td>0.23</td>
<td>0.54</td>
<td>0.11</td>
</tr>
<tr>
<td>PAES vs. CellIDE</td>
<td>0.44</td>
<td>0.12</td>
<td>0.49</td>
<td>0.31</td>
<td>0.54</td>
<td>0.22</td>
</tr>
<tr>
<td>PAES vs. RS</td>
<td>0.75</td>
<td>0.01</td>
<td>0.74</td>
<td>0.01</td>
<td>0.66</td>
<td>0.02</td>
</tr>
<tr>
<td>SMPSO vs. CellIDE</td>
<td>0.28</td>
<td>0.01</td>
<td>0.51</td>
<td>0.36</td>
<td>0.48</td>
<td>0.27</td>
</tr>
<tr>
<td>SMPSO vs. RS</td>
<td>0.69</td>
<td>0.02</td>
<td>0.73</td>
<td>0.01</td>
<td>0.77</td>
<td>0.01</td>
</tr>
<tr>
<td>CellIDE vs. RS</td>
<td>0.76</td>
<td>0.01</td>
<td>0.74</td>
<td>0.01</td>
<td>0.70</td>
<td>0.01</td>
</tr>
</tbody>
</table>

3) Product C60

<table>
<thead>
<tr>
<th>Pair of Algorithms</th>
<th>Objectives</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TPC</strong>: 0.8</td>
<td><strong>FPC</strong>: 0.8</td>
</tr>
<tr>
<td>RWGA vs. NSGA-II</td>
<td>0.52</td>
</tr>
<tr>
<td>RWGA vs. MOSCell</td>
<td>0.53</td>
</tr>
<tr>
<td>RWGA vs. SPEA2</td>
<td>0.48</td>
</tr>
<tr>
<td>RWGA vs. PAES</td>
<td>0.51</td>
</tr>
<tr>
<td>RWGA vs. SMPSO</td>
<td>0.46</td>
</tr>
<tr>
<td>RWGA vs. CellIDE</td>
<td>0.34</td>
</tr>
</tbody>
</table>
### Table 14. Results for Comparing the Eight Algorithms for OFV*

<table>
<thead>
<tr>
<th>Pair of Algorithms</th>
<th>C20</th>
<th>C40</th>
<th>C60</th>
<th>C90</th>
</tr>
</thead>
<tbody>
<tr>
<td>RWGA vs. NSGA-II</td>
<td>0.51</td>
<td>0.22</td>
<td>0.62</td>
<td>0.02</td>
</tr>
<tr>
<td>RWGA vs. MOCel</td>
<td>0.54</td>
<td>0.23</td>
<td>0.49</td>
<td>0.32</td>
</tr>
<tr>
<td>NSGA-II vs. SPEA2</td>
<td>0.53</td>
<td>0.14</td>
<td>0.75</td>
<td>0.01</td>
</tr>
<tr>
<td>NSGA-II vs. PAES</td>
<td>0.57</td>
<td>0.09</td>
<td>0.31</td>
<td>0.33</td>
</tr>
<tr>
<td>NSGA-II vs. SMPSO</td>
<td>0.46</td>
<td>0.15</td>
<td>0.57</td>
<td>0.11</td>
</tr>
<tr>
<td>NSGA-II vs. CellDE</td>
<td>0.32</td>
<td>0.05</td>
<td>0.53</td>
<td>0.16</td>
</tr>
<tr>
<td>NSGA-II vs. RS</td>
<td>0.57</td>
<td>0.03</td>
<td>0.65</td>
<td>0.01</td>
</tr>
<tr>
<td>MOCell vs. SPEA2</td>
<td>0.54</td>
<td>0.22</td>
<td>0.67</td>
<td>0.02</td>
</tr>
<tr>
<td>MOCell vs. PAES</td>
<td>0.48</td>
<td>0.24</td>
<td>0.72</td>
<td>0.01</td>
</tr>
<tr>
<td>MOCell vs. SMPSO</td>
<td>0.51</td>
<td>0.29</td>
<td>0.33</td>
<td>0.01</td>
</tr>
<tr>
<td>MOCell vs. CellDE</td>
<td>0.40</td>
<td>0.05</td>
<td>0.69</td>
<td>0.02</td>
</tr>
<tr>
<td>MOCel vs. RS</td>
<td>0.74</td>
<td>0.01</td>
<td>0.77</td>
<td>0.01</td>
</tr>
<tr>
<td>PAES vs. SMPSO</td>
<td>0.48</td>
<td>0.27</td>
<td>0.53</td>
<td>0.19</td>
</tr>
<tr>
<td>PAES vs. CellDE</td>
<td>0.20</td>
<td>0.01</td>
<td>0.47</td>
<td>0.31</td>
</tr>
<tr>
<td>PAES vs. RS</td>
<td>0.73</td>
<td>0.01</td>
<td>0.76</td>
<td>0.01</td>
</tr>
<tr>
<td>SMPSO vs. CellDE</td>
<td>0.72</td>
<td>0.01</td>
<td>0.53</td>
<td>0.29</td>
</tr>
<tr>
<td>SMPSO vs. RS</td>
<td>0.70</td>
<td>0.01</td>
<td>0.68</td>
<td>0.02</td>
</tr>
<tr>
<td>CellDE vs. RS</td>
<td>0.73</td>
<td>0.01</td>
<td>0.75</td>
<td>0.01</td>
</tr>
</tbody>
</table>

### 4) Product C90

<table>
<thead>
<tr>
<th>Pair of Algorithms</th>
<th>Objectives</th>
<th>OFV: 2h</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>T/F: 0.8</td>
<td>FPC: 0.8</td>
</tr>
<tr>
<td>RWGA vs. NSGA-II</td>
<td>0.22</td>
<td>0.01</td>
</tr>
<tr>
<td>RWGA vs. MOCell</td>
<td>0.53</td>
<td>0.16</td>
</tr>
<tr>
<td>RWGA vs. SPEA2</td>
<td>0.55</td>
<td>0.14</td>
</tr>
<tr>
<td>RWGA vs. PAES</td>
<td>0.52</td>
<td>0.19</td>
</tr>
<tr>
<td>RWGA vs. SMPSO</td>
<td>0.44</td>
<td>0.11</td>
</tr>
<tr>
<td>RWGA vs. CellDE</td>
<td>0.38</td>
<td>0.02</td>
</tr>
<tr>
<td>RWGA vs. RS</td>
<td>0.69</td>
<td>0.01</td>
</tr>
<tr>
<td>NSGA-II vs. MOCel</td>
<td>0.53</td>
<td>0.15</td>
</tr>
<tr>
<td>NSGA-II vs. SPEA2</td>
<td>0.49</td>
<td>0.37</td>
</tr>
<tr>
<td>NSGA-II vs. PAES</td>
<td>0.57</td>
<td>0.14</td>
</tr>
<tr>
<td>NSGA-II vs. SMPSO</td>
<td>0.49</td>
<td>0.33</td>
</tr>
<tr>
<td>NSGA-II vs. CellDE</td>
<td>0.38</td>
<td>0.06</td>
</tr>
<tr>
<td>NSGA-II vs. RS</td>
<td>0.91</td>
<td>0.02</td>
</tr>
<tr>
<td>MOCel vs. SPEA2</td>
<td>0.57</td>
<td>0.12</td>
</tr>
<tr>
<td>MOCel vs. PAES</td>
<td>0.52</td>
<td>0.21</td>
</tr>
<tr>
<td>MOCel vs. SMPSO</td>
<td>0.55</td>
<td>0.09</td>
</tr>
<tr>
<td>MOCel vs. CellDE</td>
<td>0.31</td>
<td>0.02</td>
</tr>
<tr>
<td>MOCel vs. RS</td>
<td>0.68</td>
<td>0.02</td>
</tr>
<tr>
<td>SPEA2 vs. PAES</td>
<td>0.54</td>
<td>0.16</td>
</tr>
<tr>
<td>SPEA2 vs. SMPSO</td>
<td>0.52</td>
<td>0.29</td>
</tr>
<tr>
<td>SPEA2 vs. CellDE</td>
<td>0.26</td>
<td>0.01</td>
</tr>
<tr>
<td>SPEA2 vs. RS</td>
<td>0.75</td>
<td>0.01</td>
</tr>
<tr>
<td>PAES vs. SMPSO</td>
<td>0.20</td>
<td>0.01</td>
</tr>
<tr>
<td>PAES vs. CellDE</td>
<td>0.54</td>
<td>0.19</td>
</tr>
<tr>
<td>SMPSO vs. CellDE</td>
<td>0.68</td>
<td>0.02</td>
</tr>
<tr>
<td>SMPSO vs. RS</td>
<td>0.32</td>
<td>0.03</td>
</tr>
<tr>
<td>CellDE vs. RS</td>
<td>0.73</td>
<td>0.01</td>
</tr>
<tr>
<td>CellDE vs. RS</td>
<td>0.78</td>
<td>0.01</td>
</tr>
</tbody>
</table>

*OFV: Objective Function Value*
When comparing the $OFV$ results considering all objectives for the algorithms and RS, we can see that six of them (i.e., RWGA, NSGA-II, MOCell, PAES, SMPSO and CellDE) have significantly better performance than RS since all the $\hat{A}_{12}$ values are greater than 0.5 and all the $p$-values are less than 0.05. SPEA2 has almost equivalent performance as RS since most of the $\hat{A}_{12}$ values are closer to 0.5 and the $p$-values are greater than 0.05 (TABLE 14).

**RQ3:** For $TMP$, CellDE achieved the best performance than the other algorithms since most of the $\hat{A}_{12}$ values are greater than 0.5 and the $p$-values are less than 0.05. Similarly, MOCell, NSGA-II and SPEA2 have the best performance for $FPC$, $FDC$ and $OET$ among all the algorithms, respectively (Table 13).

When comparing the $OFV$ results considering all objectives together, RWGA performs significantly better among all the search algorithms since all the $\hat{A}_{12}$ values are greater than 0.5 and most of $p$-values are less than 0.05. Moreover, SPEA2 has significantly worse performance than the other six search algorithms since most of the $\hat{A}_{12}$ values are less than 0.5 and $p$-values are greater than 0.05 (Table 14).

In general, we can conclude that different algorithms can achieve the best performance for different objectives (e.g., CellDE for the objective $TMP$) but RWGA has significantly better performance than the other search algorithms when considering all the objectives together in general.

### 6.2.2. Results and Analysis for Artificial Problems

In this section, we summarize the results as the following sections for the designed 500 artificial problems.
6.2.2.1. Results and Analysis for Research Question 2 and 3

To answer RQ2 and RQ3, the Kruskal–Wallis test (together with the Bonferroni Correction) was first performed for the results of the eight search algorithms and we obtained a $p$-value of 0.024 (less than 0.05) showing that there exist significant differences between the performances of the selected algorithms. Therefore, we further compared the algorithms (i.e., RWGA, NSGA-II, MOCeU, SPEA2, PAES, SMPSO, CellDE) with RS and then compared each pair of them based on mean fitness value ($MFV$) achieved after 25000 generations for each algorithm and each of the 500 problems. Recall that each problem was repeated for 100 times to account for random variation, i.e., $MFV$ is a mean for 100 values of $OFV$ after 100 runs (Section 5.1.3). Table 15 summarizes the results for Vargha and Delaney statistics without or with Whitney U test for the 500 artificial problems (Similarly as Table 8).

**RQ2:** RWGA, NSGA-II, MOCeU, PAES, SMPSO, and CellDE achieved significantly better performance than RS for finding optimal solutions for our minimization problem in most of the problems. For instance, RWGA has better performance than RS for 489 problems and the performance is significantly better for 480 problems. RWGA has worse performance than RS for 11 problems (none of them was significantly worse). There were no significant differences between RWGA and RS for 20 problems. The performance of SPEA2 is close to RS (SPEA2 is better than RS for 298 problems and significantly better for 251 problems. SPEA2 is worse than RS for 202 problems and significantly worse for 179 problems. Meanwhile, there is no significant difference between them for 70 problems) (Table 15).

**RQ3:** RWGA achieved the best performance among all the other six algorithms in most of the problems, i.e., the performance of RWGA is better than the other six algorithms for average 418.8 problems ($\frac{432+403+464+381+453+380}{6}=418.8$) and significantly better for 401.7 problems on average ($\frac{405+397+451+362+438+357}{6}=401.7$). NSGA-II, MOCeU, PAES, SMPSO, and CellDE have almost equivalent performance, whose performance stay at the second level. Finally, the performance of SPEA2 is worse than the above-mentioned algorithms, i.e., SPEA2 achieved the worst performance in the 500 artificial problems (SPEA2 is worse than the other search algorithms for 377.8 problems and significantly worse for 354.7 problems on average) (Table 15).
Table 15. Results for the Vargha and Delaney Statistics-artificial problems (without/with Whitney U test)

<table>
<thead>
<tr>
<th>RQ</th>
<th>Pair of Algorithms (A vs. B)</th>
<th>A&gt;B</th>
<th>A&lt;B</th>
<th>A=B</th>
</tr>
</thead>
<tbody>
<tr>
<td>RQ2</td>
<td>RWGA vs. RS</td>
<td>489/480</td>
<td>11/0</td>
<td>0/20</td>
</tr>
<tr>
<td></td>
<td>NSGA-II vs. RS</td>
<td>467/449</td>
<td>33/14</td>
<td>0/37</td>
</tr>
<tr>
<td></td>
<td>MOCell vs. RS</td>
<td>438/419</td>
<td>62/46</td>
<td>0/35</td>
</tr>
<tr>
<td></td>
<td>SPEA2 vs. RS</td>
<td>298/251</td>
<td>202/179</td>
<td>0/70</td>
</tr>
<tr>
<td></td>
<td>PAES vs. RS</td>
<td>430/414</td>
<td>70/59</td>
<td>0/27</td>
</tr>
<tr>
<td></td>
<td>SMPSO vs. RS</td>
<td>388/362</td>
<td>112/93</td>
<td>0/45</td>
</tr>
<tr>
<td></td>
<td>CellDE vs. RS</td>
<td>423/399</td>
<td>77/65</td>
<td>0/36</td>
</tr>
<tr>
<td>RQ3</td>
<td>RWGA vs. NSGA-II</td>
<td>432/405</td>
<td>68/57</td>
<td>0/32</td>
</tr>
<tr>
<td></td>
<td>RWGA vs. MOCell</td>
<td>403/397</td>
<td>97/85</td>
<td>0/18</td>
</tr>
<tr>
<td></td>
<td>RWGA vs. SPEA2</td>
<td>464/451</td>
<td>36/23</td>
<td>0/26</td>
</tr>
<tr>
<td></td>
<td>RWGA vs. PAES</td>
<td>381/362</td>
<td>119/104</td>
<td>0/24</td>
</tr>
<tr>
<td></td>
<td>RWGA vs. SMPSO</td>
<td>453/438</td>
<td>47/39</td>
<td>0/23</td>
</tr>
<tr>
<td></td>
<td>RWGA vs. CellDE</td>
<td>380/357</td>
<td>120/103</td>
<td>0/40</td>
</tr>
<tr>
<td></td>
<td>NSGA-II vs. MOCell</td>
<td>269/252</td>
<td>231/204</td>
<td>0/44</td>
</tr>
<tr>
<td></td>
<td>NSGA-II vs. SPEA2</td>
<td>398/377</td>
<td>102/86</td>
<td>0/47</td>
</tr>
<tr>
<td></td>
<td>NSGA-II vs. PAES</td>
<td>266/242</td>
<td>234/219</td>
<td>0/39</td>
</tr>
<tr>
<td></td>
<td>NSGA-II vs. SMPSO</td>
<td>226/219</td>
<td>273/241</td>
<td>0/40</td>
</tr>
<tr>
<td></td>
<td>NSGA-II vs. CellDE</td>
<td>245/227</td>
<td>259/239</td>
<td>0/34</td>
</tr>
<tr>
<td></td>
<td>MOCell vs. SPEA2</td>
<td>395/374</td>
<td>105/91</td>
<td>0/15</td>
</tr>
<tr>
<td></td>
<td>MOCell vs. PAES</td>
<td>302/281</td>
<td>198/167</td>
<td>0/52</td>
</tr>
<tr>
<td></td>
<td>MOCell vs. SMPSO</td>
<td>267/243</td>
<td>233/200</td>
<td>0/57</td>
</tr>
<tr>
<td></td>
<td>MOCell vs. CellDE</td>
<td>201/182</td>
<td>299/283</td>
<td>0/35</td>
</tr>
<tr>
<td></td>
<td>SPEA2 vs. PAES</td>
<td>170/145</td>
<td>330/307</td>
<td>0/48</td>
</tr>
<tr>
<td></td>
<td>SPEA2 vs. SMPSO</td>
<td>141/107</td>
<td>359/316</td>
<td>0/77</td>
</tr>
<tr>
<td></td>
<td>SPEA2 vs. CellDE</td>
<td>179/152</td>
<td>321/303</td>
<td>0/45</td>
</tr>
<tr>
<td></td>
<td>PAES vs. SMPSO</td>
<td>223/201</td>
<td>277/255</td>
<td>0/46</td>
</tr>
<tr>
<td></td>
<td>PAES vs. CellDE</td>
<td>198/182</td>
<td>302/270</td>
<td>0/48</td>
</tr>
<tr>
<td></td>
<td>SMPSO vs. CellDE</td>
<td>244/219</td>
<td>250/238</td>
<td>0/43</td>
</tr>
</tbody>
</table>

6.2.2.2. Results and Analysis for Research Question 4

To answer RQ4, we calculated the spearman’s rank correlation between mean fitness values for each algorithm with increasing number of features ($MFV_F$, where $10 \leq i \leq 1000$ with an increment of 10, Section 5.1.3) and associated test cases ($MFV_TC$, where $5 \leq j \leq 45$ with an increment of 10, Section 5.1.3). In this case, a fixed number of features is used to represent a product and each feature is associated with different number of test cases. So we have 100*5=500 artificial problems corresponding to minimizing the test suites for various products. Table 16 shows the results of correlation between $MFV_F$ and the increasing number of features, and between $MFV_TC$ with the increasing associated test cases for various algorithms using Spearman’s rank correlation ($\rho$). Recall that a higher value of $MFV_F$ or $MFV_TC$ represents a better performance of an algorithm.

**Increasing Number of Features:** For weight-based GAs (i.e., WBGA_W1, WBGA_W2, WBGA-MO and RWGA), we observed that the values of $MFV_F$ increased significantly as the increasing number of features since the values for Spearman’s $\rho$ are greater than 0 (0.52, 0.47, 0.69 and 0.74 respectively) and the $p$-value is less than 0.0001, i.e., the
performance increases significantly with the growth of features number. For SPEA2, PAES, SMPSO and CellIDE, the values of $MFV_F$ also increase but not significantly as the increasing features since Spearman’s $\rho$ is greater than 0 and $p$-values are greater than 0.05, i.e., the performances of SPEA2, PAES, SMPSO and CellIDE increase but not significantly when the number of features increases. For NSGA-II, MOCell and RS, the performances decrease but not significantly as the increasing number of features since the Spearman’s $\rho$ is less than 0 and $p$-values are greater than 0.05. In summary, we conclude that RWGA has the ability to solve a wide range of problems with gradually better performance and the other seven algorithms are not influenced significantly by the number of features (Table 16).

**Table 16.** Spearman’s Correlation Analysis for the Select Algorithms with the Increasing Number of Features and Associated Test Cases

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Increasing Number of Features</th>
<th>Increasing Number of Associated Test Cases</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Spearman $\rho$</td>
<td>Probs &gt;</td>
</tr>
<tr>
<td>WBGA_W1</td>
<td>0.52</td>
<td>&lt;0.0001</td>
</tr>
<tr>
<td>WBGA_W2</td>
<td>0.47</td>
<td>&lt;0.0001</td>
</tr>
<tr>
<td>WBGA-MO</td>
<td>0.69</td>
<td>&lt;0.0001</td>
</tr>
<tr>
<td>RWGA</td>
<td>0.74</td>
<td>&lt;0.0001</td>
</tr>
<tr>
<td>NSGA-II</td>
<td>-0.13</td>
<td>0.5546</td>
</tr>
<tr>
<td>MOCell</td>
<td>-0.11</td>
<td>0.6852</td>
</tr>
<tr>
<td>SPEA2</td>
<td>0.09</td>
<td>0.8019</td>
</tr>
<tr>
<td>PAES</td>
<td>0.02</td>
<td>0.7980</td>
</tr>
<tr>
<td>SMPSO</td>
<td>0.15</td>
<td>0.5742</td>
</tr>
<tr>
<td>CellIDE</td>
<td>0.08</td>
<td>0.7006</td>
</tr>
<tr>
<td>RS</td>
<td>-0.08</td>
<td>0.8704</td>
</tr>
</tbody>
</table>

**Increasing Number of Associated Test Cases:** Based on the results, we observed that for WBGA_W1, WBGA_W2, WBGA-MO, RWGA, SPEA2, PAES and SMPSO, the values of $MFV_TC$ increase but not significantly as the number of associated test cases increases since the Spearman’s $\rho$ is greater than 0 and $p$-values are greater than 0.05, i.e., the performances of WBGA_W1, WBGA_W2, WBGA-MO, RWGA, SPEA2, PAES and SMPSO increase but not significantly with the growth of associated test cases. For the other algorithms (NSGA-II, MOCell, CellIDE and RS), the values of $MFV_TC$ decrease but not significantly when the associated test cases increase, showing the performances of these algorithms decrease but not significantly as the increasing number of test cases. In summary, we can conclude even with the increase in the number of test cases, the performance of search algorithms is not influenced as shown in Table 16.

**6.2.3. Timing Analysis for RWGA and the Other Selected Search Algorithms**
Similarly, we also performed a timing analysis for the best weight-based GAs (i.e., RWGA) and the other kinds of selected search algorithms (e.g., NSGA-II) as compared with RS. Table 17 shows the average running time per run taken by each selected search algorithms. We also obtained a $p$-value of 0.63 with the Kruskal–Wallis test suggesting there are no significant differences between the running time of the selected search algorithms as compared with RS. Based on the results, we can conclude that all the selected search algorithms did not take significantly different time for running as compared with RS in terms of finding an optimal solution.

Table 17. Average running time for the selected weight-based search algorithms

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>RWGA</th>
<th>NSGA-II</th>
<th>MOCell</th>
<th>SPEA2</th>
<th>PAES</th>
<th>SMPSO</th>
<th>CellDE</th>
<th>RS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time (s)</td>
<td>1.37</td>
<td>1.54</td>
<td>1.23</td>
<td>1.98</td>
<td>2.06</td>
<td>2.19</td>
<td>1.45</td>
<td>1.13</td>
</tr>
</tbody>
</table>

### 7. Discussions

In this section, we discuss the results of the experiments based on our industrial case study and the 500 artificial problems. Section 7.1 provides insight on the results of RQ1-RQ3, while Section 7.2 discusses results of RQ4. Recall that RQ1-RQ3 address questions related to cost/effectiveness and performance of our minimization problem (as compared with RS), while RQ4 refers to the impact of the increasing number of features and the number of test cases on the performance of the selected multi-objective search algorithms (Section 5.1.1). Before getting deeper into the specific discussion related with the research questions, the key results are first summarized as below:

- Different algorithms achieve the best performance for each objective, i.e., CellDE for TMP, MOCell for FPC, NSGA-II for FDC and SPEA2 for OET;
- RWGA achieves significantly best performance among all the selected search algorithms when considering all the objectives together;
- The performance of weight-based GAs (e.g., RWGA) increases with the growth of features, but is not significantly influenced by the increasing number of associated test cases;
- All the other search algorithms are not significantly impacted by the increasing complexity of problems (increasing features and associated test cases).

#### 7.1. Discussions Related to RQ1-RQ3

Weight-based GAs require sets of weights (e.g., $w_1$, $w_2$ and $w_3$) that can be provided by
domain experts based on the testing requirements. Our results and analysis show that RWGA presents the best performance among the three weight-based GAs and RS. In fact, RWGA assigns weights dynamically during the search and thus the search is more efficiently guided towards the best weights, to reach the threshold values for TMP, FPC, FDC and OET.

When comparing the best weight-based GA (RWGA) and the other search algorithms, from our industrial case study, we observe that the solutions obtained by RWGA, NSGA-II, MOCell, PAES, SMPSO, CellDE can reach the expected thresholds for each objective (Section 6.2.1). SPEA2 managed to reach the expected thresholds for TMP and OET but not for FPC and FDC. RS cannot find optimal solutions for the thresholds (RQ1). When comparing the selected search algorithms with RS, all the algorithms except SPEA2 have significantly better performance than RS in finding optimal solutions for each objective function and OFV considering all the objectives. SPEA2 has significantly better performance than RS for TMP and OET but has almost equivalent performance as RS for FPC, FDC and OFV (RQ2).

Moreover, when studying the performance of each individual algorithm, different algorithms achieve the best performance for different objectives (e.g., CellDE for TMP, MOCell for FPC, NSGA-II for FDC and SPEA2 for OET). However, when combining all objectives together, RWGA has significantly better performance than all the other search algorithms (RQ3). For the artificial problems (RQ1-RQ3), we observe consistent results with our observations on the industrial case study. This difference in performance can be explained as follows: RWGA aims at balancing all the required objectives and finding an optimal solution by considering all the objectives together. Moreover, by dynamically changing weights at each generation, RWGA more thoroughly explores the search space of weights. This dynamic assignment of the weights (i.e., RWGA) is promising to find an optimal solution in a given number of generations, as compared to a strategy using only fixed weights for all the generations (e.g., WBGA_W1 and WBGA_W2). Recall from Section 2 that NSGA-II, MOCell, SPEA2, PAES, SMPSO and CellDE are used to seek a set of optimal non-dominated solutions. However, when considering all the objectives together (OFV), such non-dominated solutions may be worse than the solutions found by RWGA that aims at balancing all the objectives together. Notice that the solution obtained by RWGA may not dominate the solutions obtained by the other search algorithms (e.g.,
NSGA-II) but RWGA can manage to find the best solution with maximum overall fitness function \((OFV)\) for the minimization problem. However, if users only concern one of the most important objectives (e.g., \(FDC\)), our approach with corresponding tool (Section 8) will recommend the most suitable algorithm accordingly (e.g., MOCell for \(FDC\)).

### 7.2. Discussions Related to RQ4

When looking into the performance of each individual algorithm (RQ4) as the number of features and test cases increase (\(MFV_F\) and \(MFV_{TC}\) as shown in Fig. 2 and Fig. 3), we observe that RWGA presents the best performance and has the ability to solve a wider range of problems with gradually better performance (Fig. 2). In addition, when the number of test cases associated to each feature increases, the performance of RWGA is not significantly impacted (Fig. 3).

![Fig. 2. Mean Fitness Values for Number of Features](image-url)
Fig. 3. Mean Fitness Values for Number of Test Cases

To show the combined effect of the increase in the number of features and test cases on the performance of search algorithms, we show surface plots based on the mean fitness value (MFV, Section 5.1.3) in Fig. 4. From Fig. 4, we observe that RWGA reaches the best performance among all the algorithms. Moreover, the performance of WBGA and WBGA-MO also increases as the number of features increases. Finally, the performance of the other seven algorithms is not significantly impacted (notice that this observation is consistent with the results presented in Fig. 2 and Fig. 3). For instance, the performance of NSGA-II stays stable as the increasing number of the features and associated test cases. It is interesting to observe that the performance of SPEA2 remains worse than the other search algorithms without significantly changing with the growth of features and associated test cases. Notice that RS performs worse among all the selected search algorithms. These observations suggest that the selected search algorithms together with our proposed fitness function are scalable and sufficient for test minimization for the products of varying complexity. Section 7.2.1 and Section 7.2.2 further provide detailed explanations about the performance when the number of features and test cases increases.
Fig. 4. Surface Plots for All the Selected Algorithms

7.2.1. Increasing Number of Features

Three different types of solution spaces need to be first introduced to understand the rest of the section:

1) The Solution Space (black ellipse in Fig. 5) refers to the space of potential solutions for any search algorithm, depending on the number of test cases. When the number of features increases, then the solution space can either: A) increase when more test cases associated with new features are introduced; or B) remain the same when no more test
cases are introduced to the solution space (i.e., the existing test cases can cover the new features). For instance, in our industrial case study (Cisco), two call protocols H323 and SIP are designed and relevant test cases are developed for testing H323 and SIP. If a new call protocol based on H323 and SIP is designed, the existing test cases may be reused for testing such new protocol. In this case, no new test cases will be introduced to the solution space.

Fig. 5. An Overall Diagram for Three Types of Space

2) The Balanced Solution Space (red ellipse in Fig. 5) represents the space for optimal solutions, balancing all the objectives together. Weight-based GAs aims at guiding the search within this subspace in order to find optimal balanced solutions. When the number of features increases, the balanced solution space can either A) grow, B) remain the same or C) decrease.

3) The Non-dominated Solution Space (green ellipse in Fig. 5) refers to the space for non-dominated optimal solutions. The algorithms based on dominance theory (i.e., NSGA-II, MOCell, SPEA2, PAES, SMPSO and CellDE) can guide the search to find a set of non-dominated solutions within this solution space. Similarly, the non-dominated solution space can either A) grow, B) remain the same, or C) decrease with the growth of features.

Based on the above-mentioned definitions, our observations for experiments can be explained under the situation when the solution space, the balanced solution space and the non-dominated solution all grow as the increasing number of features. More specifically, the gradual improvement of performance of weight-based GAs (WBGA, WBGA-MO and RWGA) is interpreted as follows: Weight-based GAs (e.g., RWGA) guides the search to find optimal balanced solutions, considering all the objectives. Hence, the space of balanced solutions grows when the space for potential solutions grows. As a result, weight-based GAs (i.e., RWGA) easily finds an optimal balanced solution from a larger balanced solution space with the aim for balancing all the objectives together.

For the other algorithms based on the dominance theory (i.e., NSGA-II, MOCell, SPEA2, PAES, SMPSO and CellDE), the performance is not significantly influenced by
increasing number of features since these algorithms guide the search in finding a set of non-dominated solutions (usually one objective is the main concern). The overall fitness value \( \text{OFV} \), Section 5.1.3) does not change significantly even when the space for non-dominated solutions increases, as shown in (a) and (b) of Fig. 5. For instance, solution \( a \) and solution \( b \) are two obtained non-dominated solutions by NSGA-II with \( \text{TMP}=0.8, \text{FPC}=0.5, \text{FDC}=0.5, \text{OET}=0.5, \text{AEF}=0.5 \) and \( \text{TMP}=0.8, \text{FPC}=0.5, \text{FDC}=0.5, \text{OET}=0.5, \text{AEF}=0.5 \), respectively, i.e., the main objective for solution \( a \) is the percentage of test minimization (\( \text{TMP} \)) while the main concern for solution \( b \) is feature pairwise coverage (\( \text{FPC} \)). But for the overall fitness value (\( \text{OFV} \)) when considering all the objectives, solution \( a \) and solution \( b \) are equivalent since the \( \text{OFV} \) values for both solutions are the same (i.e., both values of \( \text{OFV} \) for solution \( a \) and solution \( b \) are 0.56, Section 5.1.3), which is the main reason the performance of NSGA-II, MOCell, SPEA2, PAES, SMPSO and CellDE stays stable as shown in Fig. 2 and Fig. 4.

For random search (RS), the performance remains the worst and is not influenced by the increasing features since it chooses solutions randomly without any guideline.

For the other combinations for the solution space, the balanced solution space and the non-dominated solution, the observations for experiments may not be the same as what we observed in our experiments. This further requires more focused experiments that assess the performance of the search algorithms with varying combinations of increase, decrease, and no change in the three types of solution spaces.

7.2.2. Increasing Number of Associated Test Cases

When increasing the number of test cases, we observed no significant impact on the performance for all the selected algorithms, which can be explained as follows: The complexity of problems mainly depends on the number of features (\( \text{FPC} \) measures how many pairs are covered, but the number of test cases is ignored) and the relationships between features and test cases are not complicated in our case (i.e., the only relationship between features and associated test cases is that one feature can be associated with a certain number of test cases as discussed in Section 4). As a result, when the complexity of a problem is mainly determined by the number of features, the number of associated test cases may have no significant impact on the performance of algorithms as shown in Fig. 3 and Fig. 4. However, in other cases, more complicated relationships may exist between
features and associated test cases, e.g., testing some features requires several steps and each step is covered by a certain number of test cases or execution of test cases depends on the execution of other test cases. Thus, the number of associated test cases may have significant impact on the performance of algorithms. We plan to further investigate this problem by using different case studies and other focused experiments in the future.

Concluding Remarks: When multi-objective test minimization for software product lines is sought, we recommend using CellDE, MOCell, NSGA-II and SPEA2 for the objectives \( TMP, FPC, FDC \) and \( OET \), respectively, when one objective has more importance than the others. In contrast, using RWGA is recommended when optimizing all the objectives together is sought. The latter situation happened in our industrial case study.

8. Automation

In this section, we present the tool support for test minimization using search algorithms along with our proposed fitness functions. The tool is called TEst Minimization with Search Algorithms (TEMSA) developed to minimize the test cases at the same time improving effectiveness and reducing predetermined cost based on various cost/effectiveness measures. TEMSA is implemented as a web-based application on top of the jMetal library using Java Sencha ExtJS (a JavaScript Framework for Rich Web Apps) [31]. The architecture of TEMSA is shown in Fig. 6. (TEMSA can be tried out at the website: http://zen-tools.com/TEMSA/)

![Fig. 6. Architecture for TEMSA](image)

The input of TEMSA is a set of selected features and test cases for testing a product as an XML file in our industrial case study. To use TEMSA, users have to provide an input file corresponding to our specific XML schema (the related XML schema and sample input XML file can be downloaded from our tool website). Cost measures (e.g., \( OET \)) and effectiveness measures (e.g., \( TMP, FPC, FDC \) and \( AEF \)) are integrated into TEMSA for
covering various objectives. Other objective measures can also be coded into TEMSA by formulating them as specific objective functions mathematically as we discussed in Section 3.2. For the test cases, each one includes three key attributes for cost/effectiveness measures, i.e., \( SucR \), \( EF \) and \( AET \) (Section 3.2), which are obtained from test execution history. Based on these inputs, TEMSA outputs a set of minimized test cases organized as another XML file which also conforms to our specific XML schema and can be transformed into any other schema when needed. In our context, the output can be put into test execution engine for scheduling and further executing for System Under Test (SUT). Afterwards, the output of executed test cases will be an input to update test execution history in the repository, such as \( SucR \), \( EF \) and \( AET \) for the executed test cases.

Moreover, our tool recommends different algorithms depending on test minimization objectives based on the results of our empirical study as shown in Fig. 7. First, a user selects an objective with highest importance (Activity A1) and TEMSA will automatically recommend the relevant best algorithm based on the selected objective. For instance, if the highest importance is to reduce the number of test cases (\( TMP \)), then CellIDE will be recommended by TEMSA (Activity A2). Similarly, if feature pairwise coverage (\( FPC \)), fault detection capability (\( FDC \)) and the allowed time budget (\( OET \)) are the most important objectives, TEMSA will recommend users to adapt MOCell, NSGA-II and SPEA2 for their test minimization, respectively (Activity A3, A4 and A5). Moreover, if a user wants to balance all the objectives together for test minimization, they do not need to select any objective, (Activity A1) and RWGA will be recommended by TEMSA automatically (Activity A6).

![Algorithm Recommendations by TEMSA](image)

**Fig. 7.** Recommendations for Algorithms by TEMSA
9. Threats to Validity

In this section, we discuss four threats to validity for our experiments and how we address them.

9.1. Internal Validity

Internal validity threats exist when the outcome of results are influenced by internal factors (e.g., parameters of search algorithms) and are not necessarily due to the application of the treatment (e.g., search algorithms) being studied [27, 32]. A possible threat to internal validity is that we have experimented with only one default configuration settings for the parameters of the selected search algorithms. However, these settings are in accordance with the common guidelines in the literature and our previous experience on testing problems [6, 28, 30].

9.2. Conclusion Validity

Conclusion validity threats are concerned with factors that can influence the conclusion that can be drawn from the results of the experiments. The most probable conclusion validity threat in experiments involving randomized algorithms is due to random variations. To address it, we repeated experiments 100 times to reduce the possibility the results were obtained by chance. Furthermore, to determine the probability of yielding higher performance by different algorithms, we measured the effect size using Vargha and Delaney statistics test since it is appropriate for non-parametric effect size measure, which matches our situation. Meanwhile, we performed Mann-Whitney U test to determine the statistical significance of the results [28]. Finally, Spearman’s rank correlation coefficient is used to measure the potential impact on the performance of selected search algorithms by increasing number of features and associated test cases since it is mainly used for non-parametric correlation coefficient measure and suits our objective in this paper [33].

9.3. Construct Validity

For the construct validity threats, we looked at the validity of the comparison measures for all the selected search algorithms. The most frequently observed threat was using some measures of cost that have severe limitations, as they are not precise. To reduce construct validity threats in our context, we used the same stopping criteria for all algorithms, i.e., number of fitness evaluations. We ran each algorithm for 25000 evaluations to seek the best solution for test minimization. This criterion is a comparable measure across all the
algorithms since each iteration requires updating the obtained solution and comparing the computed value of fitness function.

9.4. External Validity

External validity threats exist when the outcome of results are influenced by external factors and are not necessarily due to the application of the treatment being studied [27, 32]. One common external validity threat in the software engineering experiments is about generalization of results. One may argue the results cannot be generalized for other case studies if we ran our experiments on just one industrial case study. However, such threat to external validity is common to all empirical studies. In addition, we also conducted an empirical evaluation using carefully-designed 500 artificial problems and the results are consistent with the industrial case study.

10. Related Works

In the context of regression testing, the optimization problem can be classified into the following three problems, i.e., test case selection, minimization, and prioritization [7, 11, 14, 34, 35]. More specifically, test case selection focuses on selecting a subset of test cases from the existing test suite with aim to test a modified version of systems or programs while test minimization eliminates the redundant test cases from the existing test suite for the current systems or programs in order to reduce the cost of testing (e.g., time). Test prioritization orders the test cases in the existing test suite to test the current systems or programs with the objective of achieving the pre-defined criteria (e.g., maximum number of executing test cases) in a given time budget. The main difference between selection and minimization is that test minimization eliminates the redundant test cases permanently for the systems (programs), while test case selection refers to select relevant test cases temporarily for testing the modified version of the systems [7, 11]. However, from the perspective of optimization, there is no significant difference between test case selection and test minimization [7], i.e., both of them aim at choosing a subset of test cases from the existing test suite at the same time achieving pre-defined objectives for optimization, which was called “test case selection” problem in [7].

Our work is related to the above-mentioned selection problem, but as opposed to the existing works, our work lies in the context of product line testing (e.g., feature pairwise coverage is used to measure the effectiveness of the minimized test suite based on the
coverage of feature pairs from the product line view rather than code level (e.g., code coverage) in the most of the existing works for regression testing). Although there exists a large number of test selection/minimization techniques in the context of regression testing, there is not enough evidence that these techniques can be adapted to product lines (e.g., whether code coverage in regression testing is equivalent to the feature coverage in product line testing).

Yoo and Harman [14] used a greedy algorithm and a multi-objective search algorithm NSGA-II for test case selection for the following three measures: code coverage, fault detection history, and execution time in the context of regression testing. Notice that these three measures are similar to our cost/effectiveness measures $FPC$, $FDC$ and $OET$. The difference with our work is that first we select a set of relevant test cases for a new product using feature models as we discussed in [3, 4] and then we perform test minimization based on two additional measures that are $TMP$ and $AEF$. In addition, we conducted an empirical study to compare ten multi-objective algorithms belonging to four different mechanisms and evaluate their performance, which was not addressed in [14].

In another work related to test prioritization in regression testing [36], a two-objective problem (i.e., code coverage and execution time) is converted into a single-objective problem using an arithmetical combination of weights for the fitness function. These two objectives are close to our cost/effectiveness measures $FPC$ and $OET$. However, as compared with code coverage, our defined $FPC$ mainly focuses on the coverage of feature pairs from the view of product line rather than code level of the systems. Moreover, additional three measures are defined mathematically in our work and we do not limit our scope within weight-based GAs, i.e., different types of search algorithm are empirically evaluated, e.g., NSGA-II and SPEA2.

As compared with our previous work [6], we define two additional cost/effectiveness measures (i.e., $AEF$ and $OET$) for test minimization problem. Moreover, we further compare the performances for the best weight-based GA (i.e., RWGA) and the other six different search algorithms using an industrial case study and 500 designed artificial problems, which cover various kinds of search algorithms and make the results and analysis more persuasive.

To our knowledge and existing published reviews [11, 12], none of the existing works studied test minimization in the context of product line considering $TMP$, $FPC$, $FDC$, $OET$
and \textit{AEF} all together. In addition, it is one of the few works, which extensively compares the performance of search algorithms belonging to different mechanisms (e.g., evolutionary algorithms, swarm algorithms, and hybrid algorithms) for test suite minimization in the context of product lines.

11. Conclusion and Future Work

In this paper, we have proposed and have evaluated a fitness function in conjunction with ten multi-objective search algorithms, to minimize test suites in the context of product line testing. Our fitness function takes effectiveness measures (i.e., Test Minimization Percentage (\textit{TMP}), Feature Pairwise Coverage (\textit{FPC}), Fault Detection Capability (\textit{FDC}) and Average Execution Frequency (\textit{AEF})) and cost measures (i.e., Overall Execution Time (\textit{OET})) into account to guide the search. The search algorithms, along with our fitness function, are evaluated on an industrial case study and 500 artificial problems of varying size and complexity using a tool we have developed for this purpose, called TEst Minimization with Search Algorithms (TEMSA).

Given a set of thresholds for each objective, the results show that, unlike RS, WBGA-MO, RWGA, NSGA-II, MOCell, PAES, SMPSO and CellDE can reach these thresholds, while SPEA2 achieves acceptable performance for \textit{TMP/OET}, which is not the case for \textit{FPC/FDC}. Meanwhile, we observe that the performance of WBGA (i.e., WGBA\textsubscript{W1} and WGBA\textsubscript{W2}) is not stable since it largely depends on the pre-determined weights for the objectives. When comparing the selected algorithms, our results show that WBGA, WBGA-MO, RWGA, NSGA-II, MOCell, PAES, SMPSO and CellDE significantly outperform RS and SPEA2 has almost equivalent performance than RS. From the results on artificial problems, we conclude that RWGA also achieves the best performance when taking all objectives into account. Moreover, RWGA can solve a wider range of problems and its performance is improved when the number of features number is growing. Notice the performance of other weight-based GA is also significantly increased with the growth of features. For rest of the algorithms, the performance is not significantly influenced by the increasing number of features or test cases, when it comes to finding an optimal solution.

Our short-term future plan includes the analysis of another industrial case study in order to complement our experimental results and refine the design of TEMSA. We also plan to
refine the fitness function by considering other objectives, such as the number of available computing resources, and evaluate other major search algorithms, such as Archive-Based hYbrid Scatter Search (AbYSS), which combines both scatter search and genetic algorithms

**Acknowledgements.**

We gratefully thank Marius Liaaen and his team at Cisco for providing us the valuable industrial case study. All the authors acknowledge funding from the Research Council of Norway.

**References:**


175
Random-Weighted Search-Based Multi-Objective Optimization Revisited

Shuai Wang\textsuperscript{1,2}, Shaukat Ali\textsuperscript{1} and Arnaud Gotlieb\textsuperscript{1}

\textsuperscript{1}Certus Software V\&V Center, Simula Research Laboratory, Norway
\textsuperscript{2}Department of Informatics, University of Oslo, Norway
\{shuai, shaukat, arnaud\}@simula.no

Abstract. Weight-based multi-objective optimization requires assigning appropriate
weights using a weight strategy to each of the objectives such that an overall optimal
solution can be obtained with a search algorithm. Choosing weights using an appropriate
weight strategy has a huge impact on the obtained solutions and thus warrants the need to
seek the best weight strategy. In this paper, we propose a new weight strategy called
Uniformly Distributed Weights (UDW), which generates weights from uniform
distribution, while satisfying a set of user-defined constraints among various cost and
effectiveness measures. We compare UDW with two commonly used weight strategies,
i.e., Fixed Weights (FW) and Randomly-Assigned Weights (RAW), based on five
cost/effectiveness measures for an industrial problem of test minimization defined in the
context of Video Conferencing System Product Line developed by Cisco Systems. We
empirically evaluate the performance of UDW, FW, and RAW in conjunction with four
search algorithms ((1+1) Evolutionary Algorithm (EA), Genetic Algorithm, Alternating
Variable Method, and Random Search) using the industrial case study and 500 artificial
problems of varying complexity. Results show that UDW along with (1+1) EA achieves
the best performance among the other combinations of weight strategies and algorithms.

Keywords: Uniformly Distributed Weights; Multi-Objective Optimization, Search Algorithms

1. Introduction

A weight-based multi-objective optimization problem requires finding an optimal set of
weights for all the objectives, while satisfying the required constraints (e.g., the priority of
various objectives defined by users) on weights capturing the complex tradeoff
relationships among the objectives with the aim of finding an overall optimal solution for the problem. Such weight-based multi-objective problems have been solved efficiently together with search algorithms (e.g., Genetic Algorithms) in the literature [1-6]. As compared with other types of commonly used techniques (i.e., Pareto-based techniques) [1, 5], weight-based techniques offer the following advantages: 1) These techniques balance all the objectives to find an optimal solution rather than obtaining a set of non-dominated solutions thus eliminating the effort for users to select among the obtained solutions; 2) These techniques are straightforward to implement with computational efficiency; and 3) When objectives have different priorities, weight-based techniques can tackle such situation easily by assigning customized weights to each particular objective.

In the literature [1], the following two weight assignment strategies are commonly used: 1) Assigning fixed weights (equal or unequal) to each objective, termed as Fixed Weight (FW) strategy in this paper; 2) Assigning random weights to each objective based on a set of pre-defined constraints, coined as Randomly-Assigned Weights (RAW) strategy. Even though these two strategies have shown promising results [1-4, 6], they still suffer from some limitations. With FW, it is rare that all objectives have equal weights and determining appropriate weights usually depends on domain-expertise. A potential solution is to ask users to specify a set of constraints among weights rather than giving exact values. With RAW, we observe that it does not guarantee equally-distributed uniformity of selection of weight, i.e., all the potential weights do not have the same probability to be selected. Since one set of fixed weights can determine a specific search direction, there may be no equivalent probability to choose various search directions to find an optimal solution [1].

To reduce the randomness in weight selection using RAW, we propose a new weight assignment strategy called Uniformly Distributed Weights (UDW), which generates weights from a uniform distribution, while meeting a set of user-defined constraints. The strategy gives an equal importance to the generation of weight for each criterion, while preserving the relative importance of criterion. Said otherwise, the goal is to keep the advantages of uniform random generation, while having priorities defined among the various criteria. We evaluate the proposed weight strategy UDW as compared with the FW and RAW using an industrial problem of test minimization for Video Conferencing Systems (VCSs) product line from Cisco Systems. Being specific, this test minimization
problem is a multi-objective optimization problem having five distinct objectives: Test Minimization Percentage (TMP), Feature Pairwise Coverage (FPC), Fault Detection Capability (FDC), Average Execution Frequency (AEF), and Overall Execution time (OET). A fitness function defined on all these five objectives is used in conjunction with the following search algorithms: Genetic Algorithm (GA), (1+1) Evolutionary Algorithm (EA), and Alternating Variable Method (AVM) to compare the three distinct weight assignment strategies. Random Search (RS) is used as the comparison base line. Moreover, inspired by the industrial problem, we created 500 artificial problems of varying complexity to evaluate the three weight assignment strategies in conjunction with all the four algorithms.

The obtained results show that: 1) With FW, RAW and UDW, (1+1) EA significantly outperformed the other search algorithms; 2) With (1+1) EA, UDW significantly performed better than FW and RAW; 3) The performance of (1+1) EA and GA with UDW was significantly improved with the increasing complexity of problems.

The rest of the paper is organized as follows: Section 2 provides a relevant background on FW and RAW. Section 3 presents UDW strategy followed by the description of our industrial and artificial case studies (Section 4). Section 5 presents the empirical evaluation with an overall discussion and Section 6 addresses threats to validity. Related work is discussed in Section 7 and Section 8 concludes the paper.

2. Background

Fixed Weights (FW) assigns fixed normalized weights (between 0 and 1) to each objective [1]. These weights can be obtained from domain knowledge of experts. For instance, our case requires five weights \( w_1, w_2, w_3, w_4, w_5 \) corresponding to each objective \( (TMF, FPC, FDC, OET, AEF) \).

Randomly-Assigned Weights (RAW), inspired by Random-Weighted Genetic Algorithm (RWGA) mainly used for weight-based optimization [1], generates a set of distributed normalized weights for each objective while still satisfying the user defined constraints. For instance, we have a constraint \( w_2 > w_1 \) and using RAW, a random distributed value is first generated for \( w_2 \) from 0 to 1 and then another distributed value is generated for \( w_1 \) from 0 to \( w_2 \). At each generation when running search algorithms, each objective is assigned to a distributed normalized weight as above mentioned and meets all
the defined constraints, i.e., the weights for each objective are dynamically changed during each generation until the best solution is found or the termination criteria for the algorithms is met. Using RAW, multiple search directions can be stipulated by assigning dynamic weights during each generation [1]. However, RAW cannot guarantee that each point within the range for weights has the same probability to be selected thus each search directions cannot be reached uniformly.

3. Uniformly Distributed Weights
Inspired by our previous work in [7], the Uniformly Distributed Weights (UDW) uniformly selects weights at random when these weights are subject to a set of defined constraints. Notice that solving this problem efficiently is challenging as sampling uniformly tuples of values from an unknown domain is not trivial. More specifically, the tuples of weights generated by UDW must satisfy: 1) all the user-defined constraints and 2) an equi-probability selection that guarantee that each search direction has equivalent probability to be reached. Formally speaking, consider a multi-objective optimization problem \( P \) involving \( m \) optimization objectives \( O = \{o_1, o_2, ..., o_m\} \), each objective has a specific weight, corresponding to the set \( W = \{w_1, w_2, ..., w_m\} \). Relations among these objectives are captured by a set of \( n \) arithmetic constraints \( C = \{c_1, c_2, ..., c_n\} \) over the variables in \( W \). Notice that such constraints can be pre-defined by the users based on their particular domain expertise.

The core idea of UDW lies in the pre-computations of subdomains of the input domain (formed by the Cartesian product of each individual weight domain) and consideration of subdomains out of which uniform random sampling is trivial. More precisely, using an arbitrary division parameter \( k \), UDW first divides the input domain into \( k^m \) equivalent subdomains, where \( m \) is the number of weight variables. Second, a systematic refutation is used to eliminate subdomains that do not satisfy the constraints, and finally, uniform random generation of tuples of weights is realized by selecting a remaining subdomain and a tuple at random. Notice that the selected subdomain and tuple may still not satisfy the constraints, and thus rejection may still happen. But, the systematic refutation process would have eliminated most parts of the input domain that do not contain any solution (fully conflicts with the constraints).

The UDW algorithm (shown as below) takes as inputs, the set of weights \( W \), the constraint set \( C \), the division parameter \( k \), and the length of the expected sequence of
weight tuples (\(K\)), i.e., the number of generations during the search\(^1\). The output of \(UDW\) is a sequence of \(K\) weight tuples for \(W\), called \(WK\), such that the sequence is uniformly distributed over the solution set of \(C\). Notice that \(UDW\) returns fail (\(WK = \emptyset\)) when none of the weight tuples satisfy the constraint set \(C\), which shows none of the solutions will be found for the optimization problem based on the defined constraints. The Algorithm works as follows: a weight tuples sequence (\(WK\)) is first initialized as empty (\(Step 1\)) and then the input space is divided into \(k^m\) subdomains, i.e., \(\{WD_1, ..., WD_{k^m}\}\) (\(Step 2\)). The refutation process eliminates a number of subdomains with respect to the constraint set \(C\) (\(Step 3\)). The remaining subdomains (i.e., \(\{WD'_1, WD'_2, ..., WD'_p\}\)) are sampled uniformly and from them, uniform tuples of weights are generated at random until \(K\) sets are generated (\(Step 4\)). Notice that at this step, only the generated tuples that satisfy the constraint set \(C\) are kept. The generated set of tuples \(K\) is returned and used to assign weights to each objective, in order to guide the search in the search algorithms (\(Step 5\)). Notice that despite some similarities with the algorithm presented in [7, 8], our \(UDW\) algorithm is original in terms of uniformly generating random weights for multi-objective test suite optimization.

---

**Algorithm \(UDW\): Uniformly-Distributed Weight Strategy**

**Input:** \(W = \{w_1, w_2, ..., w_m\}\), \(C = \{c_1, c_2, ..., c_n\}\), division parameter \(k\) (Integer) and length of the expected weight tuples \(K\) (Integer)

**Output:** \(W_1, W_2, ..., W_K\) for Objectives \(O\) or \(\emptyset\)

**Step 1:** \(WK := \emptyset\)

**Step 2:** \((WD_1, ..., WD_{k^m}) := \text{Divide}([w_1, w_2, ..., w_m], k)\);

**Step 3:** for all \(WD_i \in (WD_1, ..., WD_{k^m})\) do

if \(WD_i\) is fully unsatisfiable w.r.t. \(C\) then remove \(WD_i\) from \((WD_1, ..., WD_{k^m})\);

**Step 4:** Suppose \(\{WD'_1, WD'_2, ..., WD'_p\}\) is the remaining list of domains;

if \(p \geq 1\) then

while \(K > 0\) do

choose \(WD'_j\) uniformly and randomly from \(\{WD'_1, WD'_2, ..., WD'_p\}\);

choose \(W\) uniformly and randomly from \(WD'_j\);

if \(W\) satisfies \(C\) then add \(W\) to \(WK\); \(K := K - 1\);

**Step 5:** return \(WK\) for Objectives \(O\).

---

### 4. Case Study

\(^1\) Each generation requires assigning a new set of weights to each objective during the search.
**Industrial Case Study.** Our industrial partner is a product line of Video Conferencing Systems (VCSs) called *Saturn* developed by Cisco Norway [8]. *Saturn* has several products, e.g., C20 (low end product) and C90 (high end product). Test suite minimization for testing a product is essential since it is practically impossible to execute all the test cases developed for the whole product line within the allocated budget [9, 10]. However, such minimization may have descendent impact on the effectiveness of testing (e.g., fault detection capability) when reducing the number of test cases. Thus, this minimization problem can be formulated as a multi-objective optimization problem that is well solved by various search algorithms [1, 5], i.e., we aim at reducing the cost of testing while preserving the effectiveness. We chose this problem to evaluate our proposed weight assignment strategy and such problem can be represented formally as: search for a solution $s_k$ (a subset of the test cases) from the solution space $S$ (all combinations of the test cases for testing a given product) to achieve the following objectives (i.e., maximum effectiveness and minimum cost):

$$\forall s_l \in S \cap s_l \neq s_k:$$

$$\textit{Effectiveness}(s_k) \geq \textit{Effectiveness}(s_l) \text{ and } \textit{Cost}(s_k) \leq \textit{Cost}(s_l)$$

Moreover, three effectiveness measures were previously defined in [6]: *TMP* was used to measure the amount of reduction for the number of test cases as compared with the original test suite; *FPC* and *FDC* were defined to calculate the feature pairwise coverage (each feature represents a testing functionality for VCS testing) and fault detection capability achieved by a potential solution. More detailed definitions and mathematical formulas for these effectiveness measures can be consulted in [6]. Through more investigation with Cisco, an additional effectiveness measure called Average Execution Frequency (*AEF*) was defined to count the average execution frequency for a solution thereby measuring its priority and a cost measure called Overall Execution Time (*OET*) was defined to measure the execution time cost for the potential solution obtained by search algorithms. Moreover, a fitness function based on the cost/effectiveness measures was defined to guide the search. This fitness function converted multi-objective minimization problem into single objective problem based on the weight-based theory [1], which is shown as follows:

$$\textit{Fit}_F = 1 - w_3 \ast \text{nor} (\text{TMP}) - w_2 \ast \text{nor} (\text{FPC}) - w_3 \ast \text{nor} (\text{FDC}) - w_4 \ast (1 - \text{nor}(\text{OET})) - w_5 \ast \text{nor} (\text{AEF})$$
Notice \( \text{nor} (x) \) is a normalization function, which is computed as follows: \( x/(x + 1) \). A lower value of fitness function \( \text{Fit}_F \) (we called such value as Overall Fitness Value (OFV)) represents a better solution. Moreover, \( w_1, w_2, w_3, w_4 \) and \( w_5 \) are a set of weights assigned to \( \text{TMP, FPC, FDC, OET and AEF} \) respectively and are required to satisfying a basic constraint \( \sum_{i=1}^{5} w_i = 1 \) in addition to other constraints capturing tradeoff relationships among the objectives.

We chose four products C20, C40, C60 and C90 from \textit{Saturn} that includes 169 features and each product can be represented by a subset of these features. Each feature can be tested by at least one test case. More specifically, C20, C40, C60 and C90 includes 17, 25, 32 and 43 features respectively and 138, 167, 192 and 230 test cases relevant for testing these products, respectively. Each test case \( tc_i \) has a success rate for execution (\( \text{SucR}_{tc_i} \)) for calculating \( \text{FDC} \), an average execution time (\( \text{AET}_{tc_i} \)) for measuring \( \text{OET} \) (\( \text{OET} = \sum_t \text{AET}_{tc_i} \)) and an execution frequency (\( \text{EF}_{tc_i} \)) for obtaining \( \text{AEF} \) (\( \text{AEF} = \sum_t \text{EF}_{tc_i} / n_s \), \( n_s \) is the number of test cases included in a specific solution). In summary, for \textit{Saturn}, each feature is associated with 5-10 test cases and each test case is associated with 1-5 features with \( \text{SucR}_{tc_i} \) ranging from 50\% to 95\%, \( \text{AET}_{tc_i} \) ranging from 2 minutes to 60 minutes and \( \text{EF}_{tc_i} \) ranging from 1 time to 50 times per week.

\textbf{Artificial Problems.} We further defined 500 artificial problems to evaluate the performance of \( FW, RAW \) and \( UDW \). Notice that the artificial problems are inspired by our industrial case study but with expansion for generality. To achieve this, we first created a feature model containing 1200 features and a test case repository of 60,000 test cases. For each test case \( tc_i \), three key attributes are assigned randomly, i.e., \( \text{SucR}_{tc_i} \) ranging from 0\% to 100\%, \( \text{AET}_{tc_i} \) ranges from 1 minutes to 100 minutes and \( \text{EF}_{tc_i} \) ranges from 1 time to 100 times per week. Moreover, we created artificial problems with the increasing number of features and associated test cases for each feature, i.e., we used a range of 10 to 1000 with an increment of 10 for features number and each feature can be associated with test cases ranging from 5 to 45 with an increment of 10. Thus, \( 100*5 = 500 \) artificial problems were obtained in this way. Notice that such artificial problems are also designed based on the domain expertise of VCS testing and thorough discussions with test engineers at Cisco.

\textbf{5. Empirical Evaluation}
5.1. Experiment Design

Recall that our main goal is to minimize the test suite for testing a product with high effectiveness (\(TMP, FPC, FDC\) and \(AEF\)) while meeting the time budget (\(OET\)).

To generate suitable weights, it is of paramount importance to provide a set of pre-defined constraints in a given context. In our industrial case study, through the domain analysis and several thorough discussion with test engineers at Cisco, we observed that: 1) \(OET\) has the highest priority among all the objectives; 2) \(FPC\) and \(FDC\) are more important than \(TMP\) and \(AEF\); and 3) \(AEF\) has the lowest priority among all the objectives. So we came up with five key independent constraints for the test minimization problem, i.e., \(w_4 > w_2\), \(w_4 > w_3\), \(w_2 > w_1\), \(w_3 > w_1\) and \(w_1 > w_5\). Based on the defined constraints, specific weights for each objective can be generated using different weight strategies for the industrial case study and artificial problems.

Using the experiments, we want to address the following research questions:

**RQ1:** With each weight strategy (i.e., \(UDW\), \(FW\) and \(RAW\)), which search algorithm achieves the best performance for each objective and \(OFV\)?

**RQ2:** With the best search algorithm, which weight strategy can achieve the best performance for each objective and \(OFV\)?

**RQ3:** How does the increment of the number of features and associated test cases influence the performance of the search algorithms along with each weight strategy?

Being specific, our experiment first compares each pair of the four algorithms with each weight strategy to determine which algorithm can achieve the best performance (RQ1). Afterwards, we choose the best algorithm and compare its performance with each weight strategy to evaluate whether \(UDW\) can outperform the other two weight strategies (RQ2), which also shows which combination of search algorithms and weight strategies can achieve the best performance. Notice that for industrial case study, we evaluate the values for each objective (i.e., \(TMP, FPC, FDC, OET\) and \(AEF\)) and as for artificial problems, we only evaluate the values of \(OFV\) to assess the performance and scalability for the selected algorithms with different weight strategies. To address RQ3, Mean Fitness Value for each problem \(MFV_{i,j} = \frac{\sum_{r=1}^{nr} OFV_{r}}{nr}\) is defined to measure the mean overall fitness value for a certain number of runs \(nr\) (in our case, \(nr = 100\)), where \(i\) is the feature number and \(j\) is the test case number \((10 \leq i \leq 1000\) with an increment of \(10\) and \(5 \leq j \leq 45\) with an
increment of 10). \( OFV_r \) is the obtained fitness value after the \( r_{th} \) run.

Moreover, Mean Fitness Value for Feature \((MFV_F_i = \frac{\sum_{j=5}^{45} MFV_{ij}}{5})\) and Mean Fitness Value for Test Case \((MFV_{TC}_j = \frac{\sum_{i=10}^{1000} MFV_{ij}}{100})\) are further defined to measure the mean fitness function in a given number of features or associated test cases. In this way, a specific \( MFV_F \) and \( MFV_{TC} \) can be calculated for each number of features and test cases and RQ3 can be addressed using statistical analysis.

In addition, for all the search algorithms, the maximum number of evaluation for the fitness function is set as 5000 and we collected the optimal solution after the 5000\( _{th} \) fitness function evaluation. For GA and (1+1) EA, the mutation of a variable is done with the standard probability \( 1/n \), where \( n \) is the number of variables. We used a standard one-point crossover with a rate of 0.75 for GA and set the size of population as 100. RS was used as the comparison baseline to assess the difficulty of the problems [11]. According to the guidelines in [11, 13], each algorithm is run for 100 times to account for random variation inherited in search algorithms.

### 5.2. Statistical Tests

To analyze the obtained result, the Vargha and Delaney statistics and Mann-Whitney U test are used based on the guidelines for reporting statistical tests for randomized algorithms [11]. The Vargha and Delaney statistics is used to calculate \( \hat{A}_{12} \), which is a non-parametric effect size measure. In our context, \( \hat{A}_{12} \) is used to compare the probability of yielding higher values for each objective function and overall fitness value \((OFV)\) for two algorithms \( A \) and \( B \) with different weight strategies. If \( \hat{A}_{12} \) is 0.5, the two algorithms have equal performance. If \( \hat{A}_{12} \) is greater than 0.5, \( A \) has higher chances to obtain better solutions than \( B \). The Mann-Whitney U test is used to calculate \( p \)-value for deciding whether there is a significant difference between \( A \) and \( B \). We choose the significance level of 0.05, i.e., there is a significant difference if \( p \)-value is less than 0.05. Based on the above description, we define that algorithm \( A \) has better performance than algorithm \( B \), if the value of \( \hat{A}_{12} \) is greater than 0.5 and such better performance is significant if \( p \)-value is less than 0.05.

To address RQ3, we choose the Spearman’s rank correlation coefficient \( (\rho)\) to measure the relations between the \( MFV_F \) and \( MFV_{TC} \) obtained by the algorithms with different
number of features and test cases [12]. More specially, there is a positive correlation if $\rho$ is greater than 0 and a negative correlation when $\rho$ is less than 0. A $\rho$ close to 0 shows that there is no correlation between the two sets of data. Moreover, we also report significance of correlation using $\text{Prob} > |\rho|$, a value lower than 0.05 means that the correlation is statistically significant.

5.3. Results and Analysis

Results and Analysis for Industrial Case Study. Notice that due to limited space, all detailed results can be consulted in the technical report [18] (i.e., Table 4 and Table 5). We only present the key findings for each research question (RQ1-RQ2).

Results and Analysis for RQ1. Based on the obtained results (Table 4 in [18]), we concluded that all the three search algorithms (i.e., AVM, GA and (1+1) EA) significantly outperformed RS for each objective and $OFV$ with each weight strategy (i.e., $FW$, $RAW$ and $UDW$). Moreover, (1+1) EA achieved significantly better performance than GA and AVM for each objective and $OFV$ (RQ1). In addition, AVM had significantly worse performance than GA for each objective and $OFV$. In summary, for each objective and $OFV$ for the four Saturn products, the performance of the four algorithms can be sorted as (1+1) EA, GA, AVM and RS, from better to worst.

Results and Analysis for RQ2. Based on the results of RQ1, we chose the best algorithm (1+1) EA and compared its performance in conjunction with $FW$, $RAW$ and $UDW$ for each objective and $OFV$ in the four Saturn products (Table 5 in [18]). According to the results, we concluded that (1+1) EA along with $RAW$ and $UDW$ significantly outperformed (1+1) EA with $FW$ for each objective and $OFV$. Moreover, the performance of (1+1) EA with $UDW$ was also significantly better than (1+1) EA with $RAW$ for $TMP$, $FPC$, $FDC$, $OET$, $AEF$ and $OFV$. Meanwhile, we report the average time used by each algorithm per run, i.e., 2.17 seconds for AVM, 1.78 seconds for GA, 1.55 seconds for (1+1) EA and 1.20 seconds for RS, which shows running search algorithms require similar effort (i.e., time) as compared with RS. In summary, we concluded (1+1) EA along with $UDW$ achieved the best performance.
Results and Analysis for Artificial Problems. Recall that the evaluation for artificial problems is based on the overall fitness value (OFV) for each of the 500 problems. To answer RQ3, we calculated the spearman’s rank correlation using mean fitness values $MFV_F$ and $MFV_TC$ (Section 5.1) for each algorithm with different weight strategies. The detailed results and analysis are discussed in detail as below.

Results and Analysis for RQ1. Table 1 summarizes the results for comparing the selected search algorithms with $FW$, $RAW$ and $UDW$ for the 500 artificial problems. Two numbers are shown in each cell of the table split by a slash. The first number in the column $A>B$ shows the number of problems out of 500 for which an algorithm $A$ has better performance than $B$ ($\hat{A}_{12} > 0.5$), $A<B$ means vice versa ($\hat{A}_{12} < 0.5$), and $A=B$ means the number of problems for which $A$ has equivalent performance as $B$ ($\hat{A}_{12} = 0.5$). The second number after “/” in the column $A>B$ means the number of problems out of 500 for which $A$ has significantly better performance than $B$ ($\hat{A}_{12} > 0.5$ && $p < 0.05$), $A<B$ means vice versa ($\hat{A}_{12} < 0.5$ && $p < 0.05$), and $A=B$ means the number of problems for which there is no significant difference in performance between $A$ and $B$ ($p \geq 0.05$). We concluded the results as below.

$AVM$ vs. $GA$: $AVM$ outperformed $GA$ for on average 40.8% problems (i.e., $(167+231+214)/3/500*100\% = 40.8\%$), but for 34.94% problems, $AVM$ performed significantly better than $GA$. $AVM$ performed worse than $GA$ for on average 59.2% problems and in 52.4% problems, $AVM$ was significantly worse than $GA$. There were no significant differences between $AVM$ and $GA$ for 12.66% problems.

$AVM$ vs. $(1+1)$ EA: $AVM$ performed better than $(1+1)$ EA for on average 22.92% problems and 17.86% out of these 22.92% problems, $AVM$ significantly outperformed $(1+1)$ EA. On the contrary, for on average 77.06% problems, $AVM$ performed worse than $(1+1)$ EA and for 72.6% problems, $AVM$ performed significantly worse than $(1+1)$ EA. For 9.54% problems on average, there were no significant differences between $AVM$ and $(1+1)$ EA.

$AVM$ vs. $RS$: $AVM$ outperformed $RS$ for 68.34% problems with the three weight strategies on average but for 60.14% problems the results were statistically significant. There were no significant differences for 15.54% problems on average.

$GA$ vs. $(1+1)$ EA: For 33.4% problems on average, $GA$ was better than $(1+1)$ EA, but
for 29.2% problems, GA was significantly better than (1+1) EA. Moreover, GA was worse than (1+1) EA for 66.6% problems and for 61.74% out of these 66.6% problems, GA was significantly worse than (1+1) EA. There were no significant differences between GA and (1+1) EA for 9.74% problems.

### Table 1. Results for comparing the algorithms with each weight strategy for artificial problems

<table>
<thead>
<tr>
<th>Weight Strategy</th>
<th>Pair of Algorithms</th>
<th>A&gt;B</th>
<th>A&lt;B</th>
<th>A=B</th>
<th>Best Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FW</strong></td>
<td>AVM vs. GA</td>
<td>167/132</td>
<td>333/297</td>
<td>0/71</td>
<td>(1+1) EA</td>
</tr>
<tr>
<td></td>
<td>AVM vs. (1+1) EA</td>
<td>121/98</td>
<td>379/344</td>
<td>0/58</td>
<td></td>
</tr>
<tr>
<td></td>
<td>AVM vs. RS</td>
<td>377/345</td>
<td>123/101</td>
<td>0/54</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GA vs. (1+1) EA</td>
<td>134/115</td>
<td>366/348</td>
<td>0/37</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GA vs. RS</td>
<td>364/331</td>
<td>136/112</td>
<td>0/57</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(1+1) EA vs. RS</td>
<td>436/408</td>
<td>64/37</td>
<td>0/55</td>
<td></td>
</tr>
<tr>
<td><strong>RAW</strong></td>
<td>AVM vs. GA</td>
<td>231/199</td>
<td>269/238</td>
<td>0/63</td>
<td>(1+1) EA</td>
</tr>
<tr>
<td></td>
<td>AVM vs. (1+1) EA</td>
<td>107/76</td>
<td>393/376</td>
<td>0/48</td>
<td></td>
</tr>
<tr>
<td></td>
<td>AVM vs. RS</td>
<td>312/267</td>
<td>188/135</td>
<td>0/98</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GA vs. (1+1) EA</td>
<td>195/164</td>
<td>305/276</td>
<td>0/60</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GA vs. RS</td>
<td>397/340</td>
<td>103/63</td>
<td>0/97</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(1+1) EA vs. RS</td>
<td>449/423</td>
<td>51/38</td>
<td>0/39</td>
<td></td>
</tr>
<tr>
<td><strong>UDW</strong></td>
<td>AVM vs. GA</td>
<td>214/193</td>
<td>286/251</td>
<td>0/56</td>
<td>(1+1) EA</td>
</tr>
<tr>
<td></td>
<td>AVM vs. (1+1) EA</td>
<td>116/94</td>
<td>384/369</td>
<td>0/37</td>
<td></td>
</tr>
<tr>
<td></td>
<td>AVM vs. RS</td>
<td>336/290</td>
<td>164/129</td>
<td>0/81</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GA vs. (1+1) EA</td>
<td>172/149</td>
<td>328/302</td>
<td>0/49</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GA vs. RS</td>
<td>369/323</td>
<td>131/105</td>
<td>0/72</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(1+1) EA vs. RS</td>
<td>427/399</td>
<td>73/54</td>
<td>0/47</td>
<td></td>
</tr>
</tbody>
</table>

**GA vs. RS:** There were on average 75.34% problems, for which GA achieved better performance than RS with the selected weight strategies and for average 66.26% problems, GA significantly outperformed RS. Moreover, for 15.06% problems, there were no significant differences between GA and RS.

**(1+1) EA vs. RS:** In case of (1+1) EA, it performed better than RS for 87.46% problems. Out of these 87.46% problems, on average 82% problems, (1+1) EA was significantly better than RS. For 9.4% problems, there were no significant differences between (1+1) EA and RS.

**Concluding Remarks for RQ1:** Based on the results, RQ1 can be answered as follows: the performance of AVM, GA and (1+1) EA are all significantly better than RS with all the selected three weight strategies in term of finding an optimal solution for our minimization problem. Moreover, (1+1) EA outperformed GA and GA performed better than AVM together with all the weight strategies. In summary, (1+1) EA achieved the best performance for the three weight strategies in most of the problems.

**Results and Analysis for RQ2.** Table 2 summarizes the results for comparing **FW**, **RAW** and
UDW with the best algorithm (1+1) EA for the 500 artificial problems. The data in the columns \(A>B\), \(A<B\) and \(A=B\) is organized in the same way as in Table 1.

### Table 2. Results for comparing the weight strategies along with (1+1) EA for artificial problems

<table>
<thead>
<tr>
<th>Pair of Weight</th>
<th>A&gt;B</th>
<th>A&lt;B</th>
<th>A=B</th>
</tr>
</thead>
<tbody>
<tr>
<td>FW vs. RAW</td>
<td>187/165</td>
<td>313/276</td>
<td>0/59</td>
</tr>
<tr>
<td>FW vs. UDW</td>
<td>119/91</td>
<td>381/364</td>
<td>0/45</td>
</tr>
<tr>
<td>RAW vs. UDW</td>
<td>197/175</td>
<td>303/276</td>
<td>0/49</td>
</tr>
</tbody>
</table>

**FW vs. RAW:** In conjunction with (1+1) EA, for 37.4% (187/500=37.4%) problems, FW outperformed RAW, but for 33% of problems, there were significant differences. On the contrary, RAW performed better than FW for 62.6% problems and for 55.2% problems, RAW was significantly better than FW. There were no significant differences between FW and RAW for 11.8% problems.

**FW vs. UDW:** For (1+1) EA, FW performed better than UDW for 23.8% problems, but only for 18.2% problems, FW significantly outperformed UDW. Moreover, the performance of FW was worse than UDW for 76.2% problems and for 72.8% problems, FW was significantly worse than UDW. Meanwhile, there were no significant differences between FW and UDW for 9% problems.

**RAW vs. UDW:** Combined with (1+1) EA, for 39.4% problems, RAW performed better than UDW and there were significant differences for 35.2% out of these 39.4% problems. UDW outperformed RAW for 60.6% problems and UDW significantly outperformed RAW for 55.2% problems. In addition, there were no significant differences between RAW and UDW for 9.8% problems.

Similarly, the average time taken by each algorithm is reported per run for the 500 artificial problems, i.e., 4.58 seconds for AVM, 3.96 seconds for GA, 3.41 seconds for (1+1) EA and 3.04 seconds for RS, which shows adapting search algorithms takes similar time as compared with RS.

**Concluding Remarks for RQ3:** Based on the above results, we can answer RQ3 as follows: along with the best algorithm (1+1) EA, UDW achieved the best performance among the three weight strategies and RAW outperformed FW significantly in most of the artificial problems, i.e., the UDW weight strategy in conjunction with (1+1) EA achieved the best performance in our context.

**Results and Analysis for RQ3.** Table 3 provides the results for Spearman’s correlation
analysis ($\rho$) between mean fitness value for feature ($MFV_F$) with the increasing number of features and mean fitness value for test case ($MFV_TC$) with the growth of test cases for the 500 artificial problems. Recall that a lower value of $MFV_F$ or $MFV_TC$ represents a better performance of an algorithm with a weight strategy.

**Increasing Number of Features:** For $FW$, $RAW$ and $UDW$, we observed that the $MFV_F$ values obtained by (1+1) EA and GA decreased significantly with the growth of feature number since all the $\rho$ values were less than 0 and the values of $Prob > |\rho|$ were all less than 0.0001 (Table 3), i.e., the performance of (1+1) EA and GA significantly improved as the increasing number of features. For AVM, the $MFV_F$ values also decreased when the number of features increases but such decrease was not statistically significant, i.e., AVM also performed better as the increasing number of features but not significantly. Finally, the performance of RS was worse with the growth of the number of features (not significantly) (Table 3).

<table>
<thead>
<tr>
<th>Weight Strategy</th>
<th>Algorithms</th>
<th>Increasing Features</th>
<th>Increasing Test Cases</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Spearman $\rho$</td>
<td>$Prob&gt;</td>
</tr>
<tr>
<td>$FW$</td>
<td>AVM</td>
<td>-0.07</td>
<td>0.4513</td>
</tr>
<tr>
<td></td>
<td>(1+1) EA</td>
<td>-0.68</td>
<td>&lt;0.0001</td>
</tr>
<tr>
<td></td>
<td>GA</td>
<td>-0.65</td>
<td>&lt;0.0001</td>
</tr>
<tr>
<td></td>
<td>RS</td>
<td>0.12</td>
<td>0.4870</td>
</tr>
<tr>
<td>$RAW$</td>
<td>AVM</td>
<td>-0.14</td>
<td>0.3764</td>
</tr>
<tr>
<td></td>
<td>(1+1) EA</td>
<td>-0.71</td>
<td>&lt;0.0001</td>
</tr>
<tr>
<td></td>
<td>GA</td>
<td>-0.64</td>
<td>&lt;0.0001</td>
</tr>
<tr>
<td></td>
<td>RS</td>
<td>0.16</td>
<td>0.5137</td>
</tr>
<tr>
<td>$UDW$</td>
<td>AVM</td>
<td>-0.09</td>
<td>0.6125</td>
</tr>
<tr>
<td></td>
<td>(1+1) EA</td>
<td>-0.70</td>
<td>&lt;0.0001</td>
</tr>
<tr>
<td></td>
<td>GA</td>
<td>-0.69</td>
<td>&lt;0.0001</td>
</tr>
<tr>
<td></td>
<td>RS</td>
<td>0.09</td>
<td>0.4626</td>
</tr>
</tbody>
</table>

**Increasing Number of Associated Test Cases** (*Increasing Test Cases* column): The $MFV_TC$ values by AVM, (1+1) EA and GA decreased but not significantly with the growth of associated test cases, i.e., the performance of AVM, (1+1) EA and GA in along with all the weight strategies improved (but not significantly) with the growth of associated test cases. On the contrary, RS performed worse (not significantly) when the number of associated test cases increased (Table 3).

**Concluding Remarks:** Among all the weight strategies, (1+1) EA and GA performed significantly better as the increasing number of features and the performance of AVM and RS were not significantly influenced by the number of features. Moreover, the performance of all the selected search algorithms with the weight strategies are not
significantly influenced by the increasing the number of associated test cases.

5.4. Overall Discussion

First, based on the results of RQ1 and RQ2, the reason why UDW performs better than FW can be explained as follows: 1) FW uses fixed predefined weights, meaning that the search space, which is the Cartesian product of all weights, cannot be fully explored. Actually, the optimal solution might not be found in this restricted search space explored in a single direction; and 2) UDW uniformly assigns random weights generated during each generation of a search algorithm, which allows the search algorithm to explore multiple directions.

Moreover, with uniformly distributed weights (UDW) at each generation, a search algorithm can be guided towards an optimal solution more efficiently than randomly generated weights (RAW). This may be because that RAW does not permit us to search uniformly at each generation, as the weights cannot be selected with the same probability, due to the presence of constraints. Consider two weights $w_1$ and $w_2$ from 0 to 1; for the sake of simplicity, but without losing any generality, we suppose that $w_1, w_2 \in \{0, \frac{1}{N}, \frac{2}{N}, \ldots, \frac{N-1}{N}, 1\}$ and the following constraint holds: $w_1 > w_2$. Suppose that we have two distinct candidates for $w_1$ namely $w_1^0 = \frac{N_1}{N}, w_1^1 = \frac{N_2}{N}$, where $N_1, N_2 \in \{0, 1, \ldots, N\}$ and one candidate for $w_2$ called $w_2^0$ (the solution space of the constraint for $w_1$ and $w_2$ is represented as Fig. 1).

Using RAW, if $w_1^0$ is selected first, the probability of selecting $w_2^0$ is $\text{Prob}(\{w_2^0 \mid w_1^0\}) = \frac{1}{N} \cdot \frac{1}{N_1}$, as the constraint $w_1 > w_2$ holds. If $w_1^1$ is selected, the probability of selecting $w_2^0$ is $\text{Prob}(\{w_2^0 \mid w_1^1\}) = \frac{1}{N} \cdot \frac{1}{N_2}$. As a consequence, the probability of selecting $w_2^0$ is not the same in both cases ($\frac{\text{Prob}(\{w_2^0 \mid w_1^0\})}{\text{Prob}(\{w_2^0 \mid w_1^1\})} = \frac{N_2}{N_1}$), meaning that the couples ($w_1^0, w_2^0$) and ($w_1^1, w_2^0$) do not have the same probability to be selected. Hence, there is no equi-probability when it comes to the selection of relevant search directions to find an optimal solution. Unlike RAW, UDW guarantees the equi-probability selection of $w_2^0$ whatever be the first weight value selected (i.e., $\text{Prob}(\{w_2^0 \mid w_1^0\}) = \text{Prob}(\{w_2^0 \mid w_1^1\}) = \frac{1}{N} \cdot \frac{1}{N} = \frac{2}{N \cdot N}$). Thus, each search
direction has an equal probability to be reached and thereby, guiding search to find an optimal solution is more equally supported.

Followed by (1+1) EA, GA has significantly better performance than AVM. This can be explained from the fact that both of these algorithms are global search algorithms and thus managed to find global optimal solutions as compared to AVM, which is a local search algorithm. Notice that the performance of (1+1) EA is significantly better than GA and this may be due to the reason that (1+1) EA uses only mutation for exploring the search space as compared to GA which uses both mutation and crossover for exploration and exploitation of search space respectively requiring more generations to find a global optimal solution. By increasing the number of generations (5000 in the current experiment settings), we expect that the performance of GA can be improved, which requires further empirical evaluations.

For the experiments based on the 500 artificial problems, the results were consistent with our industrial case study (RQ1-RQ2). When we looked at the impact of varying number of features (10-1000) and associated test cases (5-45) on the performance of each algorithm along with three weight strategies (RQ3), we observed that (1+1) EA and GA performed significantly better as the increasing number of features, but for AVM and RS, the performance was not significantly influenced with the growth of features. Such interesting behavior can be explained based on the fact that (1+1) EA and GA are global search algorithms, which can still manage to explore the search space and find a global optimal solution even with the increased complexity (more features). On the other hand, AVM’s performance (local search) cannot scale with the increased complexity since AVM can be guided towards finding local solutions in the search space but the global optimal solutions might be missed. Moreover, we observed that increasing the number of test cases improved the performance of all search algorithms except RS though not significantly. This phenomenon may be due to the following two reasons: 1) Complexity of test minimization problem for a product is directly related to the number of features and thus increasing the test cases may not affect the performance of the search algorithms; 2) Increased number of test cases means that a feature can be tested with more test cases and thus increases the solution space within the entire search space, i.e., search algorithms can find solutions with better fitness since more solutions are available (though not statistically significant).
In summary, (1+1) EA along with UDW weight strategy achieved the best performance in our experiments and thus is suitable for test minimization problem in our industrial context. Moreover, the results suggest weights selected by domain experts might not be accurate to obtain an optimal test minimization solution in practice and thus automated weight strategies such as UDW are needed for an optimal solution.

6. Threats to Validity

A prominent construct validity threat is related to the measure used to compare the various algorithms and to avoid such threat we used the measured fitness value, which is comparable across all selected search algorithms. Another common construct threat to validity is the use of termination criterion for the search. In our experiments, we used number of fitness evaluations comparable across all the algorithms.

When using search algorithms, parameter settings may affect the performance of the algorithms (internal validity). In this direction, we used default parameter settings for all the algorithms and these settings have demonstrated promising results [13]. In addition, the complexity of UDW in the general case is exponential in the number of weight variables in the problem (i.e., \( O(k^m) \) where \( k \) is an arbitrary division parameter and \( m \) is the number of weight variables). This is a limitation if one wants to consider optimization problem with more than 20 independent objectives \((m > 20)\) but our experience with both academic and industrial case studies show that the number of considered objectives never goes up to seven. Consequently, this potential exponential blow-up is not considered as a threat to our approach in practice.

A common conclusion validity threat is due to random variation inherited in search algorithms and thus we repeated our experiments 100 times to reduce the probability that the results were obtained by chance. Moreover, we used appropriate statistical tests for analyzing the data, i.e., Vargha and Delaney, Mann-Whitney U test and Spearman’s rank correlation coefficient based on the guideline proposed in [11].

Generalization to new case studies is required to increase the confidence on the results (external validity) and we conducted an empirical evaluation using 500 artificial problems besides an industrial case study and obtained consistent results.

7. Related Works

A comprehensive review for search-based software engineering (SBSE) is available in
In particular, Harman listed a set of potential objectives used for multi-objective test optimization in regression testing [16], which have been extensively studied in the existing literature [14]. In [15], a two-objective problem (i.e., code coverage and execution time) is converted into a single-objective problem for test prioritization using an arithmetical combination of weights for the fitness function. However, there are at least two main differences with our work: 1) The UDW approach is not restricted to two criteria and can actually takes any number of test objectives into consideration (e.g., TMP, FDC and AEF) and 2) UDW is parameterized by a set of constraints for which it can provide a uniform sampling of weight values, which turns out to be essential when looking at the performance of various search algorithms (e.g., (1+1) EA). As compared with our previous work [6], the motivation in this paper is different. The determination of an appropriate weight assignment strategy using weight-based search algorithms turns out to be crucial to obtain an optimal solution, especially when determining the best possible weights is impossible [1].

In addition, a simple algorithm can be used alternatively to sample values uniformly at random in the presence of constraints. For example, [17] reported on such an algorithm: 1) firstly, it generates tuples of values randomly while ignoring the constraints; 2) secondly, it uses a linear constraint solver (e.g., the Simplex algorithm) to reject the generated tuples that do not satisfy the pre-defined constraints. Even if this approach is appealing by its simplicity, it does not scale up to large dimensions as shown in [7]. In fact, as soon as the constraints become complex (relational, non-linear, mixed integer-real), the number of rejected tuples grows up to a point where the number of calls to the constraint solver is intractable. Note also that using constraint propagation and refutation instead of the Simplex algorithm opens the door to the treatment of non-linear constraints (e.g., $w_1 * w_2 < 1$) but it is also incomplete to determine the exact shape of the solution space.

Moreover, the proposed UDW technique is inspired from the Path-Oriented Random Testing (PRT) approach used in the context of code-based testing [7]. Even if the algorithm used to randomly generate uniformly distributed samples is similar to the one used in PRT, we see a main difference, i.e., according to our knowledge, using uniformly random distributed weights when constraints among weights are involved in search-based test minimization has never been explored before. Expressing constraints over weights is a key aspect of the proposed UDW technique as it releases test engineers from the tedious
task of determining exact values to the weights, while preserving the benefits of RAW-approaches of search-based test minimization.

8. Conclusion and Future Work

In this paper, we proposed a new weight assignment strategy called Uniformly Distributed Weights (UDW) to generate weights by solving constraints among them with uniform distribution for solving multi-objective optimization problems. UDW can guarantee the uniformity for the selection of weights at the same time meeting all the required constraints based on the domain knowledge and expertise. We compared the proposed UDW with two commonly-used weight strategies (i.e., Fixed Weights (FW) and Randomly-Assigned Weights (RAW)) in conjunction with the following search algorithms: (1+1) Evolutionary Algorithm (EA), Genetic Algorithm, Alternating Variable Method, and Random Search based on our industrial problem of test minimization in the context of product lines. For test minimization, a fitness function was defined based on various cost/effectiveness measures (e.g., feature pairwise coverage) identified through our industrial collaboration with Cisco Systems. We performed our empirical evaluation using a Video Conferencing Systems product line provided by Cisco Systems and 500 artificial problems of varying complexity. The results showed that (1+1) EA performs the best among all the algorithms together with UDW and thus we conclude that assigning weights based on uniform distribution can significantly improve the performance of (1+1) EA for multi-objective optimization, particularly for multi-objective test minimization in the context of product lines.

In the future, we plan to replicate our experiments in other industrial case studies for assessing the proposed weight strategy UDW. We also plan to investigate the effect of uniformly distributed weights on a diverse range of search algorithms.

References:


Using Feature Model to Support Model-Based Testing of Product Lines: An Industrial Case Study

Shuai Wang\textsuperscript{1, 2}, Shaukat Ali\textsuperscript{1}, Tao Yue\textsuperscript{1} and Marius Liaaen\textsuperscript{3}

\textsuperscript{1}Certus Software V&V Center, Simula Research Laboratory, Norway
\textsuperscript{2}Department of Informatics, University of Oslo, Norway
\textsuperscript{3}Cisco Systems, Oslo, Norway
{shuai, shaukat, arnaud}@simula.no, marliaae@cisco.com

Abstract. In the context of Model-Based Testing (MBT) of product lines, effort required to develop models can be significantly reduced by applying systematic product line modeling and configuration methodologies. In our previous work, we developed such a methodology to capture variability in configurable UML state machines and aspect state machines. For each product, these state machines are to be configured for generating executable test cases. In this paper, we extended this methodology using Feature Model for Testing (FM_T) and Component Family Model for Behaviors (CFM_B). FM_T captures variable testing functionalities of a product line, whereas CFM_B provides an abstraction layer on top of the configurable state machines. With our current methodology, a test engineer doesn’t need to acquire expertise on behavioral modeling and can simply configure models for a product by selecting features in FM_T and configuring provided attributes in CFM_B. The configured models are then given input to our model-based testing tool, TRansformation-based tool for Uml-based Testing (TRUST) for executable test case generation. We applied our extended methodology to a product line of video conferencing system developed by Cisco Systems, Norway. Results show that the methodology significantly reduces the complexity of configuration; thereby significantly reducing required effort and cost (e.g., in terms of training). In addition, it does not require test engineers to have expertise in UML modeling, aspect-oriented modeling, and OCL specification and therefore eases the adoption of MBT in industry.

Keywords: Product Line; Model-Based Testing; Feature Model; Component Family Model; Aspect State Machines
1. Introduction

Model-Based Testing (MBT) supports automated generation of executable test cases from models in a systematic manner [1]. Major effort required to apply MBT in practice is to develop models representing a system under test. In the context of product lines, such effort can be significantly reduced via systematically managing commonalities and variabilities of such models developed for testing different products in a product line [2].

In our previous work, we proposed a product line modeling and configuration methodology that systematically captures various types of behavioral variability of a product line using behavioral models, including standard UML state machines modeling functional behaviors and aspect state machines modeling non-functional behaviors [3]. Results of applying this methodology to a product line of Video Conferencing Systems (VCSs) (called Saturn developed by Cisco Systems, Norway) revealed that the modeling effort required could be significantly reduced as compared to an approach where a new product has to be modeled from the scratch [3].

While investigating the adoption of this methodology in Cisco, we discovered that test engineers are required to have expertise on developing aspect state machines and Object Constraint Language (OCL) constraints—two key artefacts required to configure a product. Moreover, test engineers are also required to be familiar with concepts of UML class diagrams, UML state machines, aspect class diagrams, and aspect state machines—four key UML diagrams used for modeling functional and non-functional behaviors of the Saturn product line. To ease the adoption of the MBT approach in industry, it is essential to seek a solution that can shield test engineers from all the above-required modeling expertise.

In another our recent work [4, 5], we proposed an approach to support automated test case selection in the context of product line. As part of the approach, we developed a Feature Model for Testing (FM_T) and a Component Family Model for Testing (CFM_T) as shown in Fig. 1 (the black part). FM_T captures commonalities and variabilities of a product line and CFM_T captures the overall structure of a large number of test cases in the test case repository. The approach was evaluated based on a questionnaire-based survey in Cisco with the objective of investigating the adoption of FM_T and CFM_T to Saturn to support automated selection of test cases for testing a specific product of the
product line [4]. Results showed that test engineers at the test group of Cisco are positive about using FM_T and CFM_T through the Test Selection Front-end we developed instead of manually selecting test cases from their Product Line Test Case Repository (Fig. 1).

In this paper, we propose an extension to our previously proposed modeling and configuration methodology, reusing the FM_T we proposed in [4, 5], for the purpose of reducing modeling effort required to configure behavioral models for a product. As shown in Fig. 1 (the white part), we developed a Component Family Model for Behaviors (CFM_B) to associate its configurable parameters to model elements of a large number of behavioral models developed as part of our previous work [3] for supporting MBT. As shown in Fig. 1, test engineers are only required to perform selection and configuration through the Test Generation Front-end for FM_T, such that all relevant behavioral models can be selected and configured automatically, based on the links built between FM_T and CFM_B, and between CFM_B and the Product Line Behavioral Model Repository. The configured models are then given input to our MBT tool called TRansformation-based tool for Uml-baSed Testing (TRUST) for generating executable test cases [6].

![Fig. 1. An overview of the proposed methodology](image)

With our extended methodology, most of the details of the behavioral models are hidden from a test engineer, which is not the case for our previous work [3]—test engineers need to create, configure and select behavioral models. To compare with our previous work, this extended methodology significantly reduces the complexity of configuration; thereby significantly reducing required effort and cost (e.g., in terms of training). In addition, the extended methodology does not require test engineers to have expertise of UML modeling, aspect modeling, and OCL constraints specification, which based on our study and experience of working with our industrial partner, is an obstacle to apply MBT in practice.

The rest of the paper is organized as follows. Section 2 provides an introduction about aspect state machines, feature model, component family model and a summary of our
previous work in [3]. Section 3 describes the running example that will be used to exemplify our proposed methodology. In Section 4, we present our methodology with formalized notations and specify how each type of behavioral variability is modeled and configured. In Section 5, we discuss the tool support. Section 6 presents the evaluation of the methodology, including an industrial case study. Section 7 discusses the related work and Section 8 concludes the paper and discusses our future plans.

2. Background

In this section, we briefly introduce aspect state machines (Section 2.1), followed by related background of feature model and component family model (Section 2.2) and the prior product line modeling and configuration methodology in [3] (Section 2.3).

2.1. Aspect State Machine (AspectSM)

AspectSM [7, 8] is a UML profile supporting the modeling of system robustness behavior—a very common type of crosscutting behavior in communication and control systems. Though AspectSM was originally defined to support scalable, model-based, robustness testing, including test case and oracle generation, it is applicable to model crosscutting behaviors and AspectSM in particular simply relies on UML state machines to do it all. In AspectSM, the core functionality of a system is modeled as one or more standard UML state machines (called base state machines). Crosscutting behavior of the system (e.g., robustness behavior) is modeled as aspect state machines using the AspectSM profile. The AspectSM profile specifies stereotypes for all features of Aspect-Oriented Modeling (AOM), in which the concepts of Aspect, Joinpoint, Pointcut, Advice, and Introduction are the most important ones and are defined as stereotypes. More details and examples can be found in [3].

2.2. Feature Model (FM) and Component Family Model (CFM)

Feature modeling is a hierarchical modeling approach for capturing commonalities and variabilities in product line [9, 10]. FM can be represented as a 2-tuple (features, constraints) with four types of features, namely mandatory, optional, alternative and or. A mandatory feature means it must be included if its father feature is included in the current selection. The selection of an optional feature is optional even if its father feature is included. A father feature with a set of alternative features describes that only one of the alternative features can be included if their father feature is included. A father feature with
a set of or features means at least one of the or features is included if their father feature is included. In addition, FM contains cross-tree constraints which are supplementary relationships among unrelated features. There are two kinds of such constraints, namely require and mutually exclusive. A require relation among two features (a source and a target) means if the source feature is included into the current selection, the target feature must also be included. A mutually exclusive relation has the opposite meaning, saying that if the source feature is included then the target feature cannot be included into the current selection.

A CFM is used to represent how products are assembled and generated in a product line by modeling relationships among software architectural elements [11]. CFM can be represented as a 4-tuple (components, parts, source elements, restrictions). Components are named entities organized into a tree-like structure that can be of any depth. Each component represents one or more functional elements of the products in product line (e.g. C functions, Java classes). Parts are named and typed entities. Each part belongs to a component and contains one or more source elements. A part can be associated with given programming language features, classes or objects, but it can also be associated with other key elements. A source element is an unnamed but typed entity. Source elements are usually used to determine how the source code for the specified element is generated. Restrictions specify conditions under which a component, part or source element may be excluded from a final selection [11, 12].

2.3. Prior Modeling and Configuration Methodology

In [3], we proposed a modeling and configuration methodology to support MBT in the context of product lines. As shown in Fig. 2, the prior methodology included: 1) a classification of four types of behavioral variability, which are: state machine variability, state machine model element variability, class attribute value variability and class attributes selection variability; 2) a product configuration process including six steps to resolve the four types of behavioral variability using AOM and OCL.

State machine variability. This type of variability is related to the behaviors that are specified using significantly different state machines across different products. In this case, each individual state machine can be considered as a variant. Therefore, resolving this type of variability becomes the selection of proper behavior. Activity A1 (Fig. 2) in the configuration process resolves this variability by selecting a core behavior from a set of
available core behaviors.

**State machine model element variability.** This type of variability refers to not-extensive model elements in state machines and aspect state machines (e.g., transitions and states) for various products. These model elements can be considered as variants, stereotyped as <<Variability>>. *Activity A4* (Fig. 2) resolves this variability by developing a set of aspect state machines to configure these model elements.

**Class attribute value variability.** This type of variability is related to various configurable class attributes, which are used in guards and/or state invariants of state machines. Stereotype <<Variability>> is also applied on configurable class attributes to specify they are variants. *Activity A2/A3* (Fig. 2) resolves this type variability by developing OCL constraints to configure class diagrams and state machines modeling functional behaviors, and aspect class diagrams and aspect state machines modeling non-functional behaviors.

**Class attributes selection variability.** This type of variability refers to various configurable software/hardware parameters, which can be modeled as attributes in class diagrams. *Activity A5/A6* (Fig. 2) resolves this variability by developing aspect state machines to configure these attributes in class diagrams for software/hardware.

**3. Running Example**

We present, in this section, a running example that will be used to exemplify our methodology (Section 4). The running example is a simplified version of the Saturn product line of Cisco. The Saturn product line has a set of products (e.g., C20, C40, C60, and C90).
The core functionality of a VCS is to establish a videoconference and the Saturn product line supports the following two types of videoconferences: Multi-way and Multi-site. A Multi-way call in VCS products means one VCS can dial at most to only one Endpoint (EP1) and put the current call on hold to dial to another Endpoint (EP2). The VCS can then switch between EP1 and EP2, but can have only one active call at a time. Compared with a Multi-way call, a Multi-site call allows users to make calls to more than one Endpoint simultaneously. There is also a possibility of transmitting presentations in parallel to a videoconference using VCS products. Presentations can be sent only by one conference participant at a time and all others receive it. In the current VCSs, some of them, e.g., C20 only supports Multi-way calls and others, e.g., C40, C60 and C90 support Multi-site calls. Meanwhile, for VCSs supporting Multi-site call, the maximum number of calls supported may differ, for instance, the maximum number of simultaneous calls for C60 and C90 is three and five, respectively. The Saturn product line supports two protocols for videoconference: H323 and SIP.

4. Methodology

In this section, we first formally define notations that will be used to specify our methodology (Section 4.1). Second, we present how each type of behavioral variability is modeled and configured using FM_T and CFM_T (Sections 4.2-4.5). In Section 4.6, we discuss the consistency between different artefacts we have defined.

4.1. Notations

Suppose, we have a product line $P$ that has a set of products $P = \{p_1, p_2, \ldots, p_{np}\}$; where $np$ is the total number of products in $P$. To capture the behaviors of all the products, in our previous work [3], we developed a configurable product line Behavioral Model Repository (BMRepository) shown in Fig. 1.

$$BMRepository = \{SM, CD, ASM_N, ACD_N, ASM_C, ACD_C, OCL_C\}$$

$SM = \{sm_1, sm_2, \ldots, sm_{n_{sm}}\}$ is a set of UML state machines in the repository and each $sm_t$ is used to model a functional behavior of the product line. An example of such behavior includes the Multi-way behavior, which is modeled as a set of five hierarchical state machines in the context of Saturn. $n_{sm}$ is the total number of state machines in the repository.

$CD = \{cd_1, cd_2, \ldots, cd_{n_{cd}}\}$ contains a set of UML class diagrams capturing the structure
of the system including its Application Programming Interfaces (APIs), state variables, software and hardware configurations. Notice that each $sm_i$ is linked to exactly one $cd_j$ and $cd_j$ may have a set of associated state machines from $SM$. $n_{cd}$ is the total number of UML class diagrams in the repository.

$ASM_N = \{asm_{n_1}, asm_{n_2}, ..., asm_{n_{asm,n}}\}$ is a set of aspect state machines modeling non-functional behaviors. A typical example of such a behavior in the Saturn product line is the robustness behavior [3]. $n_{asm,n}$ is the total number of aspect state machines specifying non-functional behaviors in the repository.

$ACD_N = \{acd_{n_1}, acd_{n_2}, ..., acd_{n_{acd,n}}\}$ is a set of aspect class diagrams capturing information such as methods and attributes appearing in aspect state machines [3]. Notice that each $asm_{n_i}$ in $ASM_N$ is linked to exactly one $acd_{n_j}$ in $ACD_N$ and each $acd_{n_j}$ may be associated with a set of aspect state machines from $ASM_N$. $n_{acd,n}$ is the total number of aspect class diagrams capturing non-functional behaviors in the repository.

$ASM_C = \{asm_{c_1}, asm_{c_2}, ..., asm_{c_{n_{asm,c}}}\}$ is a set of aspect state machines for configuration developed for the State Machine Modeling Element Variability (Section 2). A typical example of such configuration in Saturn is related with configuring the maximum percentage of packet loss supported by a product during a videoconference by developing such kind of aspect state machines [3]. $n_{asm,c}$ is the total number of aspect state machines for configuration in the repository.

$ACD_C = \{acd_{c_1}, acd_{c_2}, ..., acd_{c_{n_{acd,c}}}\}$ is a set of aspect class diagrams developed for modeling the information required for $ASM_C$. Notice that each $asm_{c_i}$ in $ASM_C$ is linked to exactly one $acd_{c_j}$ in $ACD_C$ and each $acd_{c_j}$ may be associated with a set of aspect state machines for configuration from $ASM_C$. $n_{acd,c}$ is the total number of aspect class diagrams for configuration in the repository.

$OCL_C = \{ocl_{c_1}, ocl_{c_2}, ..., ocl_{c_{n_{ocl,c}}}\}$ is a set of OCL constraints for configuration. An OCL constraint can be written to configure one or more class diagrams, aspect class diagrams for configuration and aspect class diagrams for non-functional behaviors, which are, respectively, used to configure corresponding state machines, aspect state machines for configuration and aspect state machine for non-functional behaviors. $n_{ocl,c}$ is the total number of OCL constraints in the repository.
The product line \( P \) can be represented as a \( FM_T \) [4, 9], consisting of a set of features:
\[
FM_T = \{f_1, f_2, \ldots, f_{n_f}\},
\]
where \( n_f \) is the total number of features for \( P \). For modeling the \( BMRepository \), a \( CFM_B \) has been developed, which can be formulated as \( CFM_B = \{c_1, c_2, \ldots, c_n\} \) comprising of a set of components, where each component \( c_i \) represents a set of relevant behaviors, such as video call in the Saturn product line. More specifically, each component \( c_i \) can be further decomposed into a set of parts: \( c_i = \{p_{a_1}, p_{a_2}, \ldots, p_{a_{n_i}}\} \), where each part \( p_{a_j} \) can represent one behavior, such as Multi-way and Multi-site at the same time being associated with a set of state machines (\( SM_{ji} \)), aspect state machine for configuration (\( ASM_{C_{ji}} \)) and aspect state machine for non-functional behavior (\( ASM_{N_{ji}} \)) in the repository. Moreover, \( p_{a_{ji}} \) may have a set of configurable attributes \( \{a_{1_{ji}}, a_{2_{ji}}, \ldots, a_{n_{ji}}\} \), which are associated with a set of relevant configurable attributes in their corresponding class diagrams (\( CD_{ji} \)), aspect class diagram for configuration (\( ACD_{C_{ji}} \)), aspect class diagram for non-functional behavior (\( ACD_{N_{ji}} \)) or OCL constraints for configuration (\( OCL_{C_{ji}} \)) for configuring \( SM_{ji}, ASM_{C_{ji}} \) and \( ASM_{N_{ji}} \).

Note that each component \( C_i \) or part \( p_{a_{ji}} \) can be linked with one or more features in \( FM_T \) via restrictions (Section 2). In the following sections, we show how the four types of behavioral variability are modeled and configured using our methodology. For each type of variability, the following two steps are involved: 1) selecting a set of relevant features in \( FM_T \) for a product; 2) configuring the selected attributes in \( CFM_B \) as the result of step 1). We also provide an example for each of these steps for each type of variability based on the above-mentioned running example (Section 3).

4.2. State Machine Variability

Selection. By selecting a feature \( f_k \in FM_T \), a set of related parts in \( CFM_B \) is selected automatically via restrictions. As a result, the following two types of behavioral models are selected automatically through \( CFM_B \): 1) Associated class diagrams (\( CD_k \subset CD \)) and state machines (\( SM_k \subset SM \)) modeling functional behaviors; 2) Associated aspect class diagrams (\( ACD_{N_k} \subset ACD_N \)) and aspect state machines (\( ASM_{N_k} \subset ASM_N \)) modeling non-functional behaviors.
**Configuration.** Since all associated class diagrams and state machines for functional behaviors, and aspect class diagrams and aspect state machines for non-functional behaviors have been chosen, there is no need to perform configuration.

**Example.** Fig. 3 shows an example for state machine variability in the context of Saturn product line and has the following three main parts: 1) An excerpt of FM_T; 2) An excerpt of CFM_B; 3) Two associated state machines. For FM_T, there are two core features, namely, *Multi-way* and *Multi-site* (exclamation marks mean mandatory features and double-array marks describe alternative features). Since each product in Saturn product line can only support either Multi-way call or Multi-site call, *Multi-way* and *Multi-site* are represented as alternative features in FM_T, i.e., only one of them can be selected for a product. As shown in Fig. 3, in CFM_B, parts are linked with relevant features via restrictions at the same time the corresponding class diagrams and state machines are associated with the related parts in CFM_B. For instance, since C90 supports Multi-site call, feature *Multi-site* will be selected in FM_T and then part *Multi-site* will be selected automatically via restrictions defined in CFM_B. Meanwhile, the state machine associated with part *Multi-site* will be chosen automatically from the repository, and the class diagram related to Multi-site call will also be selected automatically, which is not shown in Fig. 3 due to space limitation.

### 4.3. State Machine Model Element Variability

**Selection.** By selecting a feature \( f_k \in FM_T \), a set of related parts in CFM_B will be selected automatically via restrictions. As a result, the following three types of behavioral
models are selected automatically: 1) Associated class diagrams \((CD_k \subseteq CD)\) and state machines \((SM_k \subseteq SM)\) modeling functional behaviors; 2) Associated aspect class diagrams \((ACD_Nk \subseteq ACD_N)\) and aspect state machines \((ASM_Nk \subseteq ASM_N)\) modeling non-functional behaviors; 3) Associated aspect class diagrams \((ACD_Ck \subseteq ACD_C)\) and aspect state machines \((ASM_Ck \subseteq ASM_C)\) for configuration.

**Configuration.** It requires configuring a set of attributes in the selected parts of CFM_B. Once these attributes are configured, the associated \(ACD_Ck\) and \(ASM_Ck\) are configured automatically.

**Example.** Fig. 4 shows an example and it has the following three main parts: 1) An excerpt of FM_T; 2) An excerpt of CFM_B with two configurable attributes; 3) An associated aspect state machine for configuration. As shown in Fig. 4, attribute constraint of stereotype \(<<<\text{Before}>>\) has the following value assigned:

```
```

The above constraint is defined in the context of Network class in an aspect class diagram (belonging to \(ACD_C\)) associated with the \(ConfigureStateInvariants\) aspect state machine belonging to \(ASM_C\) (Fig. 4).

As shown in Fig. 4, attributes \(packetLossMin\) and \(packetLossMax\) in the Multi-site part of CFM_B are linked to their corresponding attributes in the Network class of the aspect class diagram in the repository. For instance, C90 supports Multi-site call and the maximum percentage of packet loss that C90 can deal with is 10%. First, the Multi-site part of CFM_B is
automatically selected by selecting the *Multi-site* feature in FM_T via restrictions. Second, the two related attributes in the *Multi-site* part are set as “packedLossMin = 0” and “packedLossMax = 10”, respectively, while the corresponding attributes in *Network* class are set automatically. As a result, attribute constraint of <<Before>> (highlighted with a black box) in Fig. 4 is configured automatically since it uses the *packetLossMin* and *packetLossMax* attributes of class *Network*.

### 4.4. Class Attribute Value Variability

**Selection.** For this variability, the selection process is similar as the state machine variability. The difference is that the associated OCL constraints for configuration ( \(OCL_{C_k} \subset OCL_C\)) for functional/nonfunctional behaviors will be also selected automatically in addition to the associated class diagrams and state machines modeling functional behaviors, and the associated aspect class diagrams and aspect state machines modeling non-functional behaviors.

**Configuration.** It requires configuring a set of attributes in the selected parts of CFM_B, which in turn configure attributes in one or more classes in the associated class diagrams and aspect class diagrams. The \(OCL_{C_k}\) will be configured automatically since these are defined based on the attributes defined in class diagrams/aspect class diagrams. Finally, the state machines and aspect state machines associated to the selected class diagrams and aspect class diagrams will be configured automatically.

**Example.** Fig. 5 shows an example of this type of variability and has the following three
main parts: 1) An excerpt of FM_T; 2) An excerpt of CFM_B with a configurable attribute; 3) An associated state machine modeling a functional behavior. As shown in Fig. 5, we have the following configuration OCL constraint:

```
context Multisite inv: self.MaxNumberOfCalls = self.MaxCalls
```

Notice that the above constraint is defined in the context of the Multisite class (Fig. 5), which is associated with the Multisite state machine. The MaxCalls attribute of the Multisite part is linked to the MaxCalls attribute in the Multisite class. For instance, C60 supports Multi-site call and maximal three simultaneous calls. First, the Multi-site part will be selected, which will automatically select the Multisite class, the constraint on the Multisite class, and the Multisite state machine from the repository. Second, in the selected Multi-site part, attribute MaxCalls is set to three, which will configure the MaxCalls attribute in the Multisite class. Doing so, will automatically configure the constraint defined on the Multisite class. As a consequence, the Multisite state machine will be configured since the MaxNumberOfCalls attribute of the Multisite class is used in the guards and state invariants (as highlighted) of the state machine (Fig. 5).

### 4.5. Class Attributes Selection Variability

**Selection.** For this variability, the selection process is similar as the state machine model element variability. The difference is that this variability deals with various software/hardware configuration parameters while the state machine model element variability focuses on variability in model elements of functional state machines.

**Configuration.** The configuration process is also similar with the state machine model element variability except that here we configure software/hardware parameters.
Example. Fig. 6 shows an example for this type of variability and has three main parts:
1) An excerpt of FM_T; 2) An excerpt of CFM_B with a configurable software attribute; 3) An associated aspect state machine. As shown in Fig. 6, we introduced the following constraint to all state machines using the &lt;&lt;Introduction&gt;&gt; stereotype in the aspect state machine:

```
```

Notice that the attributes H323_Mode and ModeOfH323 are defined in the NetworkServices class. For instance, C90 supports Multi-site call and H323 protocol for videoconference. First, we select the Multi-site feature in FM_T, which will select the Multi-site part in CFM_T. As a result, the associated NetworkServices and Saturn aspect classes and aspect state machine (IntroduceConstraint) will be selected from the repository (Fig. 6). Second, the attribute ModeOfH323 in the Multi-site part is set as “On” and as a result the attribute ModeOfH323 in NetworkServices class will be set to “On”. This will configure H323_Mode software configuration specified in NetworkServices class followed by configuration of a constraint (as highlighted) in the aspect state machine since the constraint uses the H323_Mode attribute.

4.6. Consistency Between Different Artefacts

At this stage, we have defined a set of artefacts, i.e., FM_T, CFM_B, four key UML diagrams (i.e., class diagrams, state machines, aspect class diagrams, and aspect state machines) and specified a set of relationships between different artefacts. Notice that FM_T and CFM_B are built and maintained by a commercial tool called Pure::Variants (P::V) and a tool called Import Plugin and Transformation for Behaviors (IPTB) we developed (Section 5) while all the four UML diagrams are built and maintained by another commercial tool called Rational Software Architect (RSA) developed by IBM and our TRUST tool 34. Moreover, all these relationships can be checked by these tools automatically in all our cases. We summarize all artefacts and relationships between various modeling elements of the artefacts in Table 1.

<table>
<thead>
<tr>
<th>Artifact to Artifact</th>
<th>Relationship Type</th>
<th>Element to Element</th>
<th>Checking</th>
</tr>
</thead>
<tbody>
<tr>
<td>FM_T ↔ FM_T</td>
<td>Cross Tree Relationships</td>
<td>Feature ↔ Feature</td>
<td>A (P::V)</td>
</tr>
</tbody>
</table>

Table 1. Consistency between Different Artefacts*

Fig. 12. An example for modeling and configuring class attributes selection variability
<table>
<thead>
<tr>
<th>CFM_B → FM_T</th>
<th>Restrictions</th>
<th>Component/Part → Feature</th>
<th>A (IPTB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CFM_B → SMs</td>
<td>Traces</td>
<td>Component/Part → SM</td>
<td></td>
</tr>
<tr>
<td>CFM_B → CDs</td>
<td>Part.attribute → Class.attribute</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CFM_B → ASMs</td>
<td>Component/Part → ASM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CFM_B → ACDs</td>
<td>Part.attribute → Aspect Class.attribute</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CDs → SMs</td>
<td>OCL Constraints</td>
<td>Class.attribute → State Invariant</td>
<td>A (RSA/TRUST)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Class.attribute → Guard</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Operation Parameter → Guard</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Links</td>
<td>Operation → Trigger</td>
<td>A (RSA)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Signal → Trigger</td>
<td></td>
</tr>
<tr>
<td>ACDs → ASMs</td>
<td>OCL Constraints</td>
<td>Aspect Class.attribute → State Invariant</td>
<td>A (RSA/TRUST)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Aspect Class.attribute → Guard</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Aspect Class.attribute → Change Event</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Operation Parameter → Guard</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Links</td>
<td>Operation → Trigger</td>
<td>A (RSA)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Signal → Trigger</td>
<td></td>
</tr>
</tbody>
</table>

* ↔: The relationships between source artifact and target artifact are bidirectional. →: The relationships between source artifact and target artifact is one way. SMs: State Machines. CDs: Class Diagrams. ASMs: Aspect State Machines. ACDs: Aspect Class Diagrams. A: automated checking by the relevant tools.

The first column presents source artifact and target artifact of a relationship, the second column describes the relationship type, whereas the third column presents the exact model elements of artefacts that are related. The last column indicates whether and how such relationships can be checked automatically for correctness. For instance, two types of cross tree relationships (i.e., require and mutually exclusive) can be specified between a source feature to a target feature and such cross tree relationships can be checked automatically by the P::V tool (Section 1). Notice that a feature in FM_T can be a source feature or a target feature. Moreover, various restrictions are assigned in CFM_B (i.e., components or parts), which associate components or parts in CFM_B with features in FM_T. Such restrictions can also be checked automatically by the tool IPTB. Notice that restrictions can only be assigned in CFM_B, i.e., such relations are one way from CFM_B to FM_T. Similarly, other types of relations are specified between artefacts, e.g., traces from CFM_B to the four UML diagrams, which can be checked automatically by the IPTB tool and constraints/links from UML class diagrams/state machines to aspect class diagrams/aspect state machines, which can be checked automatically by the RSA/TRUST tool.

### 5. Automation

In this section, we present the tool support for our proposed methodology. Our tools are implemented as Eclipse plugins using Java.
In our methodology, a CFM_B is required to be built to maintain all the links to a FM_T and to the *Behavioral Model Repository*. Notice that our process of building a CFM_B is automated. In our current industrial application, we developed the tool IPTB (Fig. 7), which automatically builds a CFM_B and its configurable attributes based on system information. System information about VCS products is maintained as various XML files and includes API information, system states and software/hardware configuration. The tool also builds an initial set of restrictions from CFM_B to FM_T, based on the XML files capturing the system information. Notice that these restrictions in the automatically generated CFM_B still needs to be validated by test engineers to ensure their correctness. Therefore, building CFM_B can be summarized as: 1) obtaining the input for IPTB (in Cisco case, the input is a set of specific XML files as discussed above); 2) importing the input files to IPTB and building the CFM_B with related restrictions; 3) refining the CFM_B with restrictions by test engineers. Notice the effort for refining is one time effort required.

![Integrated tool architecture for the proposed methodology and our previous work](image)

Fig. 7. Integrated tool architecture for the proposed methodology and our previous work for a product line and then reused each time when a new product is tested. The *Test Generation Front-end* interface (Fig. 7) allows a test engineer to perform the selection and configuration steps of our current methodology (Section 4).

However, in other contexts, it may not be feasible to build a CFM_B automatically. Therefore, a CFM_B, restrictions from CFM_B to FM_T, and traces from CFM_B to behavioral models have to be built manually, which is again a one-time effort required for a product line.
Moreover, configured models in our context mainly consist of four types of diagrams, i.e., class diagrams, state machines, aspect class diagrams, aspect state machines, and OCL constraints on them as we discussed in Section 4. Recall that class diagrams and state machines are used for functional MBT in our industrial application and thus are directly given input to the TRUST tool for executable test script generation. On the other hand, configured aspect class diagrams and aspect state machines (modeling non-functional behavior) are required to be woven before the executable test cases can be generated from them using TRUST. Due to this reason, aspect class diagrams and aspect state machines are given input to our weaver [7] and the woven models are given input to TRUST for executable test case generation.

In our previous work [4, 5], we developed another tool called Import Plugin and Transformation (IPT) (Fig. 7) to build a CFM_T automatically for capturing the structure of a large number of test cases in the repository. The input of IPT is test case information (e.g., test ID, test scripts, software/hardware resources required for executing test cases, tags associated with test cases). Such information can be automatically obtained as an xml file from the repository of test cases in Cisco. The tool also builds restrictions from CFM_T to FM_T automatically using tag information associated with test cases. Same as for CFM_B, in other industrial contexts, it may not be possible to build CFM_T automatically. But in any case, it is only required to be built once for a product line and reused for testing every product of the product line. The Test Selection Front-end interface (Fig. 7), allows a test engineer to perform selection of test cases as we discussed in [4, 5].

6. Evaluation

In this section, we evaluate our methodology with two means: 1) comparing it with our previous work [3] to show the differences and potential benefits with the aims to ease practical adoption, and 2) reporting an industrial case study to demonstrate the benefits of applying our methodology in an industrial setting.

6.1. Comparison with our Previous Work

To configure a product, it takes six steps both for the existing and current methodologies, as shown in Table 2.

<table>
<thead>
<tr>
<th>Step</th>
<th>Previous Methodology</th>
<th>New Methodology</th>
<th>Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>Select a core behavior</td>
<td>Select a core feature</td>
<td>Selecting behavior vs. feature</td>
</tr>
</tbody>
</table>
For A1, our proposed methodology only needs to select a feature in FM_T instead of selecting class and state machine diagrams from the repository. For A2 and A3, the effort for configuring standard UML class and state machine diagrams, aspect class and state machine diagrams, and OCL constraints is reduced to assign values to configurable attributes in CFM_B. For A4, A5, and A6, the methodology proposed in the previous work requires developing aspect state machines for configuration, while the new methodology only needs to assign values to configurable parameters attributes in CFM_B, as steps A2 and A3.

In summary, as shown in the third column of Table 2, our new methodology only needs test engineers to perform two activities: selecting features through the Selection Front-end (Table 2) and assign values to configurable attributes of CFM_B through the Configuration Front-end (Table 2). For instance, for the four examples as shown in Fig. 3, Fig. 4, Fig. 5 and Fig. 6, test engineers only need to select Multi-site feature once in FM_T through the Selection Front-end and configure provided configurable attributes once through the Configuration Front-end instead of selecting features and configuring attributes for each variability in practice. Moreover, our new methodology does not need test engineers to have expertise on UML modeling, aspect state machine modeling, and OCL specification, which significantly saves the cost required for training, thereby easing the adoption of MBT methodology in industry.

### 6.2. Industrial Case Study

Our case study is the Saturn VCS product line developed in Cisco [13]. The Saturn family consists of various hardware codecs ranging from C20 to C90. C20 is the lowest end product with minimum hardware and has lowest performance in the family and C90 is the highest end product with best performance among all the products.

<table>
<thead>
<tr>
<th>A2</th>
<th>Configure functional models using OCL constraints</th>
<th>Select features and configure attributes in CFM_B by assigning values</th>
<th>Configuring models and constraints vs. selecting features and configuring attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>A3</td>
<td>Configure non-functional models using OCL constraints</td>
<td></td>
<td>Developing new models vs. selecting features and configuring attributes</td>
</tr>
<tr>
<td>A4</td>
<td>Develop aspect state machines to configure model elements</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A5</td>
<td>Develop aspect state machines for software configuration</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A6</td>
<td>Develop aspect state machines for hardware configuration</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3. Summarized Results for Configuring Various Products
Saturn product line family consists of 20 subsystems such as audio and video subsystems. Each subsystem can run in parallel to the subsystem implementing the core functionality that deals with establishing videoconferences. Each subsystem has at least one state machine specifying its functionality and on average such state machine has five states and 11 transitions. The biggest subsystem state machine has nine hierarchical state machines with 22 states and 63 transitions. Saturn product family models for non-functional behaviors consist of 5 aspect class diagrams and 5 aspect state machines modeling various robustness behaviors. The largest aspect state machine specifying robustness behavior has 3 states and 10 transitions, which would translate into 1604 transitions in standard UML state machines. Saturn product line family models also consist of 124 hardware configuration parameters and 99 software configuration parameters. All these models are stored in the Saturn product line repository. Moreover, the repository contains configuration models and OCL constraints. The repository has 24 configurable OCL constraints and 9 configurable aspect class and aspect state machine diagrams.

Results of configuring various products in Saturn are summarized in Table 3. The columns show the various types of product line models, which must be configured for various products. The Core Behavior column indicates the configuration of the state machine variability for each product, where all the products support Multi-site except C20. The Functional Behavior and Non-Functional Behavior columns show the number of instances of the class attribute value variability. For all the products, 13 and 11 configuration parameters need to be configured for each product regarding functional behaviors and non-functional behaviors, respectively. The SC (Software Configuration) and HC (Hardware Configuration) columns show the number of class attributes that need to be configured. Using our methodology, we need to configure on average 24 attributes in CFM_B, which correspond to 24 OCL constraints in the product line behavioral model repository. These OCL constraints therefore automatically configure functional and non-functional models in the repository.

7. Related Works
Feature model plays a key role for variability management in the context of product line [9]. However, features in feature model are only symbols without being associated with other models such as modeling behavior [14]. To address such challenge, several techniques for mapping feature models to other models have been proposed in the existing literature [14-17]. In [14], Czarnecki and Antkiewicz proposed a template-based approach to map feature models to other models such as UML activity and class models mainly for requirement analysis. In [15-17], two similar approaches called FeatureMapper and VML* were proposed to map feature models to other models (e.g., use case models, activity models, and business process models) by building a mapping model or a family of languages to specify various relationships between features in feature model and their realization in the other models for supporting model-driven development. As compared with the existing works, first, our objective is different, i.e., we aim at obtaining a set of relevant behavioral models (i.e., UML class diagrams, UML state machines, aspect class diagrams, and aspect state machines) using FM_T and CFM_T for test case generation. Second, our methodology can configure various behavioral models by configuring related attributes in CFM_B, which is not addressed in the existing literature.

Moreover, in our previous work [3], we compared a set of product line modeling approaches in the literature [18-22] with our previous work. The key differences with the existing works include: 1) We defined a comprehensive classification of behavioral variability in behavioral models including UML state machines and aspect state machines to support MBT of product lines; 2) We focused on both functional and non-functional models for MBT; 3) Our objective, i.e., to reduce modeling effort to support MBT, is not studied in the literature.

As compared with our previous work in [3], the methodology proposed in this paper stores configurable behavioral models, configuration aspect state machines, and configuration OCL constraints in a repository for a product line. Details of these models are hidden from a test engineer by adding an additional layer of abstraction on top of these models via a CFM_B. All the links to models are captured in CFM_B and are accessible from a FM_T capturing various testing functionalities. Notice that this FM_T is already being used by Cisco for the purpose of test selection [4, 5]. Instead of configuring behavioral models using OCL constraints and developing aspect state machines, our proposed methodology only needs to assign values to configurable attributes in CFM_B,
which eliminates the needs to acquire expertise of UML modeling, aspect-oriented modeling, and OCL constraints.

To ease practical adoption of MBT, our main objective is to hide details of behavioral models from test engineers and reduce the effort for modeling and configuration using FM and CFM in the context of product lines. To the best of our knowledge, existing works have not covered such an objective: applying FM and CFM to product line for supporting systematic and automated reuse of behavioral models to enable MBT, with the consideration of reducing modeling effort and thereby easing the adoption of MBT in practice.

8. Conclusion and Future Work

In this paper, we proposed an extension to our previous product line modeling and configuration methodology [3] to support Model-Based Testing (MBT) with the aims of reducing modeling effort and eliminating the needs to acquire expertise of UML modeling, Aspect-Oriented Modeling (AOM) and Object Constraint Language (OCL). The extended methodology includes the following main steps: 1) reusing the existing Feature Model for Testing (FM_T) to model testing functionalities of a product line; 2) defining a Component Family Model for Behaviors (CFM_B) to associate its configurable parameters to model elements of a large number of behavioral models and related OCL constraints; 3) linking CFM_B with FM_T via restrictions. With our methodology based on FM_T and CFM_B, all types of behavioral variability defined in [3] can be configured by performing two activities: feature selection in FM_T and attribute configuration in CFM_B (i.e., assigning values to configurable attributes). The configured models are then given input to our model-based testing tool, TRansformation-based tool for Uml-baSed Testing (TRUST) for executable test case generation.

We evaluated our methodology with two means. First, we compared the configuration process of our proposed methodology with our existing methodology [3] and provided discussion on similarities and differences on the processes. Second, we applied our proposed methodology to the Saturn product line and configured its four products. The results showed that our methodology significantly reduces the complexity of configuration; thereby reducing required modeling effort. Moreover, the need to acquire expertise of modeling is also eliminated.
In the future, we plan to apply our methodology to more product lines for assessing in different industrial case studies. Moreover, we are planning to conduct a questionnaire-based study with test engineers to investigate the applicability of our methodology in an industrial setup.

References:


Multi-Objective Test Prioritization in Software Product Line Testing: An Industrial Case Study

Shuai Wang¹,², David Buchmann³, Shaukat Ali¹, Arnaud Gotlieb¹, Dipesh Pradhan² and Marius Liaaen³

¹Certus Software V&V Center, Simula Research Laboratory, Norway
²Department of Informatics, University of Oslo, Norway
³Cisco Systems, Oslo, Norway
{shuai, shaukat, arnaud}@simula.no, {davbuchm, marliaae}@cisco.com, dipeshpr@ifi.uio.no

Abstract. Test prioritization is crucial for testing products in a product line considering limited budget in terms of available time and resources. In general, it is not practically feasible to execute all the possible test cases and so, ordering test case execution permits test engineers to discover faults earlier in the testing process. An efficient prioritization of test cases for one or more products requires a clear consideration of the tradeoff among various costs (e.g., time, required resources) and effectiveness (e.g., feature coverage) objectives. As an integral part of the future Cisco’s test scheduling system for validating video conferencing products, we introduce a search-based multi-objective test prioritization technique, considering multiple cost and effectiveness measures. In particular, our multi-objective optimization setup includes the minimization of execution cost (e.g., time), and the maximization of number of prioritized test cases, feature pairwise coverage and fault detection capability. Based on cost-effectiveness measures, a novel fitness function is defined for such test prioritization problem. The fitness function is empirically evaluated together with three commonly used search algorithms (e.g., (1+1) Evolutionary algorithm (EA)) and Random Search as a comparison baseline based on the Cisco’s industrial case study and 500 artificial designed problems. The results show that (1+1) EA achieves the best performance for solving the test prioritization problem and it scales up to solve the problems of varying complexity.

Keywords: Test Prioritization; Multi-objective Optimization; Software Product Lines; Search Algorithms
1. Introduction

Software Product Line Engineering (SPLE) refers to a systematic framework including a collection of methodologies, tools and techniques with the aim of designing and developing a family of similar products in a cost-effective manner [1, 2]. SPLE has attracted increasing attention from both academia and industry in recent years. Companies such as Nokia, Boeing and Philips have been reporting success cases for the adoption of SPLE into their respective practice [3]. The work presented in this paper is motivated by an industrial application of SPLE, focusing on automated testing for a Video Conferencing System (VCS) product line called Saturn, developed by Cisco Systems, Norway.

Based on the domain expertise of VCS testing and experience gained while working with our industrial partner (Cisco), we notice that the main test activities for cost-effective testing of VCS products for Saturn can be classified by employing various SPLE approaches (e.g., feature model): 1) Test Selection: Systematically selecting relevant test cases from the existing test case repository developed for testing the whole product line; and 2) Test Prioritization: Optimally prioritizing a given test suite (including a set of test cases) with the aim of minimizing cost (e.g., execution time) and maximizing the effectiveness (e.g., fault detection capability) while satisfying a given limited budget (e.g., resources). To address the first activity, we proposed an automated test case selection methodology for testing products using Feature Model (FM) and Component Family Model (CFM) [4]. As shown in Fig. 1, such methodology captures the commonalities and variabilities within a product line using a FM and models the test case repository extracted from the expertise of test engineers using a CFM. To test a given product, test engineers only require performing simple selection of relevant features in the FM and the corresponding test cases can be automatically selected from the test case repository.

Based on our further experience focusing on a future test scheduling system for testing Saturn, we observe that test prioritization is crucial due to the ever-increasing number of test cases for the product line. In practice, considering limited budget (i.e., available time and resources), it is usually not feasible to execute all the possible test cases for testing the products thus it requires an efficient strategy to prioritize the given test cases. The target of such prioritization is minimization of the cost and maximization of the effectiveness of testing while meeting the given limited budget. However, it is commonly recognized that it
is a tradeoff among various costs (e.g., required resources) and effectiveness (e.g., feature coverage) objectives, which can be formulated as a multi-objective optimization problem [5, 6].

To deal with the above-mentioned problem, we introduce a search-based multi-objective test prioritization technique as shown in Fig. 1. More specifically, cost and effectiveness measures are first defined with respect to the required objectives for prioritization. In our case, one cost measure and three effectiveness measures are carefully defined based on the expertise of VCS testing, i.e., Overall Execution Cost (OEC), Prioritized Extent of test cases (PE), Feature Pairwise Coverage (FPC) and Fault Detection Capability (FDC). A novel fitness function is further defined considering all these cost/effectiveness measures and integrated with three commonly used search algorithms (Alternating Variable Method (AVM), Genetic Algorithm (GA) and (1+1) Evolutionary Algorithm (EA)) and Random Search (RS) as a comparison baseline. When particular products need to be tested, after obtaining the relevant test cases from the test case repository [4], test engineers can further cost-effectively prioritize the selected test cases for finding the optimal order using our proposed search-based technique (Fig. 1).

Moreover, the defined fitness function along with the search algorithms is empirically evaluated using the Cisco’s industrial case study and 500 designed artificial problems. The results show that (1+1) EA achieves the best performance for solving the multi-objective
test prioritization problem and it scales up to solve the problems of varying complexity.

The rest of the paper is organized as follows: Section 2 provides a compact introduction of the selected search algorithms and Section 3 describes our test prioritization problem in an industrial context followed by a formal representation of the problem. In Section 4, we present definitions and functions for the cost/effectiveness measures and a fitness function used by all the algorithms. Section 5 describes our industrial case study and designed artificial problems followed by an empirical evaluation in Section 6. Related work is discussed in section 7 and Section 8 concludes the paper.

2. Background

In this section, we briefly introduce three commonly used search algorithms, which will be adapted for our test prioritization problem. Notice that Random Search (RS) is used as the comparison baseline to assess the performance of the algorithms, which is a common practice in the existing literature [5-8]. Notice that RS has historically been the industrial practice for test prioritization at Cisco and therefore can be a meaningful baseline for comparison.

Alternating Variable Method (AVM) was selected as a representative of local search algorithms [9]. AVM tries to optimize (minimize or maximize) fitness of a variable, while keeping the rest of the variables constant, which are generated randomly. The search is stopped if a solution or an optimized value for this variable is found. Then AVM switches to the next unchanged variable until all the variables are explored.

Genetic Algorithm (GA) was selected since it is the most commonly used global search algorithm in search software engineering [9]. More specifically, a population of individuals (i.e., candidate solutions) is evolved through a series of generations, where reproducing individuals evolve through crossover and mutation operators. Interested readers may consult the following reference for more details [10].

(1+1) Evolutionary Algorithm (EA) [11] is simpler than GA, but it can be more effective in some cases (e.g., [12]). In (1+1) EA, population size is one, i.e., only one individual exists in the population and the individual is represented as a bit string. As opposed to GAs, a bitwise mutation operator, rather than the crossover operator, is used for exploring the search space.
3. Test Prioritization Problem Description and Representation

In this section, we present our industrial test prioritization problem in detail (Section 3.1) followed by problem representation in a mathematical way to guide the search towards an optimal solution (Section 3.2).

3.1. Test Prioritization Problem in Industry

Our industrial partner in the context of this work is Cisco Systems, Inc, Norway, which develops a product line of Videoconferencing Systems (VCSs) namely Saturn to offer means for organizing high quality face-to-face meetings without the need of gathering all participants physically [13]. By employing SPLE, VCSs can be designed, implemented and marketed efficiently. The core functionality of a VCS is to establish a videoconference among different participants at various physical locations. There is also a possibility of transmitting presentations in parallel to a videoconference using VCS products. Notice that each VCS product at Cisco on average has three million lines of C code. As commercial products, VCS products must be tested thoroughly.

More specifically, as shown in Fig. 2, to test VCS products in Saturn, execution of a specific test case first requires allocation of corresponding test resources to set up the environment for execution beforehand. In our case, such test resources refer to: 1) correct software version deployed on a VCS product since test cases requires particular software for executing; and 2) correct subnet of network since various test cases require different network environment for execution to test VCS products. On the other hand, one test resource can be allocated for executing one or more test cases. Notice that it may take a different amount of time to allocate different test resources for the same test case. Since the available test resources are usually limited in practice, it is commonly recognized by test engineers at Cisco that it is of paramount importance to prioritize a given test suite that includes a set of relevant test cases. Moreover, the prioritized test cases need to achieve high degree for certain test criteria (e.g., feature pairwise coverage) and fault detection.
capability thereby improving the efficiency of testing VCS products. Therefore, based on the industrial investigation, we can summarize our prioritization problem as cost-effective prioritization of a given test suite for testing products while meeting the budget of available test resources, i.e., maximize the test cases that can be executed within the test resource budget at the same time achieving high test coverage and fault detection capability.

However, through thorough discussions with test engineers at Cisco, we observed that the current practice at Cisco for such prioritization is to manually prioritize some test cases based on the domain expertise and randomly prioritize the others that cannot be prioritized using domain expertise. Such manual and random process poses several challenges, including: 1) it is time-consuming and error-prone for manual prioritization; 2) it is mainly driven by the domain expertise thus it is not an objective and repeatable process; 3) it cannot guarantee maximum usage of the available test resources, i.e., it is possible that more test cases can be executed in another order within the given test resources; 4) it may result in the prioritized test cases having low test coverage and fault detection capability since various test engineers may obtain a different ordering of test cases, which have different test coverage and fault detection capability; and 5) it is not systematic and scalable since it largely depends on the expertise of test engineers, i.e., the current practice for prioritization is likely close to a random prioritization process.

3.2. Problem Representation

According to the aforementioned challenges, our test prioritization problem can be formulated as a multi-objective optimization problem, which takes four objectives into account (i.e., overall cost for allocating test resources, number of prioritized test cases, feature pairwise coverage and fault detection capability achieved by the prioritized test cases). To precisely define our test prioritization problem, the following definitions are provided first.

3.2.1. Basic Concepts

Let \( P = \{p_1, p_2, p_3 \ldots p_{np}\} \) be a product line with a set of products, where \( np \) is the number of products in \( P \).

\( F = \{f_1, f_2, f_3 \ldots f_{nf}\} \) is a feature model with a set of features to represent \( P \) [4], where \( nf \) is the number of features, i.e., functionalities need to be tested for \( P \).

\( F_{sub} = \{f_1, f_2, f_3 \ldots f_{nf_{sub}}\} \) is a subset of \( F \) representing the functionalities of one or more
specific products from $P$, which is to be tested [4], where $f_i$ can be any feature in $F$ ($f_i \in F$) and $nf_{sub}$ is the number of features of $F_{sub}$ ($1 \leq nf_{p_i} \leq nf$).

$TR = \{tr_1, tr_2, tr_3 \ldots tr_{ntr}\}$ is a set of available test resources used to set up a correct environment for executing the given test suite, where $ntr$ is the number of available test resources in $TR$. Each test resource $tr_i$ has a key attribute ($Sta_{tr_i}$) showing the current status of $tr_i$, which can be 0 showing $tr_i$ is still available or 1 meaning $tr_i$ has been allocated.

$TS = \{t_1, t_2, t_3 \ldots t_{nt}\}$ is a given test suite required to be prioritized, which consists of a number of test cases ($nt$) for testing one or more products from $P$.

$S = \{s_1, s_2, s_3 \ldots s_{ns}\}$ is a set of potential solutions including a certain number of prioritized test cases chosen from $TS$ within a given test resources for testing products in $P$, where $ns$ is the total number of solutions which can be measured as $ns = A_{nt}^1 + A_{nt}^2 + \ldots, A_{nt}^n = nt + nt \times (nt - 1) + \ldots, nt \times (nt - 1) \times (nt - 2) \times \ldots \times 1$. Each solution is a set of prioritized test cases meeting the given budget of test resources. Notice that when the available test resources are fully allocated, the process for prioritization will be terminated. Moreover, as the number of test cases increases, the number of potential solutions will exponentially explore. For instance, if there are 1000 test cases to be prioritized, the potential solutions will be $1000+1000\times999+\ldots+1000\times999\times\ldots\times1$, which is an extremely large solution space and requires an efficient strategy in terms of finding an optimal solution.

### 3.2.2. Cost/Effectiveness Measures

$Cost = \{cost_1, cost_2, \ldots, cost_{ncost}\}$ is a set of cost measures for test prioritization, e.g., execution time of test cases, cost of setting up correct resources for executing a specific test case.

$Effect = \{effect_1, effect_2, \ldots, effect_{neffect}\}$ is a set of effectiveness measures for minimization, e.g., number of prioritized test cases, feature pairwise coverage and fault detection capability.

$Cost (s_i, CostMeasure)$ is a mathematical function, which returns the cost of a particular solution $s_i$ from $S$ based on a cost measure ($CostMeasure$) from $Cost$. 

---

227
Effect\( (s_i, \text{EffectivenessMeasure}) \) is a mathematical function, which returns the effectiveness of a solution \( s_i \) from \( S \) based on an effectiveness measure \( \text{EffectivenessMeasure} \) from \( \text{Effect} \).

Each solution \( s_i \) in \( S \) consists of \( nt \) test cases from \( TS \) and has a set of values for cost measures and effectiveness measures, which are formally defined in the next section.

### 3.2.3. Problem Representation

Our multi-objective prioritization problem can be formulated as:

**Problem:** Search for a solution \( s_k \) from \( S \) for testing the given products to achieve the following objectives (i.e., minimum cost and maximum effectiveness):

\[
\forall s_i \in S_p \cap s_i \neq s_k:
\sum_{j=1}^{n_{\text{effect}}} \text{Effect}(s_k, \text{effect}_j) \geq \sum_{j=1}^{n_{\text{effect}}} \text{Effect}(s_i, \text{effect}_j)
\]

\[
\sum_{j=1}^{n_{\text{cost}}} \text{Cost}(s_k, \text{cost}_j) \leq \sum_{j=1}^{n_{\text{cost}}} \text{Cost}(s_i, \text{cost}_j)
\]

\( \forall tr_i \in TR: \text{Sta}_{tr_i} = 0 \), i.e., all the test resources in \( TR \) have been allocated.

Notice that for a given set of test resources, the test suite may not be executed completely since the prioritization will be terminated if all the available test resources have been allocated.

### 4. Measures and Fitness Function

In this section, we define four cost/effectiveness measures for the required objectives (Section 4.1) followed by a fitness function considering all the defined measures (Section 4.2).

#### 4.1. Cost/Effectiveness Measures

This section provides definitions and formal functions for one cost measure and three effectiveness measures based on the above-mentioned problem.

##### 4.1.1. Cost Measure

Cost measures \( \text{Cost} \) consists of one element in our context, i.e., overall execution cost \( \text{(OEC)} \) for a solution \( s_k \).

**Definition 1.** \( \text{OEC} \) is to measure the overall cost for the solution \( s_k \) including a set of prioritized test cases. More specifically, \( \text{OEC} \) includes two parts in our context: execution time \( \text{(ET)} \) for test cases and the time for allocating required test resources \( \text{(CTR)} \). Moreover, \( \text{CTR} \) is also comprised of two subparts in our context, i.e., the time for setting
correct software (TCS) and the time for setting correct subnet (TCN) for executing prioritized test cases. In our case, all these time for cost is recorded in minutes.

Function 1. OEC can be measured using the following function.

\[ OEC_{s_k} = \sum_{i=1}^{nt_{s_k}} (ET_{tc_i} + TCS_{tc_i} + TCN_{tc_i}) \]

\( nt_{s_k} \) is the number of prioritized test cases for the solution \( s_k \) within the given test resources, where \( 1 \leq nt_{s_k} \leq nt \). Notice that it is possible that \( nt_{s_k} \) is less than \( nt \) since the prioritization will be terminated when the available test resources are fully allocated.

4.1.2. Effectiveness Measures

In our context, effectiveness measures \textit{Effect} consists of three elements, i.e., prioritized extent (PE) within a given test resource, feature pairwise coverage (FPC) and fault detection capability (FDC). Notice that all these effectiveness measures are defined together with the test engineers at Cisco.

4.1.2.1. Prioritized Extent (PE)

Definition 2. PE is defined to measure the extent of prioritization for the given test cases achieved by a solution \( s_k \) within the given available test resources, i.e., to what extent, test cases can be prioritized for execution by solution \( s_k \) before all the available test resources are allocated.

Function 2. PE is calculated as follows:

\[ PE_{s_k} = \frac{nt_{s_k}}{nt} \]

\( nt_{s_k} \) is the number of prioritized test cases for the solution \( s_k \) within the given test resources while \( nt \) is the number of test cases for testing given products. Notice that \( PE \) value ranges from 0 to 1 and higher values show more test cases are prioritized within a given set of test resources.

4.1.2.2. Feature Pairwise Coverage (FPC)

Definition 3. FPC is to measure how much pairwise coverage can be achieved by a chosen solution \( s_k \) [8]. We choose FPC for test coverage based on domain knowledge, discussions with test engineers at Cisco, and history data about faults since a higher percentage of detected faults are mainly due to the interactions between test functionalities that can be represented as features in feature model.

Function 3. FPC can be measured as below.
\[ FPC_{sk} = \frac{Num_{FP_{sk}}}{Num_{FP_{sub}}} \]

\( Num_{FP_{sk}} \) is the number of feature pairs covered in the prioritized test cases for the solution \( s_k \) measured as follows.

\[ Num_{FP_{sk}} = \sum_{i=1}^{nt_{sk}} Num_{FP_{tc_i}} \]

\( nt_{sk} \) is the number of prioritized test cases for the solution \( s_k \), where \( 1 \leq nt_{sk} \leq nt \). \( Num_{FP_{tc_i}} \) is the number of unduplicated feature pairs covered by the prioritized test case \( i (tc_i) \). The feature pairs covered by \( tc_i \) can be computed as: \( Num_{FP_{tc_i}} = C_{\text{size}(F_{tc_i})}^2 \). \( \text{size}(F_{tc_i}) \) is the number of features tested by test case \( tc_i \). For instance, test case \( tc_i \) is used to test four features. Then, the feature pairs covered by test case \( tc_i \) are \( C_2^4 = 4*3/2 = 6 \). Notice that if some feature pairs are repeated ones compared with the feature pairs covered by the previous test cases, repeated pairs will be removed when computing \( Num_{FP_{sk}} \).

\( Num_{FP_{sub}} \) is all number of feature pairs covered by the features that are to tested and can be measured as: \( Num_{FP_{sub}} = C_{\text{size}(F_{sub})}^2 = nf_{sub} * (nf_{sub} - 1)/2 \). \( F_{sub} \) is the set of features representing the products that require to be tested including \( nf_{sub} \) features. For instance, if 20 features are to tested, all covered feature pairs are \( C_2^{20} = 190 \). Notice \( FPC \) is calculated for a chosen test solution ranging from 0 to 1 and a higher value shows better feature pairwise coverage.

**4.1.2.3. Fault Detection Capability (FDC)**

**Definition 3.** \( FDC \) is defined to measure the fault detection capability achieved by a solution \( s_k \) within the given test resources.

In our context, fault detection capability refers to the rate of successful execution for a test case in a given time, e.g., a week in our case. More specifically, the execution of a test case can be defined as a *success* if it can detect faults in a given time (a week in our case) and as a *fail* if it does not detect any fault.

**Function 3.** \( FDC \) is measured using the following function.

\[ FDC_{sk} = \frac{\sum_{i=1}^{nt_{sk}} SucR_{tc_i}}{nt_{sk}} \]
\( n_{s_k} \) is the number of prioritized test cases for the solution \( s_k \), where \( 1 \leq n_{s_k} \leq nt \). \( SucR_{tc_i} \) is the successful rate of execution for the prioritized test case \( tc_i \), which can be measured as below.

\[
SucR_{tc_i} = \frac{\text{NumSuc}_{tc_i}}{\text{NumSuc}_{tc_i} + \text{NumFail}_{tc_i}}
\]

\( \text{NumSuc}_{tc_i} \) is the number of success for test case \( tc_i \) and \( \text{NumFail}_{tc_i} \) is the number of fail for test case \( tc_i \). For instance, a test case is usually executed 1000 times per week in Cisco. So if it executes successfully for 800 times, the \( SucR \) is \( 800/1000 = 0.8 \).

Notice that \( FDC \) value also ranges from 0 to 1 and a higher value represents better fault detection capability.

### 4.2. Fitness Function

According to the afore-defined cost/effectiveness measures, the fitness function can be defined to take all these measures into account. Notice that the values obtained by different objective functions may not be comparable. For instance, \( OEC \) can be 100 minutes while \( PE \) ranges from 0 to 1. Therefore, in order to evaluate fitness function, the values obtained by each function for cost/effectiveness measures will first be normalized using the following normalization function such that they can have the same magnitude from 0 to 1 [14, 15].

\[
nor(F(x)) = \frac{F(x) - F_{\text{min}}}{F_{\text{max}} - F_{\text{min}}}
\]

\( F \) is a specific function related with a particular cost/effectiveness measure as we defined in Section 4.1 (e.g., \( OEC \)). \( F_{\text{min}} \) is the minimum value obtained by the function while \( F_{\text{max}} \) represents the maximum value achieved by the function.

Therefore, the fitness function is defined as below.

\[
Fitness_P = 1 - w_1 * (1 - nor(OEC)) - w_2 * nor(PE) - w_3 * nor(FPC) - w_4 * nor(FDC)
\]

The fitness function aims at solving a multi-objective minimization problem such that a lower value of fitness function represents better performance of prioritization for a solution. More specifically, lower values show higher values for \( PE, FPC, FDC \) and lower value for \( OEC \). In addition, \( w_1, w_2, w_3 \) and \( w_4 \) are weights assigned to \( OEC, PE, FPC \) and \( FDC \), respectively, and they need to satisfy the constraint: \( \sum_{i=1}^{4} w_i = 1 \). Notice that the weights can be pre-defined by test engineers showing the priority for different objectives (i.e., cost/effectiveness measures). In our case, based on the discussions with test engineers.
at Cisco, we respect equivalent priority for each objective, i.e., \( w_1 = w_2 = w_3 = w_4 = 0.25 \).

5. Industrial Case Study and Artificial Problems

An industrial case study of VCS product line (Saturn) is first used to evaluate our fitness function. More specially, using our model-based test selection methodology as presented in [4], 257 test cases are obtained for testing 53 relevant features belonging to a feature model built for Saturn (more details about this feature model can be consulted in [4]). Each feature can be tested by at least one test case (usually more than one), and one test case can be executed for testing one or more features. In addition, each test case owns a success rate for execution \( SucR_{tc_i} \) ranging from 50% to 100%. Moreover, there are 59 available test resources used to set up the test environment (e.g., correct software and subnet) for executing such 257 test cases. Notice that each test resource can be allocated for executing one or more test cases and execution of each test case requires one or more test resources.

Moreover, we defined 500 artificial problems to empirically evaluate whether the fitness function defined in Section 4.2 can address our test prioritization problem even with a large number of features and various test resources. To tackle such scalability issue, we first created a feature repository including 600 features, a test case repository of 3000 test cases and a test resource repository of 1000 available test resources. Based on the discussions with test engineers at Cisco, we simulated the relations at random between test cases and features, and between test cases and test resources. More specifically, one feature can be tested by 1-5 test cases whereas one test case can be used for testing 1-5 features. In addition, executing one test case requires allocating 1-5 test resources while one test resource can be used for executing 1-5 test cases. Moreover, for each test case \( tc_i \), one key attribute is assigned randomly (inspired by our industrial case study but with expansion for generality), i.e., \( SucR_{tc_i} \) ranges from 0% to 100%. Therefore, we created artificial problems with the increasing number of features and test resources, i.e., we used a range of 10 to 500 with an increment of 10 for features number and the given available test resources can be from 50 to 500 with an increment of 50. So that \( 50 \times 10 = 500 \) artificial problems were obtained in this way. Based on that, each artificial problem consists of a set of features associated with a specific set of test cases (requires to be prioritized) and a certain number of available test resources for allocating. The goal is to cost-effectively
prioritize the test suite while meeting the given available test resources using the selected algorithms.

6. Empirical Evaluation

In this section, we present our empirical study that we conducted to evaluate our defined fitness function along with selected search algorithms.

6.1. Experiment Design

Recall that for cost-effective test prioritization, we defined four cost/effectiveness measures (i.e., OEC, PE, FPC and FDC) and fitness function based on the domain expertise and discussions with test engineers at Cisco, which are used to guide the search for finding an optimal solution (i.e., optimal order of test cases within the given test resource budget). We want to address the following research questions by conducting the experiments:

**RQ1:** Are the selected search algorithms cost-effective to solve our prioritization problem as compared with RS?

**RQ2:** Which search algorithm can achieve the best performance in terms of solving the prioritization problem?

**RQ3:** How does the increment of the number of features and test resources impact the performance of the selected search algorithms (i.e., scalability of the search algorithms)?

More specially, for industrial case study, our experiment first compares the selected three search algorithms (i.e., (1+1) EA, GA and AVM) with RS to assess whether search algorithms can significantly help to improve the performance of prioritization in terms of defined objectives (RQ1). Moreover, we further compare each pair of the selected search algorithms to see which one can achieve the best performance (RQ2).

For artificial problems, we conduct the same experiment to address RQ1 and RQ2 with respect to a large number of features and test resources for assessing the performance and scalability of the selected search algorithms. In addition, to address RQ3, Mean Fitness Value for each problem \(MVF_{i,j}\) is defined to measure the mean fitness value for a certain number of runs \(nr\) as below (in our case, \(nr = 100\)), where \(i\) is the feature number and \(j\) refers to the test resource number \((10 \leq i \leq 500\) with an increment of 10 and \(50 \leq j \leq 500\) with an increment of 50). \(Fitness_{Pr}\) is the obtained fitness value after the \(r_{th}\) run.
\[ MFV_{i,j} = \frac{\sum_{r=1}^{nr} Fitness_{Pr}}{nr} \]

Moreover, Mean Fitness Value for Feature \((MFV_{F_i})\) and Mean Fitness Value for Test Resource \((MFV_{TR_j})\) are further defined to measure the mean fitness function in a given number of features or test resources budget as below. In this way, a specific \(MFV_F\) and \(MFV_{TR}\) can be calculated for each number of features and test resources and RQ3 can be addressed using statistical analysis.

\[ MFV_{F_i} = \frac{\sum_{j=1}^{500} MFV_{ij}}{10}, \quad MFV_{TR_j} = \frac{\sum_{i=1}^{500} MFV_{ij}}{50} \]

### 6.2. Statistical Tests

To analyze the obtained result, the Vargha and Delaney statistics and Mann-Whitney U test are used based on the guidelines for reporting statistical tests for randomized algorithms [16]. The Vargha and Delaney statistics is used to calculate \( \hat{A}_{12} \), which is a non-parametric effect size measure. In our context, \( \hat{A}_{12} \) is used to compare the probability of yielding higher values for the values of fitness function for two algorithms \(A\) and \(B\) with different weight strategies. If \( \hat{A}_{12} \) is 0.5, the two algorithms have equal performance. If \( \hat{A}_{12} \) is greater than 0.5, \(A\) has higher chance to obtain better solutions than \(B\). The Mann-Whitney U test is used to calculate \(p\)-value for deciding whether there is a significant difference between \(A\) and \(B\). We choose the significance level of 0.05, i.e., there is a significant difference if \(p\)-value is less than 0.05. Based on the above description, we define that algorithm \(A\) has better performance than algorithm \(B\), if the \( \hat{A}_{12} \) value is greater than 0.5 and such better performance is significant if \(p\)-value is less than 0.05.

To address RQ3, we choose the Spearman’s rank correlation coefficient \(\rho\) to measure the relations between the \(MFV_F\) and \(MFV_{TR}\) of the algorithms and different number of features and test cases [17]. The value of \(\rho\) ranges from -1 to 1 and can be interpreted as follows: 1) There is a positive correlation if \(\rho\) is greater than 0 and a strong positive correlation if \(\rho\) is close to 1; 2) A negative correlation when \(\rho\) is less than 0 and a value close to -1 shows a strong negative correlation; 3) A \(\rho\) close to 0 shows that there is no correlation between the two sets of data. Moreover, we also report significance of correlation using \(Prob>|\rho|\), a value lower than 0.05 means that the correlation is statistically significant.
6.3. Parameter Settings and Experiment Execution

For all the search algorithms, the maximum number of fitness evaluation is set as 5000, i.e., the process of running algorithms will be terminated if the fitness function has been evaluated by 5000 times. Moreover, the algorithm running will be also terminated if the available test resources are fully allocated. As long as one of the two conditions is satisfied, the optimal solution will be collected.

In addition, for GA and (1+1) EA, the mutation of a variable is done with the standard probability $1/n$, where $n$ is the number of variables. We used a standard one-point crossover with a rate of 0.75 for GA and set the size of population as 100. RS was used as the comparison baseline to assess the difficulty of the problems [7]. According to the guidelines in [16], each algorithm is run for 100 times to account for random variation inherited in search algorithms. Our experiments were run on one PC with Intel Core i7 2.3GHz with 4 GB of RAM, and Windows 7 operating system.

6.4. Results and Analysis

This section presents the results and analysis for our industrial case study and the designed artificial problem followed by an overall discussion based on the obtained results.

6.4.1. Results and Analysis for Industrial Case Study

This section presents the results and analysis for industrial case study (prioritizing 257 test cases for testing 53 features within a given 59 available test resources). For the research questions (RQ1-RQ2), we performed Vargha and Delaney statistics and Mann-Whitney U test at a significant level of 0.05.

Table 1 summarizes the results for comparing each pair of the selected search algorithms (including RS) using the Vargha and Delaney statistics and Whitney U test for the industrial case study. We answer the RQ1 and RQ2 as below.

RQ1: Based on the results (Table 1), we concluded that all the three selected search algorithms (i.e., AVM, GA and (1+1) EA) significantly outperformed RS in terms of finding an optimal solution for test prioritization problem within the given test resources since all the $\hat{A}_{12}$ values are greater than 0.5 and all the $p$-values are less than 0.05. That shows the selected search algorithms are cost-effective as compared with RS in terms of solving our test prioritization problem.
RQ2: According the obtained results (Table 1), the performance of (1+1) EA is significantly better than AVM and GA since all the $\hat{A}_{12}$ values are less than 0.5 and all the $p$-values are less than 0.05. Moreover, AVM has significantly worse performance than GA since the $\hat{A}_{12}$ value is less than 0.5 and the $p$-value is also less than 0.05. That implies that (1+1) EA can achieve the best performance among all the selected search algorithms.

| Table 1. Results for RQ1 and RQ2 using the Vargha and Delaney Statistics and Whitney U Test* |
|-----------------------------------------------|-------|-------|
| RQ    | PA       | A    | p     |
| RQ1   | AVM vs. RS | 0.72 | 0.02  |
|       | GA vs. RS | 0.75 | 0.01  |
|       | (1+1) EA vs. RS | 0.84 | 0.01  |
| RQ2   | AVM vs. GA | 0.35 | 0.03  |
|       | AVM vs. (1+1) | 0.28 | 0.01  |
|       | GA vs. (1+1) EA | 0.31 | 0.01  |

*PA: pair of algorithm, A: $\hat{A}_{12}$, p: $p$-value. All $p$-values less than 0.05 are identified as bold.

In addition, we make the relevant plots and report the average values for each core objective (i.e., the number of prioritized test cases, feature pairwise coverage (FPC) and fault detection capability (FDC) covered by the prioritized test cases). For all the plots (Fig. 3 to Fig. 5) the x-axis refers to each running of the search algorithm (in our case, each algorithm is run for 100 times) and the y-axis shows the objective values achieved by the search algorithms. We describe them in detail as below.

Fig. 3. Number of prioritized test cases

For the Number of Prioritized Test Cases: Fig. 3 shows the number of prioritized test cases obtained by each algorithm (notice that the total number of test cases is 257, which requires to be prioritized within 59 available test resources). We can conclude that (1+1) EA achieved the best performance, i.e., (1+1) EA can prioritize highest number of test cases (on average 195.8 test cases are prioritized). GA can prioritize on average 160.4 test cases, which outperformed AVM and RS. The performance of AVM (on average 127.6 test
cases) is slightly better than RS (on average 115.8 test cases). However, from the Figure 3, we can also observe that the performance of RS does not stay stable, i.e., sometimes, only few test cases such as 78 can be prioritized but more than 150 test cases can be prioritized the other time.

For Feature Pairwise Coverage (FPC): The values for FPC achieved by the selected search algorithms are shown in Figure 4. Based on the results, we can also conclude that (1+1) EA outperformed the other algorithms (i.e., GA, AVM and RS) with on average 0.77 for FPC. The performances for GA and AVM are close to each other with on average 0.68 and 0.65 for FPC, respectively. Similarly, RS still performed the worst with jumpy values of FPC (on average 0.44).

![Fig. 4. Feature pairwise coverage (FPC)](image.png)

For Fault Detection Capability (FDC): Figure 5 illustrates the performance of the search algorithms with respect to FDC. Again, (1+1) EA achieved the best value for FDC (on average 0.80) followed by GA and AVM with the FDC values of 0.72 and 0.67 on
average, respectively. RS performed worse than all the selected three search algorithms at the same time is performance did not stay stable (its FDC values is 0.56 on average).

Concluding Remarks for RQ1 and RQ2: Based on the results obtained from the industrial case study, RQ1 and RQ2 can be answered as follows: all the selected algorithms are cost-effective as compared with RS in terms of each core objective for our test prioritization problem. Moreover, (1+1) EA significantly outperformed GA, while the performance of GA was significantly better than AVM. In summary, (1+1) EA together with the defined fitness function achieved the best performance with respect to finding an optimal solution for the test prioritization problem within a given test resource budget.

6.4.2. Results and Analysis for Artificial Problems

In this section, we discuss the results and analysis for 500 designed artificial problems, which are evaluated based on the defined fitness function (Section 4). Notice that each problem was repeated for 100 times to account for random variations. To answer RQ3, we calculated the spearman’s rank correlation using mean fitness values $MFV_F$ and $MFV_TR$ (Section 6.1) for each algorithm with the growth of features and available test resources. The detailed results and analysis are discussed in detail as below.

6.4.2.1. Results and Analysis for RQ1 and RQ2

To address RQ1 and RQ2, Table 2 summarizes the results for comparing the selected algorithms with RS and each pair of the selected search algorithms using the Vargha and Delaney statistics and Whitney U test for the 500 artificial problems. Two numbers are shown in each cell of the table split by a slash. The first number in the column $A>B$ shows the number of problems out of 500 for which an algorithm $A$ has better performance than $B$ ($\hat{A}_{12} > 0.5$), $A<B$ means vice versa ($\hat{A}_{12} < 0.5$), and $A=B$ means the number of problems for which $A$ has equivalent performance as $B$ ($\hat{A}_{12} = 0.5$). The second number after “/” in the column $A>B$ means the number of problems out of 500 for which $A$ has significantly better performance than $B$ ($\hat{A}_{12} > 0.5$ && $p < 0.05$), $A<B$ means vice versa ($\hat{A}_{12} < 0.5$ && $p < 0.05$), and $A=B$ means the number of problems for which there is no significant difference in performance between $A$ and $B$ ($p \geq 0.05$). We concluded the results as below.

AVM vs. RS: There were 77.4% (387/500) problems, for which AVM achieved better performance than RS and for 68.2% problems, AVM significantly outperformed RS. Moreover, for 12.4% problems, there were no significant differences between AVM and
Table 2. Results for RQ1 and RQ2 using the Vargha and Delaney Statistics and Whitney U Test for artificial problems

<table>
<thead>
<tr>
<th>RQ</th>
<th>Pair of Algorithms</th>
<th>A&gt;B</th>
<th>A&lt;B</th>
<th>A=B</th>
</tr>
</thead>
<tbody>
<tr>
<td>RQ1</td>
<td>AVM vs. RS</td>
<td>387/341</td>
<td>113/97</td>
<td>0/62</td>
</tr>
<tr>
<td></td>
<td>GA vs. RS</td>
<td>425/399</td>
<td>75/51</td>
<td>0/50</td>
</tr>
<tr>
<td></td>
<td>(1+1) EA vs. RS</td>
<td>486/472</td>
<td>14/0</td>
<td>0/28</td>
</tr>
<tr>
<td>RQ2</td>
<td>AVM vs. GA</td>
<td>199/164</td>
<td>301/265</td>
<td>0/71</td>
</tr>
<tr>
<td></td>
<td>AVM vs. (1+1) EA</td>
<td>105/87</td>
<td>395/370</td>
<td>0/43</td>
</tr>
<tr>
<td></td>
<td>GA vs. (1+1) EA</td>
<td>137/109</td>
<td>363/327</td>
<td>0/64</td>
</tr>
</tbody>
</table>

GA vs. RS: GA outperformed RS for 85% problems while for 79.8% problems the results were statistically significant. In addition, there were no significant differences for 10% problems between GA and RS (Table 2).

(1+1) EA vs. RS: With respect to (1+1) EA, it performed better than RS for 97.2% problems. 94.6% out of these problems, (1+1) EA was significantly better than RS. Only for 5.8% problems, there were no significant differences between (1+1) EA and RS (Table 2).

AVM vs. GA: When comparing AVM with GA, for 39.8% problems, AVM outperformed GA but only for 32.8% problems, the results were significant. On contrary, GA performed better than AVM for 60.2% problems and 53% out of these problems, the performance of GA was significantly better than AVM. There were no significant differences between AVM and GA for 14.2% (Table 2).

AVM vs. (1+1) EA: There were 21% problems that showed AVM performed better than (1+1) EA and only for 17.4% problems, the results were statistical significant. On the other side, for 79% problems, (1+1) EA performed better than AVM and for 74% problems, (1+1) EA performed significantly better than AVM. For 8.6% problems, there were no significant differences between AVM and (1+1) EA (Table 2).

GA vs. (1+1) EA: Similarly, for 27.4% problems, GA was better than (1+1) EA, but for 21.8% problems, GA was significantly better than (1+1) EA. Moreover, GA was worse than (1+1) EA for 72.6% problems and for 65.4% out of these problems, GA was significantly worse than (1+1) EA. There were no significant differences between them for 12.8% problems (Table 2).

Concluding Remarks for RQ1 and RQ2: Based on the obtained results, we can answer RQ1 and RQ 2 as follows: for the designed 500 artificial problems, AVM, GA and (1+1)
EA along with the defined fitness function significantly outperformed than RS in term of finding an optimal solution for our prioritization problem. In addition, the performance of (1+1) EA was significantly better than GA while GA performed significantly better than AVM. In summary, (1+1) EA achieved the best performance in most of the artificial problems, which are consistent as what we observed from the results of industrial case study.

6.4.2.2. Results and Analysis for RQ3

The results for Spearman’s correlation analysis ($\rho$) are shown in Table 3 between mean fitness value for feature ($MFV_F$) with the increasing number of features from 10 to 500 with an increment of 10 and mean fitness value for test resources ($MFV_TR$) with the growth of test resources from 50 to 500 with an increment of 50 for the 500 artificial problems. Recall that a lower value of $MFV_F$ or $MFV_TR$ represents a better performance of an algorithm with a specific weight strategy. We concluded the results as below.

Table 3. Spearman’s correlation analysis for the algorithms with the increasing number of features and test resources

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Increasing Features</th>
<th>Increasing Test Resources</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Spearman $\rho$</td>
<td>Prob&gt;</td>
</tr>
<tr>
<td>AVM</td>
<td>-0.10</td>
<td>0.3865</td>
</tr>
<tr>
<td>GA</td>
<td>-0.08</td>
<td>0.2013</td>
</tr>
<tr>
<td>(1+1) EA</td>
<td>-0.12</td>
<td>0.1347</td>
</tr>
<tr>
<td>RS</td>
<td>0.11</td>
<td>0.5129</td>
</tr>
</tbody>
</table>

Increasing Number of Features (Increasing Features column): Among all the search algorithms, we observed that the $MFV_F$ values obtained by AVM, GA and (1+1) EA decreased but not significantly with the growth of feature number since all the $\rho$ values were less than 0 (for instance, the $\rho$ value for (1+1) EA is -0.12) and the values of $Prob>|\rho|$ were all greater than 0.05 (Table 3), i.e., the performance of AVM, GA and (1+1) EA improved but not significantly as the increasing number of features. The performance of RS decreased with the growth of the number of features but the results were also not statistically significant since the $\rho$ value was greater than 0 and the value of $Prob>|\rho|$ was also greater than 0.05 (Table 3).

Increasing Number of Test Resources (Increasing Test Resources column): Similarly, the $MFV_TR$ values achieved by AVM, GA and (1+1) EA decreased but not significantly as the increasing number of test resources since all the $\rho$ values were less than 0 and the
values of $\text{Prob} > |\rho|$ were all more than 0.05 (Table 3), i.e., the performance of AVM, (1+1) EA and GA increased (but not significantly) with the growth of test resources. On the contrary, RS performed worse (also not significantly) as the increasing of test resources since the $\rho$ value was greater than 0 and the value of $\text{Prob} > |\rho|$ was also greater than 0.05 (Table 3).

**Concluding Remarks:** AVM, GA and (1+1) EA performed better as the increasing number of features and test resources but such better performance was not statistical significant. We conclude that the performance of selected search algorithms is not significantly influenced by the increasing number of features and test resources.

### 6.4.3. Overall Discussion

We provide an overall discussion based on the results for the industrial case study and 500 artificial problems.

First, from the industrial case study, we observed that the performance of AVM, GA and (1+1) EA was significantly better than RS with for our test prioritization problem (RQ1). Moreover, (1+1) EA along with the defined fitness function achieved the best performance as compared with AVM and GA. Followed by (1+1) EA, GA significantly outperformed AVM (RQ2). This is due to the reason that GA and (1+1) EA are global search algorithms and both manage to find global optimal solutions as compared to AVM, a local search algorithm. Notice that the performance of (1+1) EA is significantly better than GA and this may be due to the reason that (1+1) EA uses only mutation for exploring the search space as compared to GA which uses both mutation and crossover for exploration and exploitation of search space respectively requiring more generations to find a global optimal solution. By increasing the number of fitness evaluations (5000 in the current experiment settings), we expect that the performance of GA can be improved, which requires more empirical evaluations.

For the experiments based on the 500 artificial problems, the results were consistent with our industrial case study (RQ1-RQ2). When we looked at the impact of varying number of features (10-500) and test resources (50-500) on the performance of each algorithm along with our defined fitness function for the 500 artificial problems (RQ3), we observed that the performance of AVM, GA and (1+1) EA was not significantly influenced with the increasing number of features and test resources. That implies that even with more complicated problems (a large number of features and test resources), the performance of
the selected search algorithms can still stay stable in terms of finding an optimal solution for our test prioritization problem. This can be explained that search algorithms (both global and local search algorithms) can scale with the increased complexity and manage to explore the search space in terms of finding an optimal solution for our test prioritization problem.

In summary, (1+1) EA achieved the best performance in our experiments and thus it is suitable for test prioritization problem along with the defined fitness function in our industrial context.

6.5. Threats to Validity

A prominent construct validity threat is related to the measure used to compare the various algorithms and to avoid such threat we used the measured fitness value, which is comparable across all selected search algorithms. Another common construct threat to validity is the use of termination criterion for the search. In our experiments, we used number of fitness evaluations and available test resource budget comparable across all the algorithms.

When using search algorithms, parameter settings may affect the performance of the algorithms (internal validity). In this direction, we used default parameter settings for all the algorithms and these settings have demonstrated promising results [18].

A common conclusion validity threat is due to random variation inherited in search algorithms and thus we repeated our experiments 100 times to reduce the probability that the results were obtained by chance. Moreover, we used appropriate statistical tests for analyzing the data, i.e., Vargha and Delaney and Mann-Whitney U test based on guidelines proposed in [16]. Finally, Spearman’s rank correlation coefficient is used to measure the potential impact on the performance of selected search algorithms with the increase in the number of features and test resources since it is mainly used for non-parametric correlation coefficient measure and suits our objective [17, 19].

Generalization to new case studies is required to increase the confidence on the results (external validity) and we conducted an empirical evaluation using 500 artificial problems besides an industrial case study and obtained consistent results.

7. Related Works
Test prioritization is an attractive topic especially in regression testing, which has been studied by a large number of researchers since the last decades [20]. The main objective for such prioritization is to seek an optimal order of a given set of test cases, which the benefits can be maximized (i.e., maximizing the effectiveness and minimization the cost) [20]. More specifically, test prioritization can be further classified into several categories such as coverage-based prioritization [21-23], human-based prioritization [24], model-based prioritization [25, 26] and cost-aware prioritization [27, 28]. Our approach falls into cost-aware prioritization, namely cost-effective prioritization within a given budget (e.g., test resources in our case).

A comprehensive review for search-based software engineering (SBSE) is available in [5], where search techniques are well described with applications to a variety of software engineering problems. In particular, Harman listed a set of potential objectives used for multi-objective test optimization in regression testing [29], which have been studied in the existing literature (e.g., execution time and fault detection capability) [20].

A close approach with this paper is related to test prioritization in regression testing [30]. In this work, a two-objective problem (i.e., code coverage and execution time) is converted into a single-objective problem using an arithmetical combination of weights for the fitness function. These two objectives are similar as our cost/effectiveness measures feature pairwise coverage (FPC) and Overall Execution Cost (OEC). However, there are at least three differences as compared with our work: 1) Our context is for software product line, i.e., our defined FPC mainly focuses on the coverage of features pairs from the view of product line rather than code level of the systems (i.e., code coverage defined in [30]); 2) The obtained optimal solution for our test prioritization problem has to satisfy the given budget of test resources, which is not addressed in [30]; and 3) two additional cost/effectiveness measures are defined in our work (i.e., Prioritized Extent (PE) and fault detection capability (FDC)), which are based on an real industrial setting and can be generalized to other product line contexts.

Another similar approach as our work aimed at solving multi-objective test generation for product lines [15]. This approach took three objectives into account (i.e., feature pairwise coverage, the number of products and the testing cost) followed by integrating them as a fitness function. Using a newly designed version of genetic algorithm, an optimal solution can be found that includes a set of generated test cases. As compared with [15],
our motivation is different: we aim at cost-effective prioritization of the existing test cases for testing products rather than generating new test cases. Moreover, we adapted three commonly used search algorithms (e.g., (1+1) EA) to solve our prioritization problem rather than designing a new search algorithm that requires extensively evaluation. In addition, additional cost-effectiveness measures are defined in our work based on our industrial collaboration, i.e., $PE$ and $FDC$, and 500 designed artificial problems are designed to assess the performance and scalability of the three search algorithms along with our defined fitness function (Random search is used as the comparison baseline for assessing the difficulty of the problems).

In one of our published work [8], to solve multi-objective optimization problem, we defined three effectiveness measures (i.e., test minimization percentage ($TMP$), feature pairwise coverage ($FPC$) and fault detection capability ($FDC$)) followed by defining a fitness function. The performance of three weight-based genetic algorithms (e.g., RWGA) was also evaluated. As compared with [8], first, the motivation in this work is different, i.e., we aim at solving multi-objective test prioritization problem within a given budget (e.g., test resources) while test minimization problem usually does not have such pre-defined budget limitations. Second, an additional cost measure (Overall Execution Cost ($OEC$)) and effectiveness measure (Prioritized Extent ($PE$)) were defined, and the fitness function was empirically evaluated with three search algorithms using an industrial case study and 500 artificial problems with increasing complexity.

8. Conclusion and Future Work

Test prioritization is an essential but difficult problem in industry since it always requires considering a tradeoff of several conflicting objectives. Inspired by an industrial application of test prioritization problem at Cisco Systems, Norway, this paper presents a search-based technique for cost-effective prioritization of a given test cases considering a limited budget (test resources). More specifically, one cost measure and three effectiveness measures were formally defined (i.e., Overall Execution Cost ($OEC$), Prioritized Extent ($PE$), Feature Pairwise Coverage ($FPC$) and Fault Detection Capability ($FDC$)), and a fitness function was further defined considering all these four cost/effectiveness measures. We also empirically evaluated three commonly used search algorithms (AVM, GA, (1+1) EA) along with the defined fitness function using an industrial case study and 500 designed artificial problems with increasing complexity.
artificial problems. Notice that Random Search (RS) was used as a baseline for assessing the difficulty of the problems.

The results of industrial case study showed that the selected three search algorithms significantly outperformed the RS and (1+1) EA achieved the significant best performance among the selected search algorithms in terms of finding an optimal solution for the test prioritization problem. From the results of 500 artificial problems, we conclude that the performance of (1+1) EA is also significantly better than the other search algorithms, which is consistent with the results of industrial case study. Moreover, we also observed that the performance of all the selected search algorithms is not significantly influenced by the increasing complexity of problems (i.e., the increasing number of features and test resources), which demonstrates that the search algorithms are scalable and can solve a wider range of problems. All these findings suggest that (1+1) EA along with our defined fitness function can assist test engineers to cost-effectively solve the test prioritization problems of varying complexity.

In the near future, we plan to integrate our solution into the existing practice of Cisco. Moreover, we plan to conduct additional case studies to further strengthen our experimental results and the fitness function with additional cost/effectiveness measures if necessary. We also plan to investigate more multi-objective search techniques (e.g., Pareto-based search algorithms) to assess the performance and scalability along with the defined fitness function in industrial settings.

References
3. Software engineering institute: Hall of Fame. Available at: www.splc.net/fame.html


19. Yue, T., Briand, L.C., and Labiche, Y. Facilitating the transition from use case models to analysis models: Approach and experiments. 2013. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 22(1).


