Application of Search-Based Software Testing in Stress-Testing of Deeply Embedded Components in Integrated Circuits

Thesis submitted for the degree of

Doctor of Philosophy

in

Computer Science

at the University of Leicester

by

Basil Eljuse

Department of Informatics

March 2020

Abstract

Application of Search-Based Software Testing in Stress-Testing of Deeply Embedded Components in Integrated Circuits

Basil Eljuse

Modern Integrated Circuits (ICs) based system design have evolved over time to support the ever demanding complex use-cases. As more functionality is packed into these ICs, there are more deeply embedded components present in these ICs. The performance of these deeply embedded components are critical in ensuring the final system behaviour. By virtue of being deeply embedded they are intrinsically difficult to test - especially in system context. This thesis is focused on the task of 'stress testing' an IC in its operational environment with the goal of identifying any circumstances under which the circuit might suffer from performance issues. It addresses how search-based algorithms provide an automated approach in testing these ICs. Here it establishes that with a suitable search-space representation search-based approaches can be applied in stress-testing of complex ICs. Here it outlines how a simple hill-climbing based search algorithm is successfully used to generate tests in an automated manner. It also studies the application of more advanced search-based algorithms like random-restart hill-climbing and simulated annealing in addressing this testing challenge. As system behaviour of complex ICs can be affected in multiple ways, this research investigates the application of multi-objective algorithms in this context. It shows that a multi-objective algorithm like Strength Pareto Evolutionary Algorithm 2 (SPEA2) is able to generate better stress tests, maximising multiple objectives. The empirical studies performed as part of this thesis are conducted on Arm® Cache Coherent Interconnect (CCI), which provides cache coherency and interconnect functionality on modern Arm based systems. It concludes that search-based software testing approaches are indeed suitable for addressing testability challenges posed by modern ICs especially at system level and provide ways to generate better stress tests.

Dedication

I dedicate this thesis to my loving God for all the blessings thou showered upon me. I also dedicate this to my parents, wife, son, daughter and everyone who supported me through this journey. Nothing would have been possible without their unwavering support, unlimited patience and enduring love, for which I am eternally thankful.

Acknowledgements

I would like to start expressing my sincere gratitude to my guide Dr. Neil Walkinshaw for all the support and guidance during my research. I would also like to thank Dr. Jose Miguel Rojas Siles and Prof. Reiko Heckel for all the support during my studies. I am sincerely grateful to Dr. Fer-Jan de Vries, the postgraduate tutor at the Informatics department (University of Leicester), who helped me through the intermediate stages of my study.

I would like to thank Arm Ltd for the support extended to me during this study, while being employed there. In particular, I would like to thank Guillaume Letellier and James King for the internal reviews they conducted in relation to this study. I would also like to thank all my line managers during this period, the HR, the Intellectual Property team and the Legal team at Arm for their support during my study period. I want to express a special note of thanks to a couple of my colleagues at Arm - Sudeep Holla and Punit Aggarwal - who helped me with some of the kernel driver details, which were crucial during the experimentation phases of my research.

The earliest support for my higher studies came as reference letters for my admission from my teachers Dr. Chitra Prasad and Dr. Ansamma John. I would like to express my sincere gratitude to both of them. Many of my friends and family showed keen interest in tracking the progress I was making and thanks a lot for all the encouragements from them - it meant a lot to me.

Contents

List of Tables

List of Figures

Acronyms

ADB Android Debug Bridge.

AMBA Advanced Microcontroller Bus Architecture.

APFD Average Percentage Faults Detected.

AUC Area Under the Curve.

AXI Advanced eXtensible Interface.

CCI Cache Coherent Interconnect.

CCIX Cache Coherent Interconnect for Accelerators.

CPU Central Processing Unit.

DfT Design for Testability.

E2E End-2-End.

EDA Electronic Design Automation.

FPGA Field Programmable Gate Array.

GP Genetic Programming.

GPU Graphics Processing Unit.

HC Simple Hill-Climbing.

HIL Hardware-In-the-Loop.

IC Integrated Circuit.

IO Input-Output.

IP Intellectual Property.

L1 Level1.

L2 Level2.

L3 Level3.

MCM Memory Consistency Model.

NSGA2 Non-dominate Sorting Genetic Algorithm 2.

PCB Printed Circuit Board.

PESA Pareto Envelop-based Selection Algorithm.

PMU Performance Monitoring Unit.

RAD Rapid Application Development.

RRHC Random-Restart Hill-Climbing.

SA Simulated Annealing.

SBST Search-Based Software Testing.

SMC Static Memory Controller.

SMD Standardised Mean Difference.

SoC System-on-Chip.

SPEA Strength Pareto Evolutionary Algorithm.

SPEA2 Strength Pareto Evolutionary Algorithm 2.

TC2 Test Chip2.

UART Universal Asynchronous Receiver Transmitter.

USB Universal Serial Bus.

UUT Unit Under Test.

Chapter 1

Introduction

1.1 Introduction

An IC is an electronic circuit design fabricated on a semiconductor material like silicon. ICs form the basic building blocks with which complex electronic systems are designed. Modern ICs offer greater functional capabilities and typically are designed in a modular fashion. It normally comprises of various individual subsystems often interconnected with core components that are deeply embedded in nature. Even though early IC designs had only limited capabilities, modern ICs do pack complex functionality in it that is needed to support the computing demands of modern era.

Testing forms an integral part of the engineering process followed in the development of electronic systems. In the simplest form, testing can be defined as the process of checking whether a given Unit Under Test (UUT) is performing its intended functionality and to identify any potential malfunctions. In case of the engineering process involved in developing ICs, a considerable amount of resources are spent on the testing related activities. This signifies the importance of testing in general.

Computing is becoming more pervasive and is present everywhere. On one hand, it can be found in a tiny sensor at many industrial premises performing instantaneous measurements, while on the other it can be in a supercomputer that performs exascale computing as part of some complex modelling and simulation activity. It can be found in a wearable computing device, a self-driving car or in the cloud infrastructure which powers the modern internet, just to list a few. This wide range of technology is made possible by the advancements in electronic system designs that are built on ICs. So testing ICs is incredibly important considering the extent of impact it has now-a-days. Insufficient levels of testing of ICs could lead to potential malfunctions which can have far-reaching consequences. These malfunctions could manifest in ways that can affect precise functional correctness of computa-

tions performed - as it happened in case of Intel's Pentium FDIV (floating point division) bug [Ede97] that resulted in a rare occurrence of incorrect computation during operation. There are other occasions where malfunctions could be fatal - as it happened in case of Intel's F00F bug [Wik], where once triggered it could result in the system to be locked up and needing a hard reboot to revive from that locked-up state. The recent security vulnerabilities around Spectre [Koc+19] and Meltdown [Lip+18] which undermines data privacy shows the extend of damage such malfunctions in ICs or poor design choices can lead to. This affected a wide range of processor architectures - Intel, Arm, AMD etc. These issues elevate the role of testing in modern IC development to levels higher than ever before.

Testing of ICs is demanding and is particularly difficult at a system level. Many of the IC components are considered as 'deeply embedded' and when tested from system level they are under multiple layers of software and hardware components. An IC component is characterised as 'deeply embedded' based on the fact that a hardware component could be performing a highly specialised functionality with very little or no direct interfaces to higher level software in the system. This absence of direct interfaces makes the testing of these components difficult at system level. This results in a key testing challenge for complex systems that are based on these IC designs. One of the testing challenges in general with embedded components is the difficulty of understanding the right context in which it is integrated in a system. In this particular research we focus on testing of an embedded component in its fixed system context. However this same component could be integrated in different configurations based on the possible variations in the final system design. The levels of increasing functional complexity and the high degree of configurability of these hardware components exacerbates the testing challenge further and addressing this forms the key focus of this research.

System testing has a particular significance in IC testing. Due to the nature of development of hardware and software components in IC development (where these two components get developed in parallel), system testing is the earliest opportunity where a full software stack can be tested on a final hardware. This often uncovers complex system integration defects, system performance issues due to incorrect configurations or sub-optimal tuning and also latent defects that can affect the overall system behaviour. Hence this presents us with the last opportunity to uncover and fix such issues which can be exposed only in a final system testing context. With the significance of system level testing on IC development, this research addresses the testing challenges posed specifically by deeply embedded IC components with full End-2-End (E2E) software stack. Even though these deeply embedded IC components are tested in isolation for functional correctness, considering their critical nature in determining the final system behaviour, this research looks at improving

the system test capabilities.

1.2 Specific Contributions

The specific contributions of this research are:

1. The testing of deeply embedded IC components at system level pose testability challenges which makes traditional test approaches less effective. This research investigates the suitability of search-based algorithms to stress test deeply embedded IC components at system level. This research performs empirical studies, as described in Chapters 4, 5 and 6, that focus on stress testing of a deeply embedded IC component from Arm called CCI.

2. A proposed search-space representation and a suitable set of relevant fitness functions to perform stress testing of CCI IC component on Arm platforms as described in Section 3.3 of Chapter 3. This research also describes an experimentation framework outlined in Section 3.4 which is used to implement the proposed test approach for conducting the empirical studies.

3. Assessment of the relative benefits of applying a set of distinct search-based algorithms to this testing problem. This research conducts empirical studies that assess the application of both single objective and multi-objective search-based algorithms for CCI testing as described in Chapter 4 and 6 respectively. It also looks at addressing local maxima problem observed with the application of local search algorithms with the use of more advanced algorithms adept to overcome a fitness landscape that have multiple local maximas as seen from Chapter 5.

1.3 Thesis Outline

The main body of this thesis consists of seven chapters. Followed by the introduction in the **Chapter 1 - Introduction** (current chapter), the **Chapter 2 - Background** provides a comprehensive background into this research. This section aim to introduce the general background to IC development and how engineering development practices and testing in particular is applied in that context. We explore the specific details of CCI in a full system context and the associated testing challenges. This chapter details the motivation for this research and provides a clear articulation of the problem statement with a pointer to the possible direction in applying search-based test techniques while seeking a solution for this research problem.

Chapter 3 - Search-Space Model and Experimentation Framework lay down the essential parts of the proposed solution in addressing the IC testing challenge taken up by this research. It captures the details of remodelling the IC testing problem as a search problem. To that effort, this chapter goes into the details of how a suitable state-space model is arrived and those fitness functions that can be used along with the search algorithms. In order to enable us to conduct some empirical studies to address key research questions, an experimentation framework is also developed. This chapter gives details of that experimental setup and how it will be used in the empirical studies which are to follow.

Chapter 4 - Suitability of Search-based Algorithms in IC Testing address the key question of whether search-based approaches are indeed suitable for applying in IC testing. It aims to conduct empirical studies to establish this suitability. It outlines the empirical study conducted on a 32-bit Arm platform with a simpler search-based algorithm, specifically simple hill-climbing algorithm. It drafts the research question to be addressed in this empirical study and describes the evaluation methodology to be followed while conducting this study. This chapter goes on to explain the results gathered from the study and provide conclusions based on the experiment's outcome.

Chapter 5 - Applying Advanced Search-based Algorithms in IC Testing details the evaluation of how some advanced search-based algorithms fare in handling the IC testing challenge. In this chapter we discuss the details of an empirical study conducted on a 64-bit Arm platform with advanced search-base algorithms like random-restart hill-climbing and simulated annealing and assess whether they provide any improvements over a local search algorithm like simple hill-climbing. The research question posed in this chapter focus on addressing the local maxima problem that can be impacting local search algorithms when applied to IC testing and it evaluates whether advanced algorithms can provide any benefits. It details the evaluation methodology followed in addressing the research questions and provide conclusions derived from the empirical study based on the observed results.

Chapter 6 - Improving IC Testing with Multi-Objective Algorithms looks at how to satisfy the real-world demands from IC testing when applying search-based approaches. Here we look beyond the mere suitability of search-based approaches and this chapter goes on to discuss the significance of multi-objective search algorithms in addressing the IC testing challenge. It goes on to detail the empirical study conducted on a 64-bit Arm platform with a multi-objective search algorithm called SPEA2. The research question posed in this chapter evaluate the suitability of multi-objective algorithms in satisfying the testing demands that are closer to the real-world testing needs for deeply embedded IC components. It aims to evaluate the benefits of using multi-objective algorithms over single objective

algorithms in this IC testing problem. It also details the evaluation methodology followed in addressing the research question and provides a conclusion based on the results observed from the empirical study.

Finally the **Chapter 7 - Conclusion and Future Work** looks at the overall conclusions drawn from this research. It derives various conclusions based on the results obtained from the three empirical studies conducted as part of this research and aim to establish the benefits offered by the proposed search-based approach in IC testing. It looks at the relevance of this research beyond the current focus of IC testing from system level. Further looking into the future, this chapter describes the evolving landscape of IC testing and assess the continued relevance of this research with that future in mind. It also looks at further challenges that can be potential followups to this research.

1.4 Contributory Publications

In this section it lists the various research publications done as part of engaging with the wider research community during the period of current research. Materials from these publications are used in various parts of this thesis - as all these publications came about as a direct result of the PhD research detailed in this thesis.

1. Basil Eljuse and Neil Walkinshaw, "A Search Based Approach for Stress-Testing Integrated Circuits", At 8th International Symposium on Search-Based Software Engineering, SSBSE 2016, held in Raleigh, NC, USA, in October 2016. In: Sarro F., Deb K. (eds) Search Based Software Engineering. SSBSE 2016. Lecture Notes in Computer Science, vol 9962. Springer, Cham. (2016)

2. Basil Eljuse and Neil Walkinshaw, "Comparison of Search Based Algorithms for Stress-Testing Integrated Circuits", At 10th International Symposium on Search-Based Software Engineering, SSBSE 2018, held in Montpellier, France, in September 2018. In: Colanzi T., McMinn P. (eds) Search-Based Software Engineering. SSBSE 2018. Lecture Notes in Computer Science, vol 11036. Springer, Cham. (2018)

3. Basil Eljuse and Neil Walkinshaw, "Application of Multi-Objective Search-Based Approach for Stress-Testing Integrated Circuits", STVR Journal - Submission accepted subject to amendments.

Chapter 2

Background

In this chapter we look at the wider context of IC development to understand better the current development practices including those related to testing IC components. We also explain why testing CCI in particular is difficult from a system perspective and outlines how Search-Based Software Testing (SBST) becomes a natural avenue to explore while addressing this testing challenge.

First in Section 2.1 we look at testing in general as practiced within the industry and the typical challenges around testing which forms the wider context for the current research. As this research focus on the testing challenges around IC components, in Section 2.2 we look into the details of how IC development happens in the industry and how the current testing problem manifests in that development workflow. As part of gaining a more in-depth knowledge of CCI component (which is the focus of our research) we look at the details of CCI and how it is currently validated in Section 2.3 and the related testing challenges.

In Section 2.4 we describe why testing CCI from system level is a difficult problem and outline the key issue this research attempts to address. We explain how SBST forms a natural avenue to be investigated as part of this research in Section 2.5 and what options we have to address the problem at hand.

2.1 Testing - A Perspective on Industry Practices

In this section we look at testing discipline in the way it is used within the industry so as to gain a broader understanding of what it pertains to and the key challenges in this domain. Many of the testing challenges addressed in this research do find relevance in the broader context of testing and hence this perspective will be helpful.

Testing is a key discipline employed in the industry as part of hardware and software engineering practices. Testing encompasses a wide variety of activities conducted during the development life-cycle, which often have varying goals defined based on the context in which it is applied. Bertolino [Ber07] explains the use of

testing as part of quality assurance to validate the intended behaviour and identify potential malfunctions. That is a very board but general description of testing discipline. A more formal definition of testing process is described by Staats *et al.* [SWH11] and we will refer back to this formal definition when discussing testing aspects as seen from the industry. First let us inspect this formal definition.

The functional model of testing presented by Staats *et al.* [SWH11] defines a *testing system* as a collection of (P,S,T,O,$corr$,$corr_t$) where:

- S is a set of specifications
- P is a set of programs
- T is a set of tests
- O is a set of oracles
- $corr \subseteq P \times S$
- $corr_t \subseteq T \times P \times S$

Here the predicate $corr$ is defined such that for $p \in P$, $s \in S$, $corr(p, s)$ implies p is correct w.r.t s. The predicate $corr_t$ defines the correctness w.r.t a test $t \in T$, where $corr_t$ holds good if and only if specification s holds for program p when running test t.

In the context of current testing of CCI component within an IC, the specification S corresponds to the Advanced Microcontroller Bus Architecture (AMBA)® Advanced eXtensible Interface (AXI)® protocol, which is used to implement the hardware cache coherency capability of the CCI (See Section 2.3.1 for additional details). The hardware implementation of the CCI logic would correspond to the program P. The data stall cycles events that can be monitored as part of the hardware logic block can be seen as the oracle O. The correctness $corr$ can be one of the 2 aspects. One is the coherency of data managed by the CCI which would result in functional correctness. The second is the effectiveness with which this concurrency of data is ensured by CCI in a timely manner. From the perspective of stress testing of CCI component, the $corr$ shows the correctness in terms of the effectiveness with which the implementation of S by P. As per the specification the occurrence of data stall cycles is a means to manage the stress on the component. Thus a test from T would satisfy $corr_t$ because that test would make the observation of the presence of data stall cycles which indicate the correctness of the implementation of S by P to indicate the CCI state when it is under stress.

Looking into the application of testing principles, there are multiple aspects of testing that presents us with very different challenges overall. Testing often starts

with the definition of a 'test objective', which when defined in a given context drives the emphasis on testing and the process subsequently followed. The test objective for a project can be defined to achieve multiple goals like a) check if the right behaviour is observant from the product b) look for potential malfunctions c) check for conformance to standards d) assess release readiness e) gain user feedback and many more. For a given project it could be any one of the above or a combination of a few or even encompass all of the above mentioned aspects to be part of a test objective. This definition will eventually drive what type of activities gets done as part of testing. When looking this from a more formal definition, *test objective* always aims to check if a program P implements a specification S [SWH11]. Here the challenge had always been how formal the specification is for one to make use of it effectively in the testing process. But none-the-less specification S have a significant role in defining test T. In this research the focus is on non-functional testing rather than functional correctness which would have benefited from the specification being available.

Once the test objective is clear the industry practitioners plan for designing and developing different types of tests. During the test development these different types of tests are typically defined depending on the granularity of the UUT. The granularity of this testing target can be defined as unit testing, component testing, integration testing, system testing and system-of-systems testing. Some test type categorisations are based on the release milestones like Alpha testing, Beta testing, Product release testing etc. There are also different test types defined based on the system qualities (ISO 25010:2011 [ISOa] have a good framework for this) that are targeted by the tests such as functional testing, reliability testing, load testing, stress testing, security testing and usability testing. During the test development, the test practitioners apply different test techniques to identify test cases and two broad categories of test techniques are used in the industry: Black-box test techniques and White-box test techniques. Typically black-box techniques are specification based - there is a test basis available to derive the test cases and the test practitioner sees the system as a black-box. These test basis (body of knowledge used as input for test analysis and design) could be requirements specification or design specifications that forms the input to the test designer other than software code itself. In case of white-box testing the test practitioner has software code as input and the structure based inputs from the code will drive the test design.

One of the key aspects involved in test design is related to 'test oracle' problem which is often signified by the extent to which observability O is accurate in representing the conformance of a program P in implementing its specification S. Barr *et al.* outline the various research efforts to address test oracle problem in their survey [Bar+15].

With the possibility of different types of tests being developed to achieve the very many objectives of the testing identified for a given project, the 'test selection and prioritisation' [Rot+99; Di +15] process becomes a key challenge for the industry practitioners. Yoo *et al.* detailed the challenge of test selection and prioritisation in the context of regression testing in their survey [YH10].

Another important challenge in the testing domain is determining when to stop testing or what amount of testing is enough. In terms of the formal definition the observability O and tests T signify the efficacy of the testing that is feasible. The 'test adequacy' refers to the aspect of testing which determines the stoppage criteria and is typically measured in the form of some coverage measurement. In some cases test adequacy criteria is defined as program-based, where the structural elements of the program in terms of control-flow and data-flow coverage [Hut+94] becomes the measures for test adequacy. There are other types of test adequacy measurements practiced - for example specification-based coverage criteria [HGW04; Raj06], where it looks at the coverage against a given specification. Moreover there are still further different approaches in defining test adequacy where it can be measured using a criteria that depicts it in terms of the fault-finding capabilities of the tests.

In our research we are focused primarily in the automated generation of tests with search-based approaches, thus focusing mostly on the challenges around test generation. While other test challenges might be relevant to the IC testing problem, we want to improve the test generation capability especially at system level.

We have looked so far at the general characteristics and challenges related to testing discipline in particular, which is deployed as part of wider hardware and software engineering development life-cycle. Let us look into more details on these engineering practices themselves where testing discipline gets employed. There are many different product development life-cycles which are adopted for hardware and software development in the industry. In case of hardware development, the engineering practices followed are very much of a phased manner - similar to waterfall, where the development progresses sequentially from one phase to another. This is a true reflection of the physical nature of the deliverable in case of hardware, where a strict ordering of activities is not only desirable but also is very essential. However in case of software there are many additional development life-cycles adopted in the industry beyond Waterfall - such as Agile, Iterative, V-Model, Rapid Application Development (RAD) and Spiral. When it comes to product development that includes both hardware and software, one of the prominent approaches adopted in industry is the hardware-software co-design development process. Some industry sectors like automotive which have to address stringent functional safety requirements (ISO 26262-1:2018 [ISOb]) follow an overlapped V-Model. So you can see that testing discipline is employed in a wide range of engineering process within the

industry and most challenges related to testing are common and cut across these engineering practices.

As we have looked at a general overview of testing discipline in general and the various engineering processes that employs this discipline, let us look into some specific details of the IC development workflow in the next section.

2.2 IC Development: Understanding the Testing Context

First let us gather some detailed context on modern IC development process which was briefly mentioned in Chapter 1. We specifically seek to understand the engineering development life-cycle followed during IC development in order to understand not only the nature of development followed here, but also to better appreciate the inherent challenges in terms of testing, especially at the system level.

Hardware-Software co-design [Tei12] workflows allow the evolution of hardware and software development to happen in parallel. This is prevalent development model adopted by semiconductor Intellectual Property (IP) industry. There is a considerable overlap in the development timeline of these distinct components. This overlap in development time is a key factor which enables quicker time-to-market that fosters product innovations. Typically the hardware development will follow a more phased approach similar to waterfall, while software might be following Agile or other relevant software development life-cycles.

Hardware-software co-design is the de-facto industry practice in semiconductor industry sector and its popularity is underpinned by a range of factors that enables and equips the hardware and software to be developed in parallel. One of the main factors that enables this parallel development to happen is the availability of extensive tooling capabilities in this area. The evolution of Electronic Design Automation (EDA) [Rob15] facilitated the use of hardware-software co-design principles for IC development. The advancements in EDA tools massively improved multiple aspects of IC design like circuit and system design, timing analysis, synthesis, formal verification and many more.

A typical scenario where a system design happens in a hardware-software co-design workflow is outlined in Figure 2.1. In the initial phase, the specifications for the system design happens. This would include the engineering specification which provides details of the various components of the overall system, its outline and layout. Based on this common specification both the hardware and software design and development streams would start evolving in parallel. The hardware development workflow progresses through synthesis, analysis and verification phases.

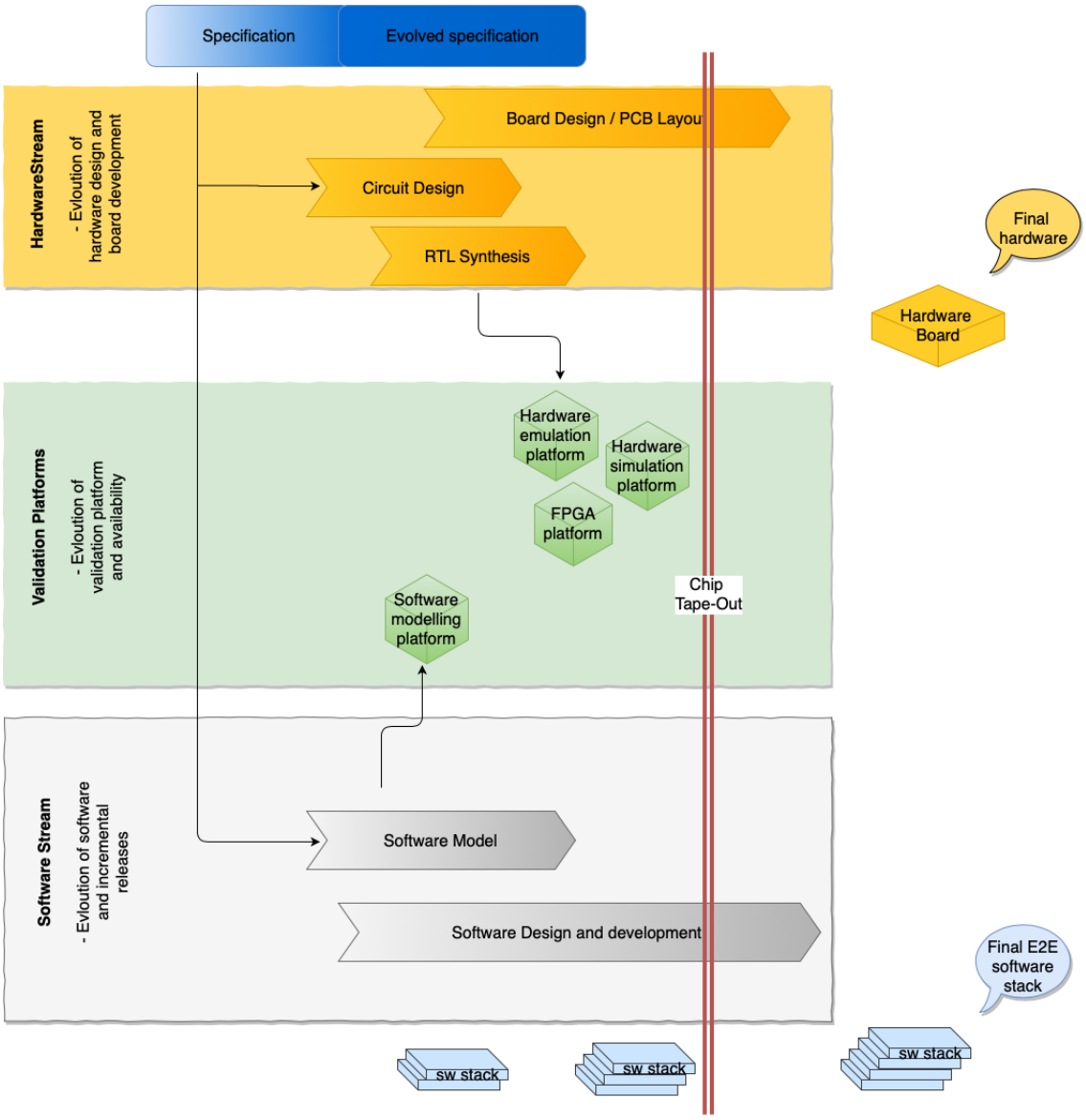

Figure 2.1: Hardware-Software Co-Design: A typical workflow

In parallel, the software development workflow progresses through the design, code development, integration and testing phases. These parallel streams converge at a point where the final hardware is available and the E2E software stack meant for that hardware is integrated as a complete system.

In the IC development workflow tape-out is a key intermediate milestone, where the IC is fabricated into a physical chip. Prior to the tape-out of the IC the output of the hardware workflows will undergo multiple stages of verification and validation. Here it uses EDA tools-based formal verification methodologies, hardware emulation and simulation platforms [Hua+11], Field Programmable Gate Array (FPGA) platform based testing [Hua+11] to perform the validation and verification of the IC. In case of the software stream the earliest platform available to perform validation are software models [CNR13] implementing the specification. As emulation and

simulation platforms become available from the hardware stream, various software deliverables are also validated on these. These validation platforms have varying degree of accuracy in terms of representing the characteristics of the target hardware but none-the-less provides crucial early opportunities to validated the incremental versions of software's behaviour on the evolving hardware. Typically one would find software models, cycle-accurate software models, FPGAs, test-benches based on hardware emulators and power-aware hardware simulators as the early validation platforms made available for software validation. But none of these early testing truly enables a fully integrated system to be analysed for the final system behaviour. This remains a key goal yet to be fulfilled as the hardware and software development streams converge when the final hardware and the full E2E software stack is available. Real system behaviour is assessed on the final hardware with the E2E software stack.

Once the IC is taped-out then the system integration happens from a hardware perspective where typically the Printed Circuit Board (PCB) layout and board assembly is done. A number of board-level or platform-level tests are conducted and once ready software bring-up on the new system begins - starting with low-level firmware, progressively integrating the whole E2E software stack on that hardware board. More system level characterisation and bench-marking happens at this stage and final system gets released. This is the earliest opportunity we have for a full product to undergo system testing, where the final hardware and the full E2E software stack can be integrated and tested as a complete system. This also presents us with a key testing problem - only at this very late stage we have some capability to bring-up the full E2E software stack on the real hardware platform and test it from a full system context. This is the premise against which the current research is exploring for better testing approaches, in order to improve the current system testing practices in IC development. The specific aim being to be able to generate better automated stress tests with search-based approaches.

As detailed in this section we learnt about the typical workflow for IC development and the context in which our current research happens. Our aim is to define an improved test methodology to test some deeply embedded components developed as part of this above-mentioned workflow. As the performance of these components are critical in the final system behaviour it becomes important to do suitable system level tests to assess if there are any issues affecting final system performance. We look further into the details of current testing practices related to deeply embedded components in the subsequent section.

2.3 CCI: Details and Testing Challenges

Now that we have understood the context in which IC development happens, let us try to learn more about CCI and what is all about. In this section we look into details of Arm CCI component, which is an essential part of modern Arm based chips, that is used to build a wide range of products including mobile phones, smart TVs, automotive infotainment and low cost infrastructure components. We also look at how testing gets done currently for CCI and the related testing challenges, which forms the key basis for this research's motivation.

2.3.1 Understanding Cache Coherency and Role of CCI in IC Design

A cache memory is a high-speed memory component included in a Central Processing Unit (CPU) designed to speed up memory operations. With the evolution of multi-cluster, multi-core CPU architectures multi-level cache designs also became prevalent, where cache is arranged in multiple levels and CPUs have a requirement to access coherent data whenever they fetch memory contents from these cache.

First let us look a bit more into the history of the evolution of cache memory usage in IC design. CPU designs always exploited the performance benefits gained by the use of cache memory, especially when the gap between CPU speeds and memory access widened [JHS99] over time. With the advent of multi-core processor based system designs, the cache memory usage in these systems evolved to be based on hierarchical multi-level caches and various cache coherence protocols [AA17] got developed.

With the hierarchical multi-level cache, now we have systems with cache memory between CPU and main memory organised at multiple levels - with the faster (due to cost typically in smaller size) cache memory closer to CPU (called L1 cache) while comparatively slower but larger sized cache memories closer to the main memory (called L2 cache). Some modern systems have a further Level3 (L3) cache called as system level cache, which makes that particular multi-level cache configuration to have 3 levels. This indeed brought-in its own level of complexity in maintaining coherency at various levels [BW88] which also contributes to the resultant testability issues.

The above mentioned cache coherency management can be done either as a software or hardware solution. CCI is one such hardware solution which implements the cache coherency management entirely as hardware logic within an IC. Thus CCI can be explained simply as a hardware component that manages the coherent view of cache memory data for the users of cache in a system. CCI provides a basic

interconnect functionality (connecting multiple components in the IC, similar to a system bus), but also provides cache coherency at hardware level as explained, which makes it a complex hardware logic unit.

The diagram in Figure 2.2 outlines the coherency states as maintained by CCI during its operation. The hardware logic ensures the coherency of cache data, as seen by the users, based on the CCI state information. As part of providing this coherency at hardware level, CCI ensures that any data in the cache that is marked as 'shared' is always up-to-date and any consumer of that data can always see coherent data. At hardware level it tracks the cache validity and will flush and re-fetch cache data from memory (if found invalid) in order to ensure that the coherent view of data is maintained all the time.

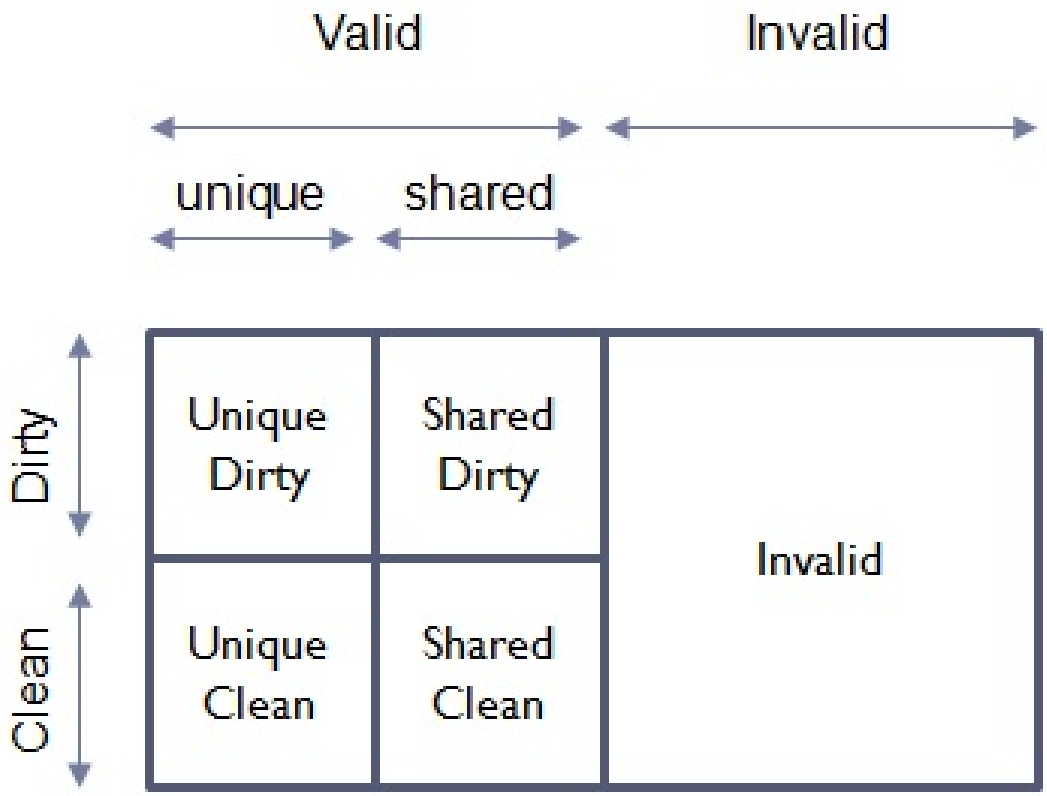

Figure 2.2: CCI States - details

CCI performs this coherency management with the help of certain transactions (operations). CCI supports various transaction types as initiated by any user (called CCI 'master') to ensure the cache coherency. These transactions are defined and governed by a specification known as Arm AMBA® AXI® protocol. The following are the transaction groups:

- Non-shared - Read / Write

- Non-cached - ReadOnce / WriteUnique / WriteLineUnique
- Cache Maintenance - CleanShared / CleanInvalid / MakeInvalid
- Shareable Read - ReadShared / ReadClean / ReadNotSharedDirty
- ShareableWrite - MakeUnique / ReadUnique / CleanUnique and
- Write-Back - WriteBack / WriteClean / Evict

In simple terms, CCI maintains the above mentioned state information. With the support of a series of transactions initiated by the various entities interacting with CCI, it performs the maintenance of cache coherency functionality. It is important to note that this is a complex functional block implemented in hardware and moreover this is a component which is deeply embedded that provides no direct interfaces to test it especially in a full system context.

2.3.2 CCI in Real System Designs - TC2 and Juno

Knowing what is the basic purpose of CCI from Section 2.3.1, let us look at the details of a few real systems where CCI is used as part of an overall hardware design. In this research we specifically look at CCI400 [Arma] from the CCI product family. This has been used in real platforms with multi-core CPU clusters, all of which have access to a multi-level cache, managed by CCI.

Test Chip2 (TC2) [Armb] is a 32-bit Arm development board, which provides a reference implementation of Arm hardware design including CCI. This platform has multi-level cache accessed across 2 heterogeneous CPU clusters with different cache configurations. This variation in the Level2 (L2) cache size is as a result of the heterogeneous nature of CPU Clusters - due to difference in both CPU type and in CPU topology - which make the data access operations across clusters different in nature. The TC2 platform has a big cluster made of dual-core Cortex™-A15 MPCore™CPUs and a little cluster made of tri-core Cortex™-A7 MPCore™CPUs.

Juno [Armc] is a 64-bit Arm development board. Similar to the TC2 platform, Juno also has a heterogeneous cluster of CPUs with multi-level cache for data access and uses CCI400 for cache coherency management. However there are some significant differences between TC2 and Juno platforms, especially w.r.t cache configuration that needs to be noted.

There are multiple variants of Juno platform and in this study we chose the R0 variant of Juno platform. The Juno R0 platform has a big cluster made of dual-core Cortex™-A57 MPCore™CPUs and a little cluster made of quad-core Cortex™-A53 MPCore™CPUs. It has both Level1 (L1) and Level2 caches, with CCI-400

providing the interconnect and cache coherency across the 2 CPU clusters. Additionally Juno[1] also has a quad-core Arm Mali™Graphics Processing Unit (GPU) with 128KB L2 cache.

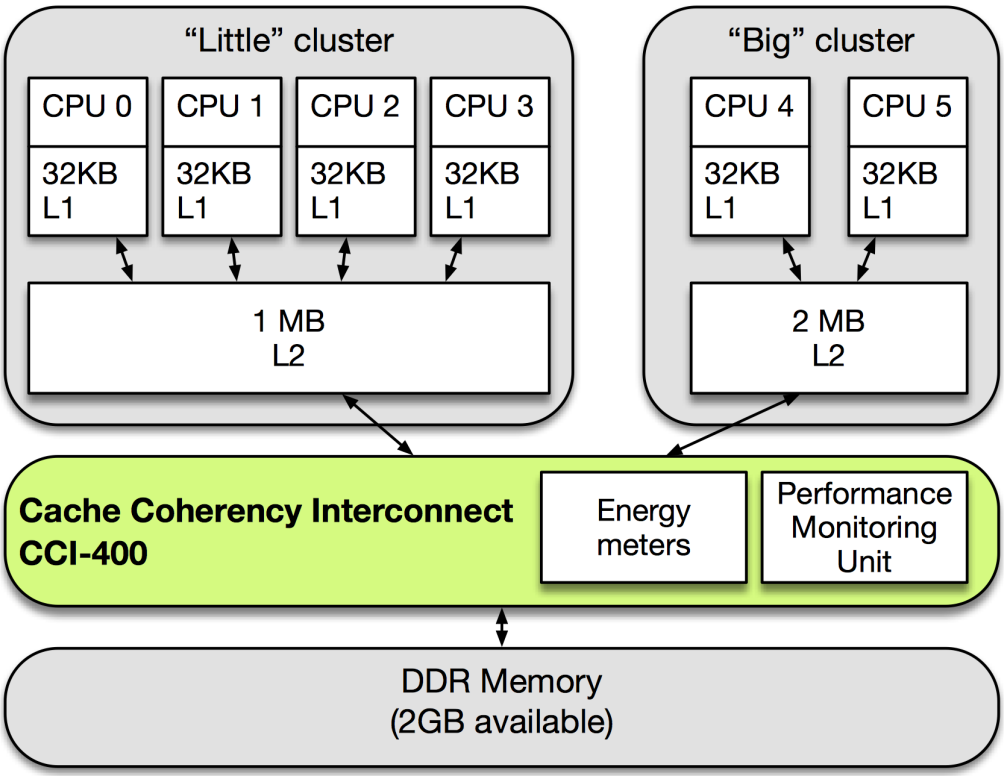

Figure 2.3: Juno platform with CCI-400.

Figure 2.3 shows where the CCI sits in the context of a Juno platform design. Juno has two processor clusters, where different processors may require access to the same data (hence the need for a cache coherent interconnect). On Juno the little cluster has 1MB L2 cache compared to that of 512KB L2 cache in case of TC2 platform [Armb]. The big cluster on Juno platform has 2MB L2 cache compared to 1MB in case of TC2 platform [Armb].

The above described systems illustrate how CCI based systems can be used in systems with different configurations. One pertinent point from a testing perspective is that the CCI component sits below a mixture of complex hardware and software layers. We deploy a full Android software (released by Linaro [Lin]) stack and an user-space application that acts as test payload performing memory operations via CPU elements. Each memory operation is modulated by a L1 and a L2 cache. The CCI sits between the L2 caches for the various clusters and the physical memory. The CCI can in other words not be tested directly which compounds the testing challenge.

[1]All references of 'Juno' would imply R0 variant of the board in the rest of the thesis.

Some of the existing capabilities of CCI provides unique benefits in understanding the behaviour of this deeply embedded component during operation. These capabilities are highlighted in the below sections. They do have significant role in the the current research on improving stress testing of CCI from a full system context, hence let us look at them in detail.

Performance Monitoring Unit: Capabilities CCI includes a built-in component called the Performance Monitoring Unit (PMU) [Arma], which gathers various statistics about the operation of the interconnect at run-time and exposes them through counters monitoring certain events. Traditionally this had always been used for debug purposes. There are multiple counters and different event types that could be monitored using this PMU logic block. CCI-400 supports four 32-bit counters allowing one event per counter to be monitored in parallel. This means one can monitor up-to 4 events in parallel without incurring any penalty in performance.

One of the interesting events which are monitored by CCI is 'stall cycles' associated with various transactions. As a general rule of thumb, the occurrence of stall cycles indicates that the CCI component has to pause some of its operations to cope with the heightened demand for ensuring coherency. The count of these stall cycles can be used as a measure of stress in the system. Larger numbers of stall cycles can be an indication of stress in the platform leading to sub-optimal system performance. This increases the observability aspect of testability of this CCI component from system level, which should be considered beyond the traditional debugging context and should be exploited during testing.

On-board energy meters: Capabilities Juno has four on-board energy meters [Armc], which can be used to measure energy consumption during its operational state. These energy meters report various instantaneous measurements like voltage, current, power etc., across various sub-components of the system. This includes the energy measurements for the CCI component also.

CCI component's power measurement will be reported under the 'rest of the system components' in the System-on-Chip (SoC) group (a SoC is the IC that has all the various components such as CPU, GPU, various interconnects and input-output ports). This excludes power measurements for the CPU clusters and GPU within the SoC, which are normally measured and reported separately. These hardware monitor counters are exposed to the user via memory mapped registers, which can be read by a user application at set intervals. Energy measurement thus obtained provides an orthogonal but none-the-less important view of how optimally the underlying system is operating. Higher stress will manifest as larger energy consumption and hence could be used as an indicator of stress conditions related to this component.

2.3.3 CCI: Current Testing Practices and Challenges

Now that we know the details of CCI and its functionality let us focus on the current set of verification methodologies employed during the testing of CCI. As in case of any other typical hardware IP component, the verification methodology for CCI do employ a range of standard techniques as below:

1. formal methods [Mar06].
2. constrained-random testing [Meh17].
3. protocol testing.
4. functional validation at various levels.

Hardware verification methodologies put great emphasis on Design for Testability (DfT) and provides a wide range of testability functions including that of built-in self-test functionality. In addition to the various tests targeting the hardware IP component in isolation, integration and system tests are also performed, some with specialised software stack mimicking typical complex software behaviours with synthetic test payloads.

With the software development progressing alongside the development of hardware components, the initial testing at a system level is often limited to early software bring-up activities which have a limited scope in terms of testing. Here the various software components which are part of the full E2E stack are brought up on the hardware platform. The early bring-up stage mainly focuses on enabling enough minimal software to be brought up. This allows the hardware to be extensively tested for stability and other characteristics which are critical part of this early bring-up stage. These platform characterisations will include understanding the thermal limits, power envelop, leakage currents, operating voltage and frequencies etc, all of which could lead to hardware level adjustments before the full system software tests could be commenced. It is important to note that this early bring-up stage does not provide the opportunity for a full-fledged system testing to assess the system behaviour with the E2E software stack. This is because here we are still stabilising the hardware and ensuring safe operational limits are determined and then set before any full system testing can happen.

Looking at the existing testing practices, predominantly cache memory testing had been focused on functional correctness, especially at an individual component level. Being a hardware component, most of these functional tests are based on hardware self-tests [The+13] [The+14]. Given that cache behaviour is critical to the overall system performance, there is an increasing need to focus on non-functional testing of this deeply embedded component at a system level. There have been some

recent efforts to auto-generate stress tests for cache coherency at SoC level by You *et al.* [You+16]. Here the target of testing is SoC (at hardware level), which is the IC that has all the various components like CPU, various interconnects and input-output ports. This employs generation of random test vectors with synthetic traffic generators at a SoC level. This provides advantages in terms of having additional coverage when compared to traditional cache testing. However this approach still does not provide the ability to test cache behaviour from a full system context with E2E software stack; instead it is done at the SoC level.

System Testing Limitations When CCI moves to system level testing, we are able to exercise the full E2E stack on the real hardware. However CCI being deeply embedded it is now under multiple layers of hardware and software components with no direct interfaces to interact with. This makes CCI testing at system level very difficult. None-the-less CCI is still considered a critical component of the IC design whose performance can impact the overall system behaviour. There is a much higher emphasis on non-functional testing like stress testing to be conducted on this CCI component in a full system context. The CCI component has many tunable configurations that can be set when integrated into the full system. It is not possible to exhaustively test these configurations at component level. Moreover the CCI component behaviour can be affected by other components integrated within the system. All these factors makes it important that we perform effective testing of CCI in a full system context.

Currently, tests are largely developed by hand for system testing of cache behaviour. In order to cater for specific configurations - based on specialist knowledge of the target system - separate tests are designed. The downside of this is that there is very limited reuse of such tests; a slight change in configurations can lead to very different behaviours, necessitating very different test sets. Automating an approach to generate these tests would improve the re-usability and an overall reduction in the time and effort required to test these complex IC based systems.

As one can see from the above description, testing CCI which manages cache coherency of a multi-level cache based system is difficult. This difficulty is further increased when we want to perform these tests from a full system context which is a very important goal before the final product release. This research aim to address this specific testing challenge in the context of testing CCI from system level.

2.4 Problem Statement

With the detailed understanding of CCI and deeply embedded components in general, especially looking at the testing challenges in this area, let us try to understand

why CCI testing is difficult and describe the problem this current research is addressing.

The problem statement for this research can be summarised as:

Deeply embedded components such as CCI are critical to the performance of IC based systems. Current testing practices have limited capabilities in effectively stress testing this critical component from the system level. The lack of direct interfaces to this deeply embedded component poses a number of testability challenges which cannot be addressed by traditional test methods. This research aim to devise a test methodology which can effectively stress test the CCI component from system level as a means to expose any performance issues that can affect the final system behaviour. Thus it aims to improve the overall system level testing capabilities of IC testing, specifically targeting deeply embedded components like CCI.

One of the fundamental research questions we want to address is whether search-based test methodology could be suitable for addressing the testing challenge posed by deeply embedded IC components. This is reflected in the first research question we address where we evaluate the suitability of search-based algorithms in generating stress tests for deeply embedded IC components.

Let us now inspect why CCI is particularly hard to test from a system level. From a testing perspective the multi-level cache (See Section 2.3.1) in IC designs introduce run-time behaviours which are hard to control, because the cache state at each level is inter-dependent and the cache operations at one level would impact the state of cache memory at a different level in the hierarchy. This results in the difficulty to determine the cache states at run-time, which in turn makes the validation of multi-level cache very challenging. This is more challenging in case of system testing where these run-time interactions and state changes are even harder to control or to determine.

Let us look into the details of what are the multiple factors that contribute to testability [VM95] such as 'Observability' and 'Information Loss' in addition to the traditional definition of testability, which states testability as the ability to set a test criteria and perform tests in order to determine if a given test criteria is met or not. In simple terms 'testability' can be stated as:

- how easy it is to see the internal operations during a test to determine if a test criterion is met. This maps with 'Observability' aspect of testability.
- whether all the internal state change information is visible via the final output. This maps with 'Information Loss' aspect of testability.
- how easy it is to generate a test criterion in order to perform an operation and assert if that test criterion is indeed met or not.

With the understanding of what typically testability means now let us look at testability in relation to the CCI component. Let us analyse the testing challenges w.r.t CCI component in a full system context which includes testability and others challenges. These are outlined below:

- CCI is at the bottom of a complex software and hardware stack. During operation it is difficult to provide direct inputs to the CCI component as part of setting a test criteria and performing a test to evaluate if that criteria is being met. This makes it poorly testable from system level.
- CCI is implemented alongside complex functionality like multi-stage pipelines and speculative fetch all of which introduce run-time behaviours that are hard to control. This adds to the difficulty in setting a test criteria and performing a test to evaluate that criteria, which again results in the poor testability of the component.
- CCI deals with multi-level cache hierarchies which makes it difficult to observe the detailed internal states during its operation. This makes the 'Observability' criteria for CCI component quite low.
- Cache maintenance operations happening within CCI at various levels will not be readily understood by looking at the final state of the basic memory read-write operations. This results in the 'Information Loss' criteria for CCI to be considered as high making it a poorly testable component.
- CCI being a highly reconfigurable component at both design and run-time makes it expensive to test all configurations.

Looking at the testability challenges associated with the CCI component we want to analyse whether complex search-based algorithms can better address the testing challenges posed by IC testing. In the second research question we specifically evaluate how better complex search-based algorithms might be able to generate stress tests targeting IC components.

Considering the critical nature of CCI in the final system behaviour and the above listed testability challenges, it would be beneficial to come up with an effective test methodology to stress test this component in full system context. If this test methodology can generate the tests in an automated manner then this can address the issue related to the lack of reusability of tests also, which is another factor that affects the current test practices. When focused on testing deeply embedded IC components, it is critical to ensure that search-based approach can handle multiple stress factors. In this research as part of the third research question, we evaluate

whether multi-objective search algorithms can generate better stress tests targeting deeply embedded IC components.

Now that we have understood the key problem addressed in this research, let us look at the potential of exploiting search-based approaches in solving this problem. The next section gives an outline of what options search-based approaches presents to us in addressing this testing challenge and how it provides the motivation for this research.

2.5 Search-Based Software Testing (SBST)

In the earlier sections we looked at the wider background of IC development and the testability issues affecting deeply embedded components like CCI. With a clear understanding of the problem at hand let us look at the search-based approaches and see how it can help in addressing the testing challenges posed by CCI. Let us look more into the various offerings from SBST methodology. First we look at related research on cache testing in general and the current application of SBST in the context of cache testing. Subsequently we look at various options offered by SBST approaches which is seen as a natural avenue to explore to solve the complex testing challenge related to CCI.

2.5.1 Related Research

The focus of this research is on the application of SBST methods for system testing in the context of testing deeply embedded hardware components from system level. Application of search-based test techniques are not new at system level testing and in particular targeting non-functional system properties including performance analysis as explained by Shen *et al.* [She+15]. This uses genetic algorithms relying on a combination of search-based heuristics and utilises data mining of execution traces to identify performance bottlenecks.

Further surveys by Afzal *et al.* [ATF09] confirms its application being extended to other non-functional system attributes such as safety [BPS03], usability, quality of service [Can+05] and security [Del+05]. From these it is clear that search-based algorithms can be applied to testing a range of non-functional system properties. This survey shows the use of a range of search-based algorithms including simpler ones such as hill-climbing, Simulated Annealing and population based algorithms like Genetic algorithms, Ant colony, particle swarm optimisation etc. So it does provide us with a wide range of options which is encouraging.

SBST has often been used in the context of software testing and much less for hardware testing. Most of the search-based software testing techniques applied on

hardware uses Hardware-In-the-Loop (HIL) systems [WK09; LW10]. Wegener *et al.* developed a framework that integrates an evolutionary testing framework with a testing platform that supports model-in-the-loop [Mut+11], software-in-the-loop [CS16] and HIL testing of embedded systems in an industrial usage context where it describes the testing of an antilock-braking system electronic control unit. Lindar *et al.* present an approach in which evolutionary functional testing is performed using an actual electronic control unit for test case evaluation with HIL systems. One notable advantage of HIL systems is that they provide additional testability components and interfaces which supports the application of search-based algorithms, which may not be the case for other hardware system testing scenarios. In the current study we are applying search-based algorithms on hardware components without the typical HIL setup, but seeking to employ alternate methods to test deeply embedded IC components at system level.

In relation to the testing of CCI components at system level we want to analyse the existing state-of-the-art techniques in terms of cache testing especially focusing on the non-functional testing of cache component at system level. At a broader level, we can see that search-based algorithms are successfully used in stress testing of real-time systems [BLS05]. Briand *et al.* implemented the application of search-based algorithms in a tool that could actually help testers identify test cases that will likely stress the system to such an extent that some tasks may miss deadlines. Alesio *et al.* [Ale+15] show how search approaches can be successfully used in stress testing.

In this research we aim to focus on the testing of CCI component which performs cache coherency management in the system. Typically most of the testing around cache memory and cache coherency are functional tests in nature [QM12]. Qin *et al.* developed an efficient test generation technique, which can be used to achieve full state and transition coverage in simulation based verification for a wide variety of cache coherence protocols. But this is mainly focused on testing functional correctness and what we need for our research problem is to focus on non-functional testing. All the prevalent research into cache testing are mostly focused on hardware self-tests [The+13; The+14]. Theodorou *et al.* developed a SBST program development methodology used for on-line testing of small cache memories in microprocessors. We also found that some recent research do investigate the possibility of auto-generation of stress tests targeting cache coherency at SoC level [You+16]. But crucially this was still targeted at SoC level rather than system level. You *et al.* developed a SoC system level cache coherence stress tester called 'Red Baron', which automatically generates test cases on the fly, taking into account the behavior of the CPU's cache, Input-Output (IO) coherence, system memory and interconnect. Successful application of SBST methodology in the area of memory system valida-

tion can be seen in the work using genetic algorithms with Memory Consistency Model (MCM) verification as per [EN16]. Elver *et al.* developed a test generation framework called 'McVerSi', which employs a Genetic Programming (GP) based approach to MCM test generation. 'McVerSi' relies on a novel crossover function that prioritizes memory operations contributing to non-determinism, thereby increasing the probability of uncovering MCM bugs. Further we could see that in the area of testing cache coherency management, again the focus had been mostly on functional testing as explained in [Acl+15]. Acle *et al.* proposes a method to test the cache coherence logic existing within each core in a multi-core system with ability to detect hardware defects affecting this logic. All these related research encourage us to explore SBST domain to address the CCI testing challenge we attempt to solve with this research.

We can see that there is a good amount of research on the use of search-based algorithms for improving system level testing and in the cache memory testing domain. This puts us in a firm footing and gives us the confidence to explore the use of search-based algorithms in addressing the testability challenges posed by CCI component in a system context.

2.5.2 SBST: Solution Options

SBST techniques re-frame the testing challenge as a search-problem [McM04]. This can support test automation by leaving the selection and execution of test cases to an algorithm, which selects test inputs with the aim of optimising some objective. This approach is particularly suitable in cases where a) the search-space is large [Ber07] and b) there is higher levels of non-determinism during the test execution [BLS05], which drives the popularity of SBST for automated test generation [Har07]. By defining a suitable fitness function and allowing the algorithm to explore the search-space, it presents us with a very optimal approach to tackle such complex testing scenarios akin to the CCI testing problem outlined in the previous section.

Search-based algorithms adopt a wide range of approaches which can suit certain problem domains more than others. Search-based algorithms are ultimately guided by an 'objective function' or a 'fitness function' (a function that is used to evaluate a test case); in testing this tends to be a measure of code coverage (or model-coverage if the tests are being derived from a specification). But this objective function is always problem dependent and one need to derive a suitable one for the problem at hand. Objective functions can be 'single objective' where it is trying to optimise a single aspect or it could be 'multi-objective' where multiple aspects determine the final optimisation. There are a range of algorithms that are potential candidates for our consideration.

We discuss some of the typical search-based algorithms below which are possible candidates for us to apply in addressing the CCI non-functional testing challenge. Since search algorithm's success are always dependent on the problem domain, we don't have any clear favourites to start with. Naturally we want to start from simple algorithms and once proven want to explore more complex ones that would better suite the real-world testing needs of CCI. Starting from simpler approaches we look at some of the single objective fitness function based search approaches in below sections.

Simple Hill-Climbing Given a starting-point in a search-space (this can be a test case or a test set, depending on the objective function), the hill-climbing algorithm [JY04] will start by evaluating its 'neighbourhood' - by running and evaluating adjacent test cases. It then simply picks the test case that led to the best improvement in terms of the objective function. This process is repeated until no further improvements can be made (in which case it has hit a maximum / minimum point for the evaluated fitness function corresponding to a solution in the search-space).

Simple hill-climbing is a popular choice for search-based approaches due to its simplicity and effectiveness. However, as with any search-based algorithms, its suitability is under-pinned by the nature of the search-space to which the problem is modelled. If the search-space, that the algorithm will explore, results in a fitness landscape that is multi-modal - meaning there could be multiple local maxima or minima, then this could hinder the effectiveness of the simple hill-climbing. This 'local maxima problem' is one of the major drawbacks of local search algorithms such as simple hill-climbing algorithm. None-the-less, considering the simplicity and typical effectiveness of simple hill-climbing, this is definitely an option to be explored.

In the first research question we focus on evaluating the suitability of search-based algorithms in generating stress tests for CCI components. For this evaluation we use the simplest of the search-based approaches viz, simple hill-climbing.

Random-Restart Hill-Climbing The random-restart variant of hill-climbing algorithm [JY04; YM93] provides a means to escape the local maxima by restarting the search from a randomly selected new starting point with the increased possibility of arriving at a better solution. It retains the simplicity of hill-climbing algorithm but is more suitable for search-spaces with resultant fitness landscape that have multiple local maxima present.

A random-restart hill-climbing algorithm when provided with a fixed budget will be starting with a random seed. When it arrives at a local maxima and if the execution budget is still unspent, it will save the current best solution and restarts

with a new random seed. Once the execution budget expires the algorithm provides the saved best solution from across all the restart sessions it went through. This is definitely an option to be considered.

Simulated Annealing Simulated Annealing [NJ10; PK98] is a search-based algorithm inspired by the thermodynamic principles involved in the process of 'Annealing' used in metallurgy. In the physical world, as part of the annealing process, the metal is heated to a high temperature which allows the internal state of the metal to be altered from the original state. The metal is then cooled down gradually allowing its internal properties to be changed as desired.

In Simulated Annealing we start the search process with a high 'temperature' value. In this phase, similar to that of the physical annealing process, it allows the algorithm to explore paths in the search-space even if they are led by sub-optimal neighbouring candidates. This gives the algorithm an increasing chance of escaping the local maxima at the earlier stages of the execution. As execution time progresses the temperature is reduced which in turn reduces the probability of the algorithm to select sub-optimal candidates for search-space exploration. Thus the probability of the algorithm selecting a sub-optimal neighbouring candidate for exploration is higher at the earlier stages of the execution when the 'temperature' is high and progressively reduces at later stages of execution. Simulated Annealing is seen effective in avoiding local maxima in some problem domains such as operations research and job scheduling [NJ10]. It is a worthy candidate to be considered.

Both random-restart hill-climbing and simulated annealing are advanced search-based algorithms which can effectively work in fitness landscape having multiple local maxima. These form the basis for the second research question, through which we analyse whether advanced search-based algorithms can generate better stress tests for IC components.

Now that we looked various options for single objective search algorithms, let us look at what option we do have for multi-objective search algorithms in the below section. Remember that in case of CCI testing the stress conditions could be manifested in multiple ways and multi-objective algorithms might be what is required to address the real-world testing needs.

SPEA2 Multi-objective fitness function based search algorithms are best used in the context of population-based evolutionary algorithms. These are typically beneficial when there are more than one factor to be considered as part of the fitness function evaluation. Evolutionary multi-objective optimisation problems typically focus on solving the type of problems that involve finding optimal solutions where there are multiple conflicting objectives. One of the well-used and effective multi-

objective search-based algorithm is SPEA2. Amongst the elitist multi-objective optimisation approaches SPEA2 [ZLT01] is demonstrated to outperform other options like Strength Pareto Evolutionary Algorithm (SPEA) [ZT98], Pareto Envelop-based Selection Algorithm (PESA) [CKO00] and Non-dominate Sorting Genetic Algorithm 2 (NSGA2) [Deb+00]. Hence we definitely want to keep SPEA2 as an option to evaluate for solving CCI testing problem.

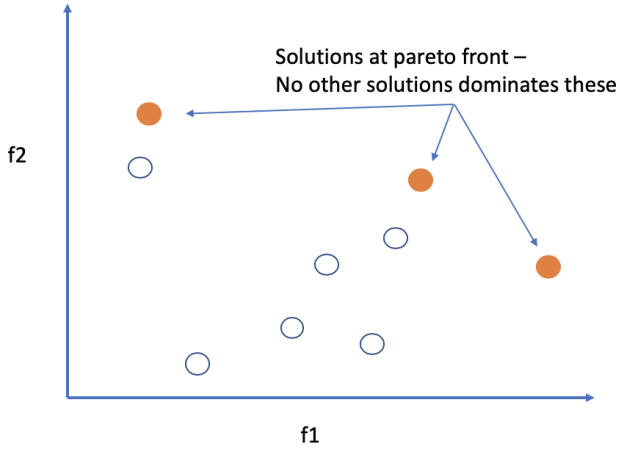

Figure 2.4: Solution candidates at pareto front

Let us look into more details of SPEA2. Like other multi-objective search algorithms, SPEA2 also aim to maximise multiple objectives and provide a solution set from the pareto optimal front (See Figure 2.4), which includes the non-dominate set of solutions [KCS06]. This means that the multi-objective algorithms aim to ensure none of the candidates amongst the final solution set dominates one another. SPEA2 particularly fares better than the other alternatives based on a few aspects like a) preserving the boundary solutions in the pareto optimal set, thus ensuring diversity of solutions, b) using better fitness assignment by including a density function, which helps to provide a solution set more closer to the ideal pareto optimal front and c) avoiding clustering of solutions in the pareto front thus again ensuring a diverse set of final candidate solutions. These positive attributes of SPEA2 made us chose this as the multi-objective search algorithm to experiment with the CCI testing challenge.

We also want to take note of some of the key improvements SPEA2 made on its predecessor SPEA:

- *the archive truncation method to preserve boundary solutions* - In SPEA, when the algorithm performs generational evolution through crossover and mutation operations, there would be situations where there are more solutions than the archive population size which results in the need for truncation. SPEA relies on clustering techniques which could still lead to outer solutions being lost.

SPEA2 provides improvement on SPEA by preserving such boundary solutions during the archive truncation process.

- *improved fitness assignment* - In SPEA the fitness assignment considers only dominating candidates. This results in a situation where individuals dominated by same archive members have same fitness value which could lead to SPEA behaving as a random search in certain cases. In SPEA2 it takes into account both dominating and non-dominating candidates when doing the fitness assignment, which provides better fitness assignment than SPEA.

- *improved use of density information* - In SPEA, only during archive truncation the clustering techniques are used that rely on density information. On the contrary, in case of SPEA2 it relies on density information during the generation of the main population and also when the generational evolution happens. This improves in reducing clustering of solutions with SPEA2.

As the stress on the CCI component can be affected by multiple factors and manifesting in multiple ways, we want to evaluate whether SPEA2 (being a multi-objective search algorithm) can generate better stress tests compared to single objective algorithms. This forms the focus of the third research question. Here we also adapt simulated annealing to use a composite fitness function to consider both objectives and assess if that can generate better stress tests.

In this section we looked at how search-based approaches are already used in cache memory testing. We arrived at a possible set of search-based approaches that could form a basis to address the system testing challenge posed by CCI testing. This research employs these algorithms in various empirical studies as part of answering some key research questions in an effort to address the CCI testing challenge.

2.6 Summary

In this chapter we looked at the wider context of testing challenges associated with deeply embedded IC components in a full system context. We looked at the traditional hardware-software co-design workflow where such components get developed and discussed the current state-of-the-art in terms of validating these. We looked at the associated testability challenges in the specific context of CCI component to understand the problem better. Here we outlined the problem statement w.r.t the testability issues of CCI at system testing and explained our motivation for improving the testing approach by adopting SBST techniques. We also looked at the various potential search-based algorithms that can be explored while addressing this research problem.

Chapter 3

Search-Space Model and Experimentation Framework

In the previous chapter we looked at the details of why testing CCI is difficult, particularly from system level and how potentially search-based approaches can form a basis to address this testing challenge. In this chapter we focus on how we re-frame the CCI testing challenge as a search problem and how we device an experimentation framework to solve this search problem. We know SBST provides multiple options to explore but we would need a framework with which we can experiment and compare the different options based on the search-space model we devise in order to address the CCI testing problem.

First in Section 3.1 we look at the IC testing problem in a more broader context in terms of the testability challenges it presents. We also reiterate why search-based approaches are indeed appealing for this problem domain.

A suitable search-space model and an appropriate fitness function are critical factors that would underpin the successful application of search-based algorithms. In Section 3.2 we look at how we devised a very intuitive search-space model for the IC testing problem considering cache behaviour alone. It also explains how we conducted some quick experiments to gather anecdotal evidence to assess whether this simple search-space representation is sufficient to address this search problem.

In Section 3.3 we explain how a fundamental rethink on the search-space representation for this problem domain resulted in a more suitable search-space model. It explains how the new search-space model is defined in terms of key attributes that affects the stress on IC components. Also in Section 3.4 we give details of an experimentation framework used to compare various SBST options. This framework allows us to conduct various empirical studies to address all the research questions posed as part of this research.

3.1 Understanding the Problem Domain

Before looking at the search-space modelling of the test problem, let us gain better understand of the problem domain. The CCI is representative of many 'deeply embedded' IC components – hardware that can only be controlled through several other layers of hardware and software. The various characteristics of the search-space – the non-determinism, the lack of an ability to reliably reset to a known state, and the sensitivity to a variety of exogenous factors – would appear to favour the use of search-based algorithms. As is the case with most embedded systems, the CCI is difficult to test (at least in a system context) because of the earlier outlined testability issues (See Section 2.4). It sits at the bottom of a relatively complex stack of hardware and software components (including the Android operating system). With the operating system continually manipulating the memory and catering for other routine OS processes it is virtually impossible to reset to a fixed state at run-time. This is accentuated by the run-time behaviour of various advanced features of CCI and the effects of systems with a multi-stage pipeline. The above details highlight the difficulties posed by this testing challenge.

Harman [Har07] *et al.* explain the motivation for applying meta-heuristic search-based methods on software engineering problems. They explain that the precise optimisation algorithms like linear programming which are deterministic in nature are not applicable to software engineering problems which have an objective that cannot be simply characterised by a set of linear equations. CCI testing problem also shares the characteristics of typical software engineering problems, which are:

- Global Optimum - there is no guarantee the global optimum will be found. The runtime characteristics of CCI component can result in a fitness landscape which has no single global maxima.

- Predictability - each execution potentially yield different results. Lack of ability to precisely set a known initial state and the runtime interference from other software and hardware operations in the system affecting CCI can lead to this lack of predictability.

- Computational Expense - a large number of candidate solutions need to be considered before an optimal solution is finalised. With the various factors affecting CCI and the interactions of these components will result in search-space model for CCI testing, that will generate a large amount of candidate solutions. This large volume of candidate solutions needs to be evaluated before any optimal solution could be finalised.

All the above listed characteristics of the search-space for CCI drives our motivation to explore the use of search-based testing methodologies to address this testing

challenge. Thus search-based test techniques form a natural starting point for addressing challenges posed by CCI testing. They would appear to be ideally suited for handling the complexity and lack of testability associated with CCI. Search-based approaches have been successfully applied in testing cache memory components in the past. However, these are mostly focused on functional correctness [Acl+15]. With the testing of CCI we are essentially performing a series of memory operations that would stress the functionality of CCI component in ensuring the cache coherency. So a very intuitive way of defining this search-space model is by mapping it to a set of memory operations. And the fitness function will be a mechanism to measure the stress on CCI component during the execution of these memory operations.

3.2 Initial Search-Space Model

In this section we look at some of the quick experiments done to gather anecdotal evidences to assess the suitability of an initial state-space model. First we look at an intuitive state-space representation for this cache testing problem.

Let us look at the operation of the cache memory as depicted in Figure 3.1. Here we can see that when an application access a memory location from the main memory, the cache memory fetches the cache line which includes that referenced memory location. Subsequent cache line fetch and flush operations keep the content in the cache memory up-to-date to service the user application request. Thus a 2-D array of memory locations becomes a very intuitive notion to represent a search-space for cache memory operations. With such a search-space we expect the search algorithm to find the set of memory locations that would trigger cache operations while performing read-write operations on those memory locations.

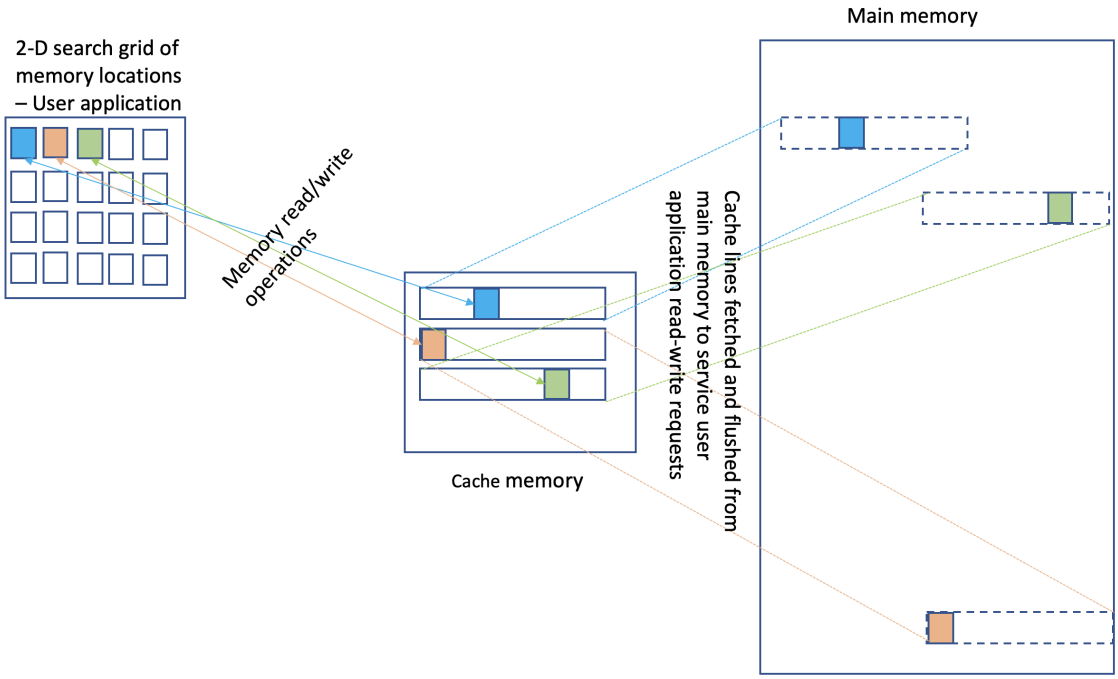

Figure 3.1: Cache memory operations defined as search grid

In order to stress test CCI component, we recognise that this has to be memory read-write operations performed during a test sequence on a set of memory locations that forces the occurrence of multiple cache line fetch and flush sequences. Keeping cache memory behaviour in mind, such a good test sequence could be constituted of read and write operations from distinct parts of the run-time memory, that will force cache management operations, which in-turn stress the CCI component. This constitutes the basis for our formulation of a very intuitive and simple search-space representation - henceforth referenced as a 'search grid' - that is used for CCI testing and this section provides its details. With this notional search-space model, when we look at the search-based algorithms to start with, hill-climbing comes to the fore-front as a simple and effective option to explore.

3.2.1 Applying Hill-Climbing with Search Grid

As alluded earlier, our simple approach for the search-space of the CCI is to define it as a grid of memory locations from which read-write operations are performed. We define this search-space as a 2 dimensional array (search grid) of memory co-ordinate positions that would be used for performing the memory read-write operations. A simple hill-climbing algorithm is implemented to explore this search-space representation. When hill-climbing algorithm is in action, every single co-ordinate position from the search grid is varied to explore the memory address locations from the immediate neighbourhood. This is done by incrementing either x or y co-ordinate

of the individual search grid element across the iterations. The algorithm moves to a search grid which results in higher stress on CCI in every iteration till no better option is found. At the end of execution, hill-climbing will determine a possible search grid that represents a set of memory access co-ordinates which maximises the evaluated fitness function indicated by the measured stress on the CCI component. Here we use data stall cycles from the PMU counters as the fitness function, which gives the measure of stress on the CCI component.

In order to assess weather this simple search-space representation is suitable, we gathered some anecdotal evidences based on quick experiments done on Arm TC2 platform. The results from this experiment showed that with this simple search-space representation the hill-climbing algorithm performed no better than random testing in terms of measured stress on CCI component. The negative result assessed, based on these anecdotal evidence gathered, can be interpreted as one of 2 things - either the search-space representation is unsuitable for this problem domain or there is an underlying limitation with the chosen search algorithm that makes it unsuitable for the resultant fitness landscape.

Rather than immediately dismissing hill-climbing algorithm's capability, we looked at an extension to basic hill-climbing algorithm implementation. Here we experimented with random-mutation hill-climbing algorithm with the same search grid search-space representation. In this hill-climbing algorithm variant, each search grid element is provided with an arbitrary set of neighbours referred henceforth as 'explorer list'. By providing this explorer list we expect to give the algorithm a larger search-space exploration possibility. The explorer list contains a set of randomly generated memory co-ordinate positions and in every iteration each search grid element will be replaced with a random element from explorer list. This action corresponds to the random-mutation behaviour of hill-climbing. As in the case of earlier hill-climbing experiments, here too the aim is to arrive at a search grid which would maximise the fitness function at the end of a given number of iterations.

Again we gathered some anecdotal evidence based on quick experiments conducted on Arm TC2 platform. The modified version of hill-climbing algorithm also failed to exhibit any benefits beyond what random tests could achieve. We suspected that the failures could be due to the resultant fitness landscape being unsuitable for a local search algorithm such as hill-climbing. This then made us explore the suitability of global search algorithm options and genetic algorithms came out as a popular choice which is explained in the next section. Here again we gathered some anecdotal evidence to see if the simple search-space representation is suitable or not.

3.2.2 Applying Genetic Algorithm with Search Grid

Let us look at general characteristics of genetic algorithms before describing how it is used in this case. Genetic Algorithms represents a class of search-based algorithms which had been proven to be effective in many problem domains that have distinct fitness landscapes. Tabassum *et al.* [TM14] analysed the landscape of problem types where genetic algorithms are used and highlighted the key benefits genetic algorithms have over traditional approaches. The design of genetic algorithms to work with multiple string-coded variables which discretizes the search-space and its ability to operate on multiple population points gives it a better chance of arriving at a global solution. The crossover and mutation operators keep improving the solution candidates in the population and the application of probabilistic rules rather than deterministic rules during generational evolution improves its chance of arriving at global maxima. All these factors encouraged us to explore the use of genetic algorithms as a viable option to consider.

Genetic algorithms demand each problem domain to be mapped to a predefined set of structures and operations. Our first challenge is to represent our search grid search-space model in terms of the structure needed by genetic algorithm. The following sections capture the various elements of this search-space representation when used with genetic algorithms.

Encoding Method The primary challenge when applying genetic algorithms in any problem domain is to arrive at the right encoding for the populations. Similar to the earlier hill-climbing approach here too the search-space is represented as a collection of memory locations of fixed size. A set of memory operations are then performed based on these memory location information. We concluded that *value encoding* is best suited in this situation.

Genome Definition At a very basic level the genotype is defined as a pair of (x,y) memory location co-ordinates which forms part of the 2D array of memory location information. The x and y values vary based on: $0 \leq x \leq xmax$ and $0 \leq y \leq ymax$, where xmax represents the maximum value that x can take in the 2D representation of memory locations and ymax similarly the maximum value y can take.

Chromosome Definition A Chromosome is defined as a 1D array of value encoded genes. The chromosome length is fixed to 64K (64 x 1024 genes) for this experiment. One chromosome represent one individual in the population and the population size is set to 30 individuals in this experiment.

Candidate Selection Strategy In order to populate the mating pool we implemented a candidate selection strategy based on *tournament selection* method. The tournament size is set to $\frac{1}{5}^{\text{th}}$ of the population size. The mating pool is populated with deterministic tournament selection candidates, thus ensuring the fittest from the tournament always results in the mating pool.

Cross-Over Operation Many possible strategies for crossover operation is considered including but not limited to - single point, multi-point and uniform crossover options. Considering its simplicity, the *single point crossover* operation is decided for our experiments and the crossover point is fixed to be the halfway point of the chromosome length. The crossover rate is set as $p = 0.9$ in our experiments to give higher chances for exploration during the search-space navigation. Hassanat *et al.* [Has+19] surveyed the choice of crossover rate for genetic algorithms and have found that the optimal range for crossover rate is between 0.5 and 1.0. However in studies where population sizes are small, the cross-over rate chosen tend to be at the higher end of the scale. Considering these we set the crossover rate as $p = 0.9$ for our experiments which had a relatively smaller population size.

Mutation Operation For the purpose of exploitation of the search-space a process of mutating the candidates during the evolution is implemented. A mutation rate of $p = 0.05$ is selected for the experiment. Hassanat *et al.* [Has+19] surveyed the choice of crossover rate for genetic algorithms and have found that higher mutation rates results in genetic algorithms to behave more like random testing and thus a lower mutation rate is advised. We experimented with mutation rates ranging from 0.0001 to 0.5 (See Appendix A.1.2) and based on the experimental results we finalised on mutation rate of 0.05. For every candidate in the population a random gene from the chromosome of the individual is mutated by incrementing the x and y co-ordinate valued by a constant mutation step. Currently the mutation step value is set to 10 for this experiment.

Generational Replacement Strategy For generational replacement the current implementation follows an *elitist replacement* strategy. The current implementation sets the elite group size as 5 and from the current population it always selects the highest ranked 5 elite candidates and when the next generation mating pool is populated it will transfer these elite candidates. This ensures healthy candidates are passed over from one generation to another, hence supporting the positive evolution over generations.

With the above structure for the genetic algorithm elements we conducted some quick experiments to gather anecdotal evidence to see if the search grid based search-

space representation is suitable or not.

Genetic Algorithm Experiment Details Let us look at some key components of the experimentation setup that is used during this early trials to assess search-space suitability. The Figure 3.2 outlines the framework components which is part of the experimentation setup.

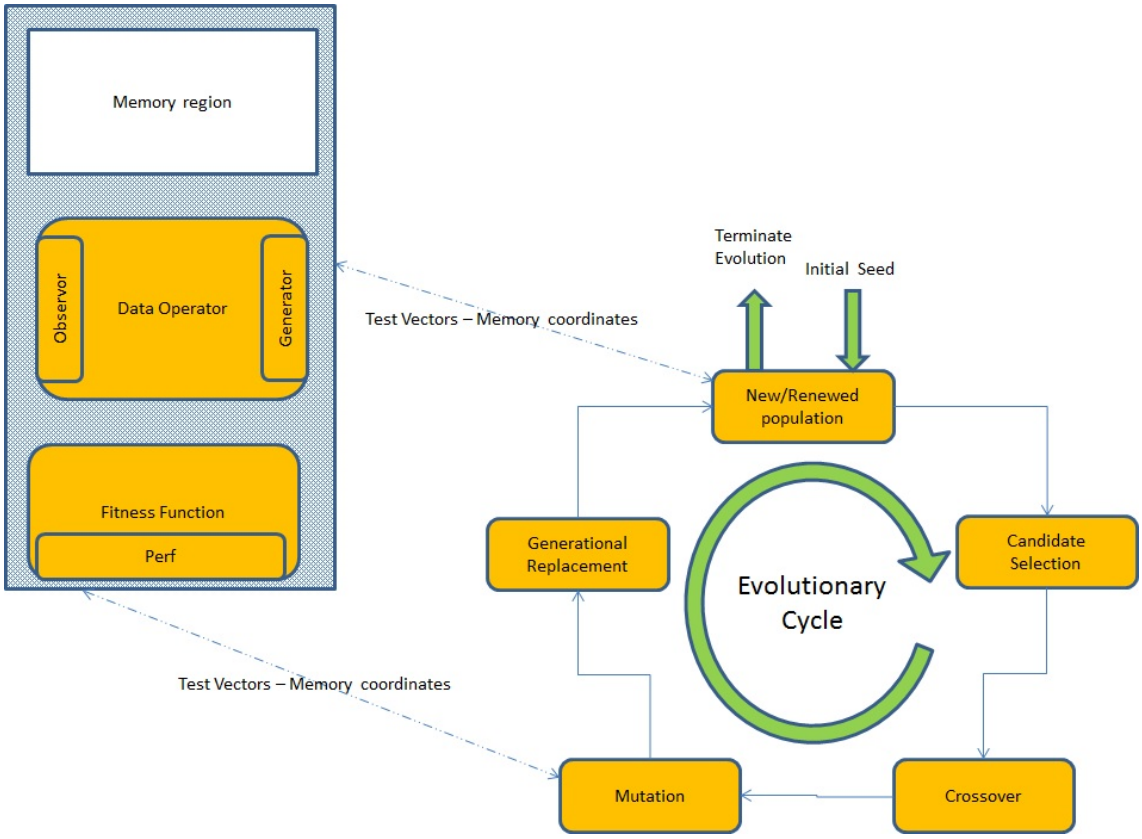

Figure 3.2: Genetic Algorithm - components and control flow.

The 'data operator' component comprises of the memory read-write agents that would perform the memory operations on the locations represented by the population candidates generated by the genetic algorithm evolutionary cycle. The 'fitness function' component uses the PMU counters and capture the perf events related to data stall cycles which gives the measure of stress on the CCI component. The individual components around the genetic algorithm evolutionary cycle like candidate selection, crossover, mutation, generational evolution etc are the standard operations that happen during the evolution of the population from one generation to the next. The Appendix A shows more details of the algorithm followed in this trial.

This quick experiment is also run on Arm TC2 platform. The algorithm starts with a random population based on individuals formed by the above defined chromosome. The algorithm evaluates the fitness of individuals as part of the execution. When an individual's fitness is to be evaluated, the data operate function is invoked

which in turn performs the memory read and write operations on the locations represented as per the individual's chromosome. The perf utility then gathers the PMU event counter values as part of the fitness function evaluation, which then represents the stress on CCI. The algorithm repeats the evolutionary cycle, which then evolves the population across the generations in order to progressively maximise the computed fitness of the population.

Genetic Algorithm Results By applying genetic algorithm with the search grid search-space representation we expected to overcome difficulties faced by hill-climbing algorithm. The rationale being, the hill-climbing algorithm's difficulty with search grid based search-space representation is thought to be as a result of its unsuitability for the resultant fitness landscape. Hill-climbing being a local search algorithm this fitness landscape is not favorable for its use. Thus by choosing a global search algorithm, it might potentially give a better outcome. As part of this some genetic algorithm parameter tuning is also conducted, which can be seen at Appendix A. However when we look at the anecdotal evidence gathered from these quick experiments with genetic algorithms, we still could not see it having any better results than what random testing could achieve. It failed in a similar manner as the case with hill-climbing trials.

The anecdotal evidence gathered from these quick experiments made us to conclude that the search-space model of search grid is not suitable for this problem domain. This search-space model is inadequate to be effectively used with search algorithms. Thus we decided to revisit the search-space model and have a fundamental rethink in defining it. This new search-space model is detailed in the next section which is adopted in this subsequent studies conducted as part of this thesis.

3.3 Final Search-Space Model

Our fundamental goal is to replace the current approach (where we use specialised hand-crafted tests for stress testing CCI component) with search-based algorithms, where we define a suitable search-space model so that we can let the search algorithm to generate the stress tests automatically by leveraging the use of CCI hardware counters as fitness function. Even though we expect the practical testing needs for CCI will demand for a search algorithm to consider more than one aspect while generating the stress tests, there is no fore-gone conclusions on whether a particular search algorithm would fare better than the others in this problem domain. This brings forth the need for defining a suitable search-space model for this problem and moreover having an experimentation framework that would allows us to compare various search-based algorithms to determine which one best suites for this problem

at hand.

As seen from previous section (See Section 3.2) the initial efforts in defining the search-space model as a search grid did not give a good outcome. This simplistic search-space model faltered due to the resultant fitness landscape being not suitable for any search algorithms attempted. This forced us to have a fundamental rethink in finalising a different search-space model that would effectively influence the stress on the CCI component. With this new approach we want to consider various broader system aspects that can play a key role in generating stress on CCI, when the memory read-write operations are performed. The following details captures the key aspects of this new search-space model and the fitness functions chosen to be used with this.

Test Payload Representation The test payload representation as per the new search-space model is determined by the key high level factors that influence the performance of CCI. In this new state-space model the test cases (or test payload) $TS =< PS, SP, AC >$ is represented in three dimensions; each expected to have an effect on the stress generated on this CCI component:

- payload size (PS) - represents the volume of read-write memory operations that would exercise the CCI during the test.
- sparsity (SP) - represents how narrow or wide the region of memory from which the read-write operations are performed during the test.
- actor profile (AC) - represents the number of entities that are performing the read-write operations during the test.

Let us look at each of these test payload attributes as per the new search-space model in detail below.

3.3.1 Payload Size PS:

We expect the volume of memory read-write operations to have an effect on the stress on CCI component. This is represented by payload size attribute. The payload size $PS = (x, y)$ where x and y is varied to represent the amount of test data that is read or written. Here x and y represent the number of columns and rows in memory that will be required to represent the data, where a basic unit of data is 4-bytes. Each element in the 2D array of x by y represents a memory co-ordinate position from which the read-write operation would happen.

3.3.2 Sparsity SP:

We expect the nature of memory read-writes - especially in terms of whether memory is read-from or written-to from same memory region or from far apart memory regions - to have an effect on the stress on CCI component. This is represented by sparsity. The sparsity $SP = [1 : 4]$ is an integer representing four levels of data 'sparsity' that is to be written-to and read-from memory. The sparsity levels are defined as:

Table 3.1: Sparsity Definition

Sparsity level	**Operating memory region**	**Effect (Chance of repeated read-write operation from same memory region)**
1 - Unconstraint	Full	Least
2 - Relatively sparse	1/2	Small
3 - Dense	$1/10^{\text{th}}$	High
4 - Very Dense	$1/100^{\text{th}}$	Highest

The behaviour of the cache will differ if all of the data is to be written and read from a single, contiguous zone of memory, as opposed to a range of non-contiguous, widely dispersed regions. This is based on the principle of 'locality of reference' [Nai15] which is the basis on how memory systems work efficiently and how the cache behaviour is driven.

3.3.3 Actor Profile AC:

The number of actors (or processes) writing-to and reading-from memory can affect cache performance and influence the stress on CCI component. This is represented by actor profile. This factor not only takes into account the number of actors but also considers the manner in which they are spread across the cluster of CPUs. Definition of AC is underpinned by the following rules:

- only one read or write actor is pinned on a CPU, limiting the maximum number of actors in the configuration equal to the total number of CPUs across the clusters.
- same type of actors (either read or write) are pinned on a given cluster. This forces the read and write actors to be on separate clusters which maximises the possibility of cache operations through L2 cache.
- configuration specifies whether we have equal number of read-write actors or dissimilar number of read-write actors.

In this section we can see that the search-space is represented based on these 3 aspects that have an impact in terms of generating stress on the CCI component. A search-based algorithm during its execution will be able to change aspects of these 3 attributes in order to arrive at a test payload that would maximise the stress on the underlying platform. In the next section we look into more details of the fitness function that is used to measure the stress on CCI component with the above state-space model.

3.3.4 The Fitness Function

Testing non-functional properties of ICs such as the CCI is greatly facilitated by the fact that it can be straightforward to obtain performance data without interfering with the routine behaviour of the IC itself. Since this data is so crucial to performance-tuning, the CCI has several dedicated components to measure performance (See Section 2.3.2). At the moment these hardware components are used only during debug operations and we propose to leverage them in a testing context for the first time.

For CCI testing from system level, the basic functional operation is memory read-write operations, which results in a set of internal cache memory operations to manage the cache state. The cache state is continuously influenced by multiple factors at run-time. It is this high degree of change in cache state in response to multiple run-time factors that makes CCI testing from system level difficult to address with traditional test techniques.

As we have different hardware indicators, which provides information about the internal state of the cache memory at run-time, this can be used as a 'measure' to assess the stress or the performance of the component at run-time. This becomes a natural choice to be used as a fitness function when used with search-based algorithms.

The search-based algorithms could potentially use these hardware counters to 'assess' the internal state of the system and use it as a 'measure' of stress on the platform. This can then guide the algorithm in selecting the solutions from the modelled search-space. The fact that we have hardware counters which can be used as fitness functions (that measures stress on the platform) directly enhances the suitability of using search-based algorithms in CCI testing. Let us look at the various options we have to be used as fitness functions for testing CCI.

Single Objective Fitness Function CCI provides a hardware counter called the PMU. We use the PMU to gauge the stress on the platform during the memory read-write operations by recording the number of data stall cycles for a given test.

Data stall cycle is a hardware logic operation which cause the end points of the CCI component to stall their operation. This stalling of end points allow the CCI component to recover from the peak load state and resume its service of the end points. The occurrence of data stall cycles thus simply indicate that CCI is forced to pause the servicing of requests from its users so it can do its internal operations when the stall cycles happen. Simply put, the occurrence of more stall cycles means CCI is struggling to cope with demands and hence indication of higher stress on it. The data stall cycle counts thus provide us with an indication of the stress in the underlying IC during operation.

In the single objective based search algorithms we use these data stall cycles as the measure to guide the algorithm. The aim of the experiments is to generate a single test payload in case of single objective approach that maximises the generation to data stall cycles.

Multi-Objective Fitness Function In a complex component like CCI, multiple factors can affect its operation and the stress on this can be manifested in different ways. One way by which stress on CCI gets manifested is in the form of higher energy consumption. Some platforms provide a measurement unit called the hardware monitor (an on-board energy measurement unit). We can use this to measure the energy consumption of the CCI component during operation.

In multi-objective based search algorithms we use data stall cycles and energy measurement as the measures to guide the algorithm. Here we aim to find the test payloads at the pareto optimal front that can maximise both the data stall cycles and measured energy.

As seen above we now have means to measure the 'goodness' of a test payload in terms of the 'stress' it could be generating on the CCI component. By measuring the data stall cycles and the energy measurement we are able to evaluate if a test payload is better than the other when the search-based algorithm is navigating through the defined search-space. This paves way for us to employ search-based algorithms to generate stress tests in an automated manner from system level which is the key motivation for this research. Given that we now have a finalised search-space model and a suitable set of fitness functions, we need to define an experimentation framework which can be used to conduct the experiments. This is what is outlined in the next section.

3.4 Experimentation Framework

In the previous section we defined a search-space model and the fitness functions that can be used with potential search algorithms considered for evaluation. Here

we look at an experimentation setup which allows us to compare these algorithms. The rest of the thesis is aimed at running various experiments to assess which search approach can best address the practical testing needs for CCI from system level. The experiments range from addressing the basic question of suitability of the search-based approach in this problem domain to the evaluation of a search-based approach that is most suitable for the practical testing needs of CCI that can consider multiple aspects during the search-space exploration and during the fitness evaluation.

Such an experimentation framework is illustrated in Figure 3.3. It allows the experiment to select a configuration that picks a supported search-based algorithm and a corresponding fitness function. Once the experiment is configured (as in step 1 of Figure 3.3) the SBST algorithm will operate on the defined search-space model by selecting a test payload (as in step 2). Once the test payload is selected, based on the search-based algorithm's search-space exploration process, it dispatches the test payload (as shown in step 3) including the evaluation of the relevant neighbouring test payloads one after the other. Upon a test payload being dispatched this will then perform the memory read-write operations (as shown in step 4) that would trigger the functioning of the deeply embedded CCI component on the hardware. The search-based algorithm relies on fitness functions that exploit the hardware counters to determine the 'stress' on the IC component (as shown in step 5). The search-based algorithm continues to explore the search-space guided by the fitness function to maximise the stress on the deeply embedded component. This cycle of operation is repeated till an exit criteria is satisfied. Each test payload evaluation is repeated to account for the unavoidable variances during the computation of fitness function. All these operations are performed on-target when the experiment is conducted on the selected hardware platforms.

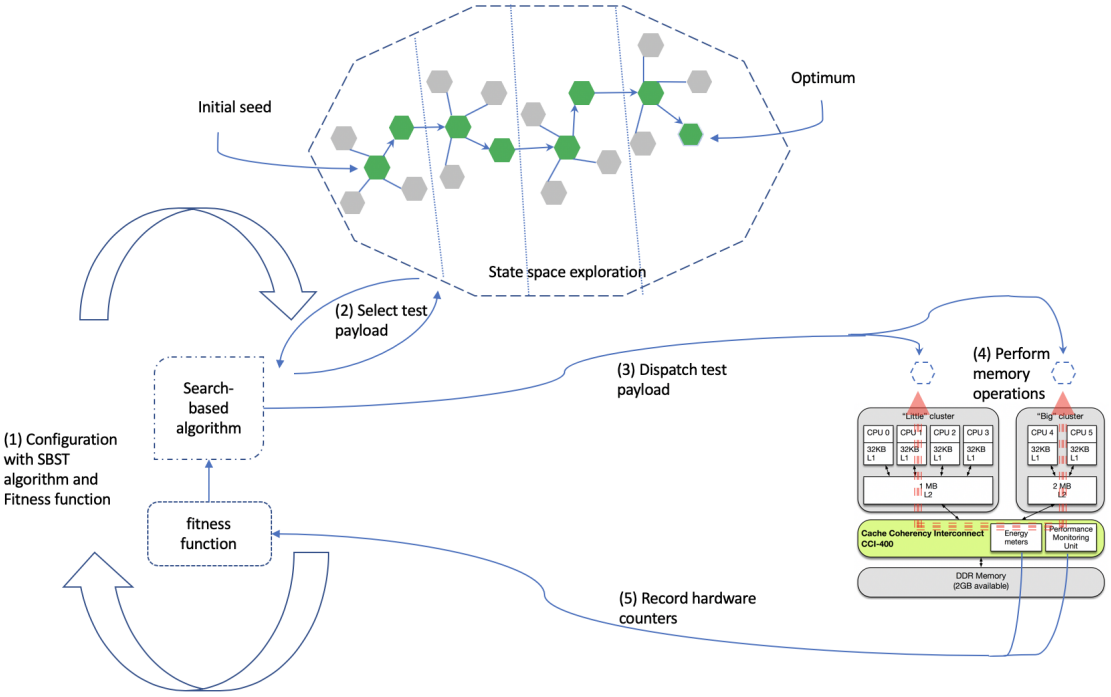

Figure 3.3: Experimental setup with search-based algorithms for CCI testing

The test payload representation as outlined in Section 3.3 has different configurations in the experimental setup based on the target platform selected. These are detailed below for each of the payload attributes:

- Payload Size PS: For our experimental setup, in case of Juno platform, the maximum payload size is configured as 512MB as opposed to 80MB in case of TC2 platform. In case of Juno we have $16 \leq x \leq 16384$ and $16 \leq y \leq 8192$ being allowed as the range for the payload size attribute. However in case of TC2 it is varied in the range of $16 \leq x \leq 5120$ and $16 \leq y \leq 4096$. The payload size limits are determined by the available memory on the selected platform. The lower and upper bounds are set based on its ability to generate data stall cycles, which is observed as part of sensitivity test conducted for the payload attributes. The algorithm will vary these parameters to control the volume of memory read-write operations.

- Sparsity SP: There are 4 levels of sparsity defined: (1) Unconstrained – the payload can be written-to or read-from anywhere in the full range of available memory, (2) relatively sparse – the operational memory is limited to half of the available memory, (3) dense – the operational memory is limited to a tenth of the memory, or (4) very dense – the operational memory is limited to a hundredth of the available memory.

Table 3.2: Sparsity

Platform	Sparsity (in MB)			
	Unconstrained	relatively sparse	dense	very dense
TC2	512	256	51.2	5.12
Juno	1024	512	102.4	10.24

- Actor Profile *AC*: The affinity of the actors of certain type (read or write) to the clusters are fixed for our experiment - read actors are pinned to a big cluster and write actors are pinned to a little cluster, to facilitate the stress test conditions on the CCI component providing coherency across clusters. The significance of pinning different actor types across cluster is give the best chance for the read-write operations to affect the L2 state. This is because, when read and write actors are across clusters, this will force the possibility of cache-misses that results in cache-fetch and cache-flush operations thus stressing the CCI component. The Tables 3.3 and 3.4 show the current configurations which are varied during tests and these are determined based on the CPU topology of the respective platform.

Table 3.3: Actor profile - TC2 platform

AC config	Cluster1 (Little)			Cluster2 (Big)	
	cpu0	cpu1	cpu2	cpu3	cpu4
config1	write	-	-	read	-
config2	write	write	-	read	read
config3	write	write	write	read	read

Table 3.4: Actor profile - Juno platform

AC config	Cluster1 (Little)				Cluster2 (Big)	
	cpu0	cpu1	cpu2	cpu3	cpu4	cpu5
config1	write	-	-	-	read	-
config2	write	write	-	-	read	read
config3	write	write	write	write	read	read

As discussed previously, the contents of the memory and cache are routinely affected by many processes within the system that are difficult to control in the context of the test-application. There are some additional steps taken in the experimentation framework to address this. To attenuate this we carry out the following steps for each test payload evaluation to reduce this potential interference:

- We stop as many Android background tasks as is possible (some cannot be stopped).

- We perform an identical memory walk-through sequence between iterations to give better chances for an equivalent initial state.
- We use data barrier instructions - an Arm architecture specific instruction[Armd] - to ensure all out-of-order data access is cleared before every test payload is evaluated.
- We perform repeated measurements of the candidate test payload to account for the possible variances during fitness function evaluation.

The experimental framework described in the above section is to be executed on a physical test setup. The below section provide details of this physical test setup where these experiments are conducted.

3.4.1 Experimental Setup Details

Throughout this research, for the evaluation of various empirical studies, we relied of two different test systems. First one is a simpler 32-bit system called TC2, which has an Android stack running (also outlined by Eljuse *et al.*[EW16]). In this setup we use performance monitor counters to measure the data stall cycles which gives an indication of the stress on the CCI component and guides the algorithm to device test payloads that maximises the generation of these data stall cycles.

Second test system used is a 64-bit Juno platform (See Section 2.3.2) which also do have an Android software stack (also outlined by Eljuse *et al.*[EW18]). We use Android 7.1.2 (Android Nougat) and its corresponding firmware for the Juno platform. The software images for these test systems are available from Linaro [Lin] which is an open-source non-profit organisation supporting Arm ecosystem.

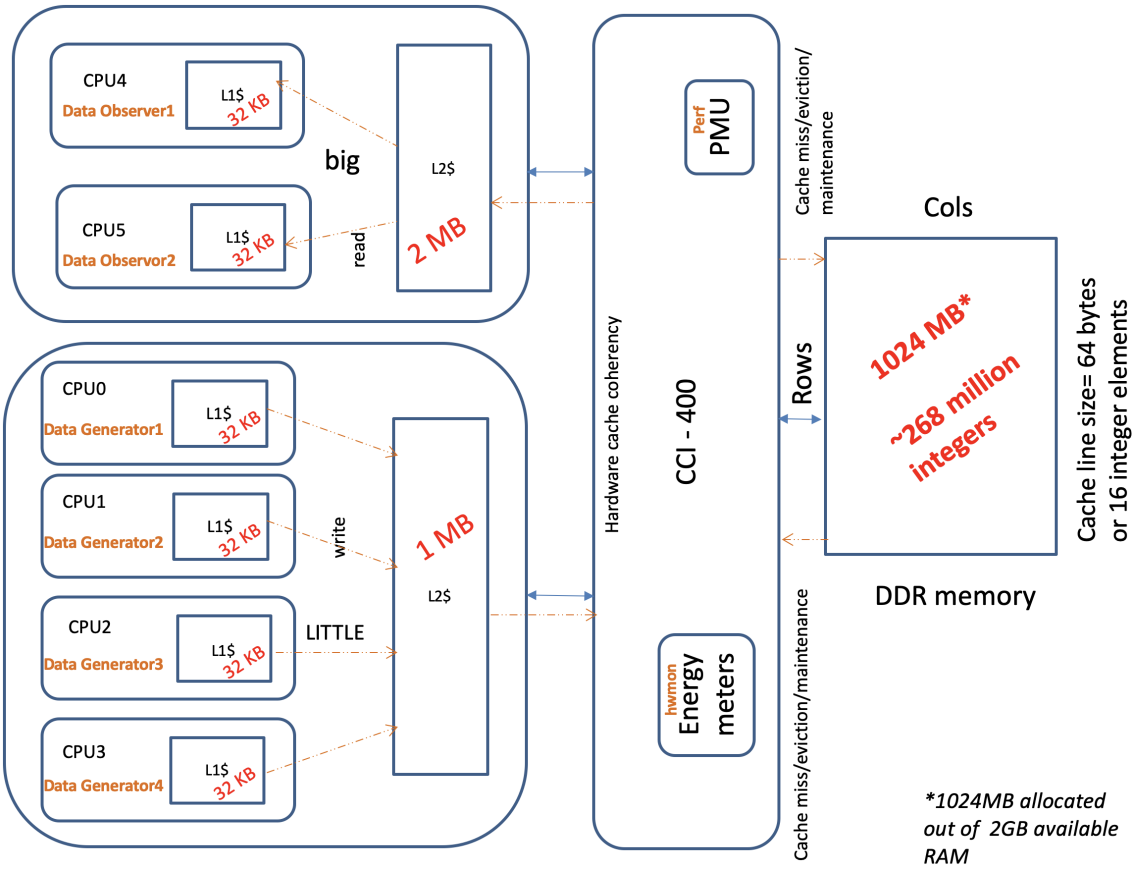

Figure 3.4: Juno platform Details

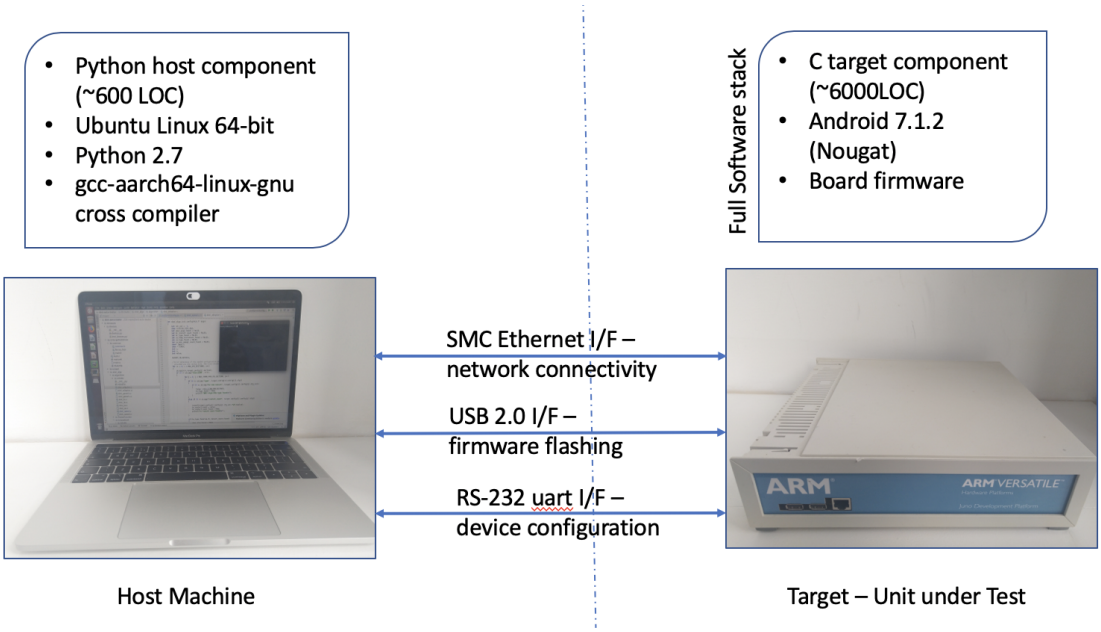

Figure 3.5: Physical Setup

The physical test setup in Figure 3.5 includes a host machine which runs Ubuntu Linux (Ubuntu 14.04 LTS 64-bit) with a host component implemented in Python (600 LOC) performing setup and post-processing logic. The test setup also have a

target component which is the physical board having the CCI component integrated as a complete system (equivalent to a mobile phone). This target component has a full E2E software stack comprising of the board firmware (distributed by Linaro) and Android 7.1.2 software. We implemented a target component (6000 LOC) written in C language which includes the on-target test executor [Elj], the collection of search-based software algorithms and the search-space implementation with the logic for evaluating the fitness function. The details of various software component sizes in this project is detailed in Appendix C.

We perform on-target execution of the test suite from Android user space. The target platform being an embedded system lead us to the selection of on-target execution. We use the Android Debug Bridge (ADB) [Goo] interface to interact with the target device from a host computer. Both TC2 and Juno provide an Universal Serial Bus (USB) interface for updating the board firmware. The target board also provides a serial Universal Asynchronous Receiver Transmitter (UART) based interface for device configuration and also Static Memory Controller (SMC) based ethernet interface as a communication channel for the host machine to interact with during the test execution. The test suite is executed on the target device with results fetched from target to host for post-processing and results analysis.

3.5 Summary

In this chapter we looked at the details of the finalisation of the search-space representation and the associated fitness functions. While the initial 'search grid' approach did not provide useful results the final search-space is modelled based on the 3 key attributes that can affect the stress on the CCI component. A set of suitable fitness functions are also finalised that gets employed in the experimentation framework used by the empirical studies which are detailed in subsequent chapters. It also gives the details of the various components in the experimentation framework that is used to answer the key research questions raised as part of addressing this testing challenge.

The rest of the thesis is trying to find which among the various search algorithm options can best support the generation of automated stress tests for CCI component from a full system context. With the search-space finalised and a suitable experimentation framework defined, we can now move onto a set of empirical studies to address some key research questions in the subsequent chapters.

Chapter 4

Suitability of Search-based Algorithms in IC Testing

In the previous chapter we described how the search-space is formulated in terms of the key attributes that would stress the CCI component. It also outlined the various fitness functions that use the hardware capabilities supported by the test platforms. With the experimentation framework that allows the search-based approaches to evaluate various test payloads as part of the search-space navigation, we can now identify test generation techniques that would maximally stress the CCI component from system level.

In this chapter we describe the first of three experiments, each of which is intended to investigate a different type of test generation approach. We will start by investigating our search-space encoding as the basis for the simplest-possible search algorithm: Hill-climbing. The rest of the chapter is structured as follows.

In Section 4.1 we introduce hill-climbing as a potential search-based approach to be evaluated along with its associated details. Section 4.2 frames the research question and outlines the evaluation methodology followed in this empirical study. In Section 4.3 we discuss the results from the empirical study and Section 4.4 outlines some threats to validity relevant for the experiment. The overall aim of this chapter is to establish the suitability of search-based approaches in this problem domain and this chapter addresses precisely that by using hill-climbing algorithm.

This chapter is shaped by the learning from the research paper presented at SSBSE 2016 (See Page 15), which investigates the potential of search-based approaches in improving the stress testing of IC components. It shared the findings from the empirical study which looked at the use of the proposed search-space representation with a search-based algorithm to improve IC testing.

4.1 Hill-Climbing Approach

Our intuition is to start with a search strategy that is as simple as possible, for which the hill-climbing algorithm is an obvious choice. Hill-climbing algorithm starts from a random solution (in our case a random configuration of our test case representation) and navigates through the search space by continuously selecting a better solution from its 'neighbouring' solutions. The hill-climbing stops once the fitness function score is maximised - either finding a local or global maxima - or when a set amount of iterations is attempted.

The choice of hill-climbing is (at least to begin with) justified because it often performs surprisingly well for other search-based testing problems, often even outperforming more sophisticated evolutionary search techniques [Har07]. Studies show that hill-climbing can be better-off or at least as effective as advanced algorithms in certain problem types. Mona *et al.* [FGG18] showed that for shortest path optimisation problem hill-climbing can be a better option compared to genetic algorithms for certain situations. In case of course time tabling problems also, the studies [RC95] [Mac+10] show that hill-climbing can outperform other advanced algorithms. It is also shown that for node placement in wireless mesh networks [Sak+14] hill-climbing can be a better option compared to advanced algorithms. These studies encouraged us to consider hill-climbing as the first option (keeping in mind its simplicity) for assessing its suitability to address the CCI testing problem.

```
Output: The optimal configuration from the input set
begin
    maximum = 0
    x ← randomInteger(5120)
    y ← randomInteger(4096)
    sparsity ← randomInteger(4)
    actors ← randomInteger(3)
    while fitnessFunction(x, y, sparsity, actors) > maximum do
        maximum ← fitnessFunction(x,y,sparsity,actors)
        Neighbours ← generateNeighbours(x,y,sparsity,actors)
        forall (x', y', sparsity', actors') ∈ Neighbours do
            /* Evaluate fitness of child testdata                    */
            if fitnessFunction(x', y', sparsity', actors') > maximum then
                x ← x'
                y ← y'
                sparsity ← sparsity'
                actors ← actors'
            end
        end
    end
    return (x, y, sparsity, actors)
end
```

Algorithm 1: Dynamic Test Data Generation with Hill-Climbing

An overview of the approach is provided in Algorithm 1. As you can see, the algorithm follows the general hill-climbing approach as described in Page 35. As part of applying hill-climbing approach in order to generate the test payload that would maximally stress the CCI component, we are adapting it to operate on the defined search-space model. Every solution candidate in this search-space model represents a test payload with its own set of values for the 3 attributes (as defined in Section 3.4). The neighbouring nodes in this search-space will have test payloads with variations on one of the 3 attributes while keeping the other 2 same. When hill-climbing algorithm is in operation it will move from one solution candidate to the neighbouring one provided the neighbouring candidate generated higher stress in terms of the measured data stall cycles. This is how search algorithms are implemented to work on this search-space model. Let us look at some of the algorithm details below.

The algorithm uses three auxiliary functions:

- $randomInteger(x)$ provides a random integer between 1 and (including) x. These values are determined based on the memory region allocated for the test application considering the test platform attributes.
- $fitnessfunction(x, y, sparsity, actors)$ evaluates the generated test cases that

conform to the parameters. For the fitness function we have a straight-forward mechanism of using the built-in hardware counters to guide the search algorithm in maximising the stress conditions. These test cases are run multiple times (we opt for 20 times) to have better statistically significant result, and the average number of data stall cycles is returned.

- $generateNeighbours(x, y, sparsity, actors)$ generates all of the possible configurations that are 'adjacent' to the given configuration. For each parameter a number of new test cases are generated. In case of payload size it generates two new test cases where we increase and decrease the size by a pre-defined step value, whilst keeping the other parameters fixed to their original values. In case of sparsity parameter we generate 3 new test cases and finally for actor profile we generate 2 new test cases. This produces, for a given test set, a total of 7 new configurations.

4.2 Methodology

In this section we present the study to evaluate the application of search-based algorithms in stress testing complex ICs. Section 3.3 outlined a search-space representation that can be applied to the CCI testing problem.

In this section we use the search-space characterisation presented previously to investigate the appropriateness of search-based algorithms for stress testing the CCI. We start the investigation into this with simpler algorithms to address RQ1 in 2 sub-parts:

- RQ1a - Does hill-climbing outperform random testing for testing CCI components?
- RQ1b - What is the sensitivity of payload attributes in generating stress on CCI component?

For the evaluation of hill-climbing based approach (as in RQ1a and RQ1b), a 32-bit system called TC2 was relied on which has an Android stack running and detailed in Section 2.3.2. We use performance monitor counters to measure the data stall cycles which gives an indication of the stress on the CCI component and guides the algorithm to device test payloads that maximises the generation of these data stall cycles.

The test input representation as outlined in Section 3.3 has different configurations in the experimental setup based on the target platform. The details of payload attributes as customised for our TC2 setup can be seen in Page 53 for Payload Size and Sparsity and in Page 54 for Actor Profile.

Evaluation Criteria for RQ1a: Hill-climbing vs Random

Shamshiri *et al.* [Sha+15] showed that search-based algorithms do not always outperform random tests. To establish whether this is the case for IC based systems we compare the search-based approach to a random testing baseline.

For the hill-climbing approach, we use the search to select values for the three attributes, using the data stall cycles as the fitness function. Each search started from a random starting point. The hill-climbing algorithm was allowed to continue until it hit a maximum (i.e. no more better solution could be found from the neighbouring candidates from the search-space representation). The level of data stall cycles at the final iteration is reported as the final result. This experiment is repeated 30 times to accommodate the randomness inherent in the initial starting point and the quasi non-deterministic behaviour of the CCI component.

To apply the random-search we repeatedly selected random values for the three attributes (payload, sparsity, and actors), and measured the resulting data stall cycles. For each test we repeated this 10 times (10 was also the maximum number of iterations achieved by any hill-climbing execution, so to be fair the random tests were given same number of iterations), and selected as result the test case that produced the maximum number of data stall cycles obtained over the 10 inputs. Similarly to the hill-climbing experiments, this process was repeated 30 times to account for the inherent randomness. When a random test case is evaluated we recorded the maximum data stall cycles obtained over the 10 repetitions as the result of that individual test case. When hill-climbing results and random test results are compared with each other, it is based on results from the 30 repetition of the respective experiments. Here we use the median values obtained from those 30 experiments to determine which option fared better in generating the data stall cycles.

Evaluation Criteria for RQ1b: Payload Attribute Sensitivity

In order to understand the sensitivity of each of the payload attributes we carried out a follow-up study, varying each of the attributes while keeping the other 2 attributes constant to investigate the effect on the generated stall cycles. The constant value for a given attribute is chosen as the middle point in the range where that could be varied. In case of the payload-size attribute we fixed the actor profile and sparsity attributes and kept varying the size from a minimum (1MB) to a maximum (80MB) value with a predefined step. In case of actor profile we fixed the other two attributes and varied the actor profile against the possible variations of single, dual and multi-actor configurations. Similarly for sparsity attribute we fixed the other 2 and varied the level of sparsity - unconstrained, relatively sparse, dense and very dense - by

setting the maximum value for the x and y coordinates for the memory location addresses being populated.

4.3 Results and Discussion

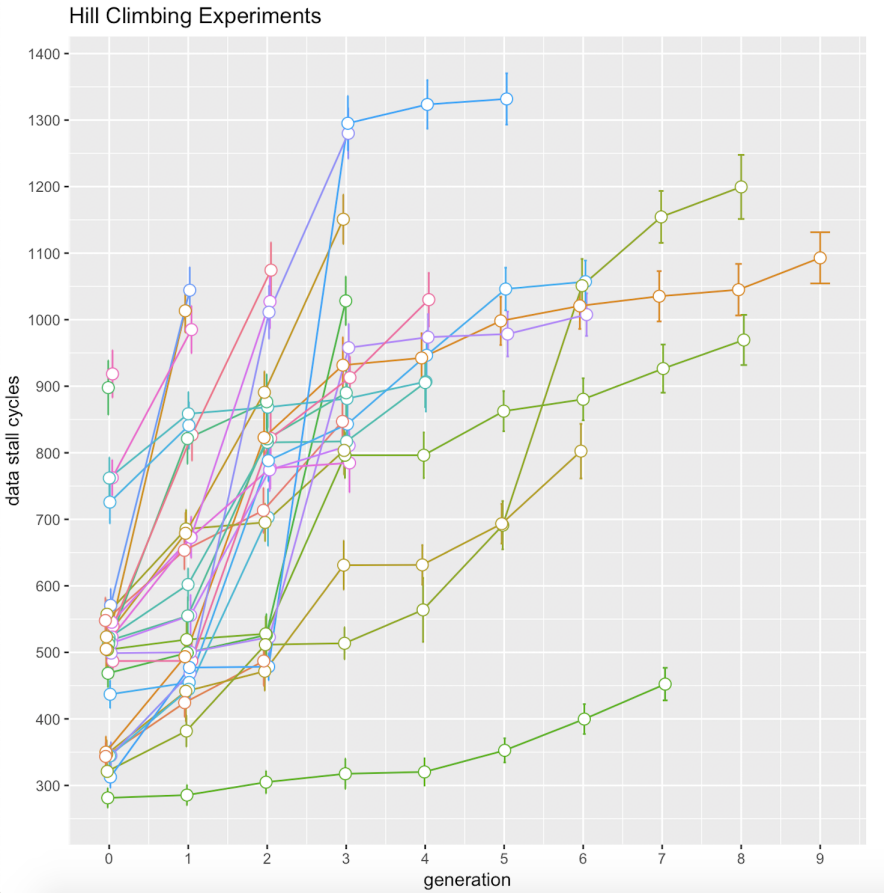

Figure 4.1: Hill-Climbing scores. *30 experiments showing how different experiments hit local maxima*

RQ1a - Does hill-climbing outperform random testing for testing CCI components? Figure 4.1 shows the results of 30 hill-climbing runs (where each 'run' started from a random seed selected by the algorithm). From the runs we observe the algorithm progressively generates better scores as it navigates the search-space. Some runs yielded more than 1000 data stall cycles while other runs generated a lower score. The algorithm evaluated candidate test payloads at each generation and as part of the fitness function evaluation performed repeated measurements of the candidate (see Section 3.4). These repeated measurements are to address the observed variances in recorded data stall cycles that occur as a result of various runtime behaviours affecting the CCI state. We notice that there are some runs which stopped with relatively modest scores below 1000. And there are also some

runs which failed to progress beyond the initial iteration. These observations are noteworthy because they do signify the presence of local maxima.

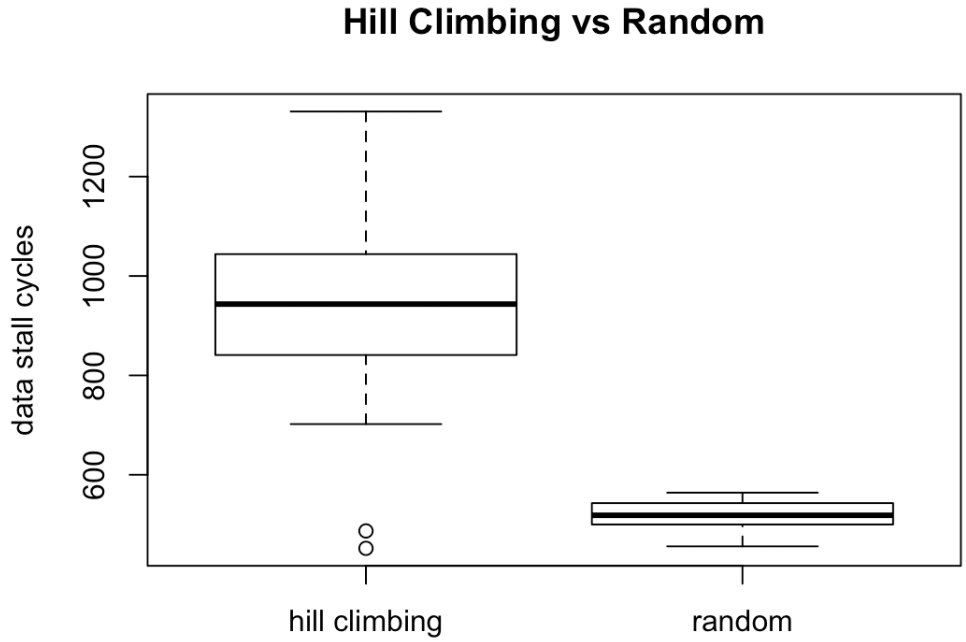

Figure 4.2: Hill-Climbing scores. *Hill-climbing generated better scores than equivalent random test generation experiment*

In Figure 4.2 we compare the hill-climbing results against the random tests. The results show that hill-climbing has a higher median best score of 943 when compared with random tests with 518. The hill-climbing experiment generated a median score which is 1.8 times better than the random tests. While random tests generated a maximum score of 564 the hill-climbing experiment generated a maximum score that is more than 2 times that of random tests at 1331. Even though hill-climbing generates some best scores at the same range as random tests (recorded as the 2 instances marked as outliers) they predominantly outperform the random tests baseline.

RQ1b - What is the sensitivity of payload attributes in generating stress on CCI component? The results of the experiment done to understand the the effect of the 3 individual test payload attributes can be seen in Figure 4.3. It is evident from the Figure 4.3 that the attributes viz, PS, SP and AC have a distinct effect on the ability to generate the data stall cycles and thus the stress on the IC component. We notice that both PS and AC have a similar spread, while SP has the narrowest spread. If we consider the inter-quartile ranges it is evident that PS provides the highest spread and thus has a bigger impact in terms of generating stress conditions when looking at all the 3 attributes in isolation. One natural question we had was whether we would achieve maximum data stall cycles just by

setting maximum the scales for each of these 3 attributes. We inspected the results from hill climbing which generated really higher data stall cycles (as in case of those with really higher scores near 1300) and noted that none of these have any of the individual attributes set to the maximum allowed value. Hence we conclude that it is a combination of changes in each of these attributes that resulted in the higher data stall cycles rather than merely maximising the attributes to their highest possible values.

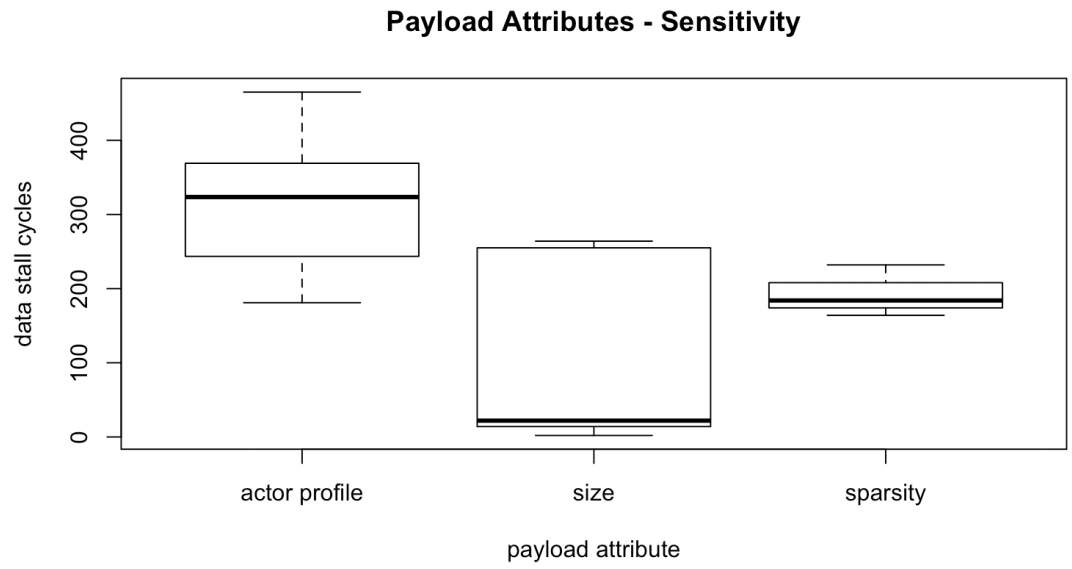

Figure 4.3: Sensitivity of payload attributes. *Each payload attribute evaluated in isolation.*

Conclusion: In this chapter we set out to evaluate whether the hill-climbing search algorithm can be applied successfully to generate stress tests for the CCI component and whether that provides an improvement over random testing. We can see that hill-climbing algorithm out-performed the random testing baseline, which highlights the suitability of search-based testing to this problem domain. From the experiments we also observed that there are runs which hit local maxima.

4.4 Threats to Validity

Threats to Internal validity : A large variation in the measure of data stall cycles could pose a threat to the validity of the results, unless controlled. We observed in our experimentation a large extent of variation in the PMU counter values even when same test data was executed repeatedly. The potential for additional variability is introduced by the fact that the test framework is also running on the target. Also in case of Juno platform the GPU has an L2 cache which can affect the operations

of the CCI-400 component. This threat to validity is addressed in the study by repeating the measurements and reporting the average measurement, rather than reporting instantaneous measurements.

Threats to External validity : In this study we apply SBST to test the CCI component by leveraging the built-in hardware counters like PMU, which is used as the fitness function. This hardware capability is specific to CCI component and we rely on the use of data stall cycles as the event to be counted as part of the fitness function evaluation. If this proposed test methodology is to be applied on other similar IC components, it is crucial to identify a suitable hardware capability that can be used as part of fitness function evaluation. In the absence of such hardware capabilities it may not be possible to apply this test methodology on similar testing challenges, unless other meta-heuristics could be devised as fitness function.

4.5 Summary

In this chapter we detailed the empirical study with a 32-bit Arm platform to assess if search-based testing approach is suitable for addressing the testing challenges associated with deeply embedded components in a system context. From the experiments it is clear that we are able to use search-based algorithm to generate tests that can stress the platform more than the random test baseline. From this experiment we also observed the well-known issue of local maxima that affects local search algorithms such as hill-climbing.

This empirical study opens up the possibility of leveraging the power of search-based algorithms to generate better tests as guided by the built-in hardware counters to improve testing of CCI from system level. This paves way for other search-based algorithms which might provide even further benefits and can alleviate the limitations observed with the simpler search-based algorithms.

Chapter 5

Applying Advanced Search-based Algorithms in IC Testing

In the previous chapter we assessed the basic applicability of search-based approaches to testing IC components from system level. The empirical study indicates that a simple search algorithms like hill-climbing outperforms random testing. However it also highlighted the issue of local maxima, which can hamper algorithms such as hill-climbing. In this chapter we investigate more advanced search-based algorithms which can potentially overcome local maxima problem and evaluate what benefit these can provide in IC testing challenge addressed as part of this research.

We start with Section 5.1 where we introduce random-restart hill-climbing and simulated annealing as 2 candidate algorithms that can address local maxima issue. In Section 5.2 we frame the research questions to be addressed in this empirical study and outline the evaluation methodology to be followed. In Section 5.3 we detail the results and list a relevant set of threats to validity (in Section 5.4) against which these result are to be considered.

This chapter is developed based on the insights gained from the research paper presented at SSBSE 2018 (See Page 15), which explores options from search-based algorithms that can address limitations faced when using local search algorithms in testing IC components. It shared the findings from the empirical study which investigated how random-restart hill-climbing and simulated annealing algorithms could address the local maxima problem noticed from earlier research.

5.1 Use of Advanced SBST Algorithms

As seen from Chapter 4, we established that indeed SBST methodologies are suitable for application in the testing of complex IC components like CCI. However the evidence also confirmed the limitations of simpler search-based algorithms which

leads us to the investigation of more advanced search-based algorithms that could possibly overcome these limitations. This chapter focuses on the application of some of the advanced algorithms addressing these limitations.

Random-Restart Hill-Climbing Algorithm: The random-restart variant of hill-climbing algorithm provides a means to escape a local maxima once encountered, by restarting the algorithm from a new random starting point with the aim of arriving at a better solution [Jac02]. It retains the simplicity of hill-climbing algorithm but is much more effective in the case of search-spaces with the resultant fitness landscape having multiple local maxima present.

```
Input:
maxrestart - maximum number of restarts allowed
Output: The best solution from all restarts
begin
    globalBest ← null
    maximum = current = restarts = 0
    while restarts < maxrestarts do
        candidate ← randomSeed()
        current = 0
        while fitnessFunction(candidate) >= current do
            current ← fitnessFunction(candidate)
            Neighbours ← generateNeighbours(candidate)
            for (candidate′) ∈ randomNeighbour(Neighbours) do
                if fitnessFunction(candidate′) > current then
                    candidate ← candidate′
                    current ← fitnessFunction(candidate′)
                    break;
                end
            end
        end
        if current > maximum then
            globalBest ← candidate
            maximum ← current
        end
        restarts ← restarts + 1
    end
    return globalBest
end
```

Algorithm 2: Dynamic Test Data Generation with Random-Restart Hill-Climbing

An overview of the approach is provided in Algorithm 2. As you can see, the algorithm follows the general random-restart hill-climbing approach as described in Page 35. The random-restart hill-climbing operates on the search-space with

the aim of finding a solution candidate that will maximise the stress on the CCI component. Every solution candidate in this search-space model represents a test payload with its own set of values for the 3 attributes (as defined in Section 3.4). The neighbouring nodes in this search-space will have test payloads with variations on one of the 3 attributes while keeping the other 2 same.

When random-restart hill-climbing algorithm is in operation it moves from one solution candidate to the neighbouring one provided the neighbouring candidate generated higher stress in terms of the measured data stall cycles until it reaches a solution candidate where none of its neighbours are better than the current one. This signifies that the random-restart algorithm has reached a maxima which can be local. At this point it would save the solution candidate. If the termination condition is not achieved the algorithm will restart the search from a new random starting solution candidate. Once the termination condition is achieved it would return the best solution candidate discovered over the various restarts it has attempted.

The ancillary functions are identical to conventional hill climbing (see Section 4.1), apart from the following:

- *randomSeed* Selects a random starting candidate for the random-restart hill-climbing search. Here the seed represents the test payload which has its 3 attribute representing size, sparsity and actor profile all set to a random value.
- $randomNeighbour(Neighbours)$ Selects a non-repeated random neighbour from the available list of neighbours. This maximises the ability of the algorithm to explore the search-space as this forces a different possible neighbouring candidate to be evaluates during the state-space exploration.

Simulated Annealing Simulated Annealing is another advanced search-based algorithm explored to see if it could provide improvements over local search algorithms in testing IC components.

An overview of the approach is provided in Algorithm 3. As you can see, the algorithm follows the general simulated annealing approach as described in Page 36. The simulated annealing algorithm will operate on the search-space with the aim of finding a solution candidate that will maximise the stress on the CCI component. Every solution candidate in this search-space model represents a test payload with its own set of values for the 3 attributes (as defined in Section 3.4). The neighbouring nodes in this search-space will have test payloads with variations on one of the 3 attributes while keeping the other 2 same.

Input:
tempstart - start temperature
tempend - final temperature
tempstep - unit of increment in temperature
$flooringratio$ - means to control sub-optimal candidate selection
repeatcount - the number of times experiment iterates at a given temperature before reducing
Output: The best solution
begin
  $globalBest \leftarrow null$
  $candidate \leftarrow$ `randomSeed()`
  $maximum \leftarrow current \leftarrow$ `fitnessFunction`$(candidate)$
  while $tempstart > tempend$ **do**
    $repeats = repeatcount$
    while $repeats > 0$ **do**
      $Neighbours \leftarrow$ `generateNeighbours`$(candidate)$
      $candidate' \leftarrow randomNeighbour(Neighbours)$
      $fitnessscore \leftarrow$ `fitnessFunction`$(candidate')$
      if $fitnesscore > current$ **then**
        $candidate \leftarrow candidate'$
        $current \leftarrow fitnessscore$
        if $current > maximum$ **then**
          $maximum \leftarrow current$
          $globalBest \leftarrow candidate$
        end
      else
        $flooringscore \leftarrow maximum * flooringratio$
        if
         (`acceptanceProbability`$(fitnessscore, current, tempstart)$)
         & $(fitnessscore > flooringscore)$ **then**
          $candidate \leftarrow candidate'$
        else
          $repeats \leftarrow repeats - 1$
        end
      end
    end
    $tempstart \leftarrow tempstart * tempstep$
  end
  return $globalBest$
end

Algorithm 3: Dynamic Test Data Generation with Simulated Annealing

When simulated annealing algorithm is in operation it will move from one solution candidate to the neighbouring one (as part of the state-space exploration) with the aim of maximising stress on CCI. During the early part of the execution, simulated annealing algorithm can select a 'sub-optimal' solution candidate (meaning this candidate generate lesser stress on CCI component than current candidate) based on the temperature (which will be high at the early part of execution) and favourable acceptance probability. This possibility of selecting search paths with sub-optimal solution candidates enables the simulate annealing algorithm to avoid local maxima as it performs wider search-space exploration. However as the execution progresses the temperature degrades and this in turn makes the possibility of selecting sub-optimal candidates harder. At the later parts of the execution the simulated annealing will keep searching for new solution candidates that will only generate better solution candidates than the current one. Once the termination condition is achieved the algorithm will provide the best solution candidate found. Let us look at some of the algorithm details below.

The algorithm uses the following auxiliary functions, in addition to the ones defined in Algorithm 2.

- $acceptanceProbability(newscore, oldscore, temperature)$ determines the acceptance probability of the newscore based on computing $ap = e^{costdifference/temperature}$. Here cost difference is based on the newscore and oldscore. This computed ap is compared against a randomly generated number between 0 and 1 and if found greater then the function will return True.

The various parameters for Simulated Annealing algorithm are set as below for the current experiment:

- *tempstart* the algorithm starts with an initial temperature of 1.0.
- *tempend* the algorithm stops when the temperature reaches 0.1.
- $flooringratio$ the algorithm uses a flooring ratio of 75% and applies this on the maximum achieved score, to compute the flooring score that will control the sub-optimal node selection.
- *repeatcount* the algorithm repeats with new neighbouring candidates for 10 times before reducing the temperature.
- *tempstep* the algorithm degrades the temperature by 10% after it has exhausted the number of repeats at a given temperature.

The Simulated Annealing parameters selected above are the same used by Eljuse *et al.* in [EW18]. The temperature start, temperature stop and temperature degradation step are selected based on initial experiments as most of the cooling schedules for Simulated Annealing are problem specific. Some studies as in [PK98] provide a systematic approach for parameter selection, but most practical cases we did experimentation to arrive at the current selection of these parameters. In these experiments we varied each of the 3 temperature configurations from a predefined set and observed for results that maximises the generated data stall cycles. The aim is to select a parameter set that would progressively provides higher data stall cycles early enough within the execution budget.

We initially had no *flooringratio* defined, and this frequently led to situations where the algorithm would fail to sustain any improvements in fitness. The initial gains could be lost in later part of the experiment due to selection of sub-optimal candidates which were not favourable overall. The floor limit of 75% of the maximum score is achieved by experimentation where we looked for a value that could control the selection of sub-optimal candidates which sustains the generated data stall cycles. Similar experimentation is done to arrive at the value for *repeatcount*. A main consideration given while experimenting with repeatcount is to ensure that we selected a value which gave sufficient opportunity for the algorithm in selecting sub-optimal candidates - thus allowing more exploration of the search-space at a given temperature - before degrading the temperature.

5.2 Methodology

RQ2 investigates whether advanced search-based algorithms can improve on the simpler search-based algorithms, especially when applied to the CCI testing challenge:

- RQ2 - Can algorithms that avoid local maxima (specifically Random-restart hill-climbing and Simulated Annealing) outperform simple hill-climbing on testing CCI components?

For RQ2 a 64-bit system called Juno was used, as detailed in Section 2.3.2. We selected Juno as the target platform for this experimentation as it was the newest hardware platform with CCI400 and has support for on-board energy meters in addition to PMU counters. Switching to Juno is a natural step forward when we look beyond the current evaluation of advanced search-based algorithms and focus on multi-objective algorithms subsequently. As was the case in RQ1, we use PMU counters to measure the data stall cycles which gives an indication of the stress on CCI component and guides the algorithm to devise test payloads that maximises the generation of these data stall cycles.

The test input representation as outlined in Section 3.3 has different configurations in the experimental setup based on the target platform. The details of payload attributes as customised for our Juno setup can be seen in Page 53 for Payload Size and Sparsity and in Page 54 for Actor Profile.

Evaluation Criteria for RQ2: Random-Restart Hill-Climbing vs Simulated Annealing vs Hill-Climbing

For this experiment we use hill-climbing as a baseline and compare it against the performance of Random-restart Hill-Climbing and Simulated Annealing. All algorithms were allocated a fixed search budget of 2 hours.

The random-restart hill-climbing algorithm is allowed to restart as many times as possible within the allocated search budget, while Simulated Annealing would continue to select sub-optimal search candidates as allowed by the acceptance criteria at each temperature until either the execution budget is exhausted or the minimum temperature is achieved. The simple hill-climbing algorithm executes until either the allocated execution budget is exhausted or it has reached a local maximum and there are no more neighbouring solutions in the search-space to explore any further. Similar to RQ1, these experiments are repeated 30 times to account for random variations.

We measure, as in RQ1, the maximum number of stall cycles achieved by each approach. However, for RQ2 we also wish to compare the efficacy of the two advanced search-based algorithms. For this we devised two additional metrics - Area Under the Curve (AUC) and score gain.

To distinguish between algorithms in terms of their efficiency, we compared the AUC formed by plotting the stall cycles at each generation. The rationale is that, if two curves produce approximately similar final-scores, the curve with a greater area under the curve will have achieved this score more rapidly (this is similar in nature to the Average Percentage Faults Detected (APFD) measure used to evaluate test-prioritization performance [Rot+99]). For a curve spanning n generations, where s_x represents the stall cycles recorded at generation x, the area under the curve is computed as $auc_n = \Sigma_{i=0}^{n} s_i$. The AUC measure is used specifically to compare random-restart hill-climbing with simulated annealing as we want to find out amongst these two advanced algorithms which one can generate higher data stall cycles quicker. Remember that the two advanced search-algorithms can take search paths that might lead to sub-optimal candidates or local maxima, but will eventually reach a more optimal solution rather than being stuck at local maxima. With AUC measurement we look at how quickly the (quicker the better) optimal solution is arrived at by random-restart hill-climbing or simulated annealing. As this is not relevant to simple hill climbing (as it progressively takes the search path that leads

to the next best solution only) it is excluded from the AUC based comparison.

To account for the fact that different curves start from different initial starting-points, we also define a gain factor:

$$gain = \frac{max\_score - min\_score}{min\_score}$$

. This should tell us to what extend the algorithm was able to improve the measured data stall cycles from the solution candidate giving the lowest score to the eventual best score. This is applied to all the 3 algorithms during the comparison.

To establish whether the sets of results for different search algorithms are different, we test for statistical significance. We expect the measurements to follow non-normal distribution due to the lack of control of the run-time states, which influence the generated measurements. Accordingly we use the non-parametric Wilcoxon signed-rank test.

In addition to the statistical significance, we also measure the effect size to gauge the scale of difference. For this we use Standardised Mean Difference (SMD):

$$SMD = \frac{Difference\ in\ mean\ outcome\ between\ groups}{Standard\ Deviation\ of\ outcome\ among\ participants}$$

. The effect size should remove the influence of sample size, which often has influence in the case of computed p-values denoting statistical significance, thus giving us additional insight into the comparisons being made.

5.3 Results

A fixed execution budget of 2 hours is given for all the algorithms. However we note that the 'simple' hill-climbing took an average execution time of 1 hour 28 minutes across the 30 repeated executions, owing to the fact that some runs hit local maxima early on within the allocated search budget.

Figure 5.1 shows the date stall cycles achieved by 'simple' hill-climbing (which is acting as the baseline) versus random-restart hill-climbing and Simulated Annealing. Simulated Annealing has a similar median best score of 1250 as opposed to 1251 of random-restart hill-climbing. Both outperform the common baseline of simple hill-climbing which gave a median best score of 1205. With this we can establish that both random-restart hill-climbing and Simulated Annealing are better algorithm choices. However it is worth to note that the scale of improvements in RQ2 is relatively modest compared the scale of improvements hill-climbing achieved over 'pure' random testing in RQ1.

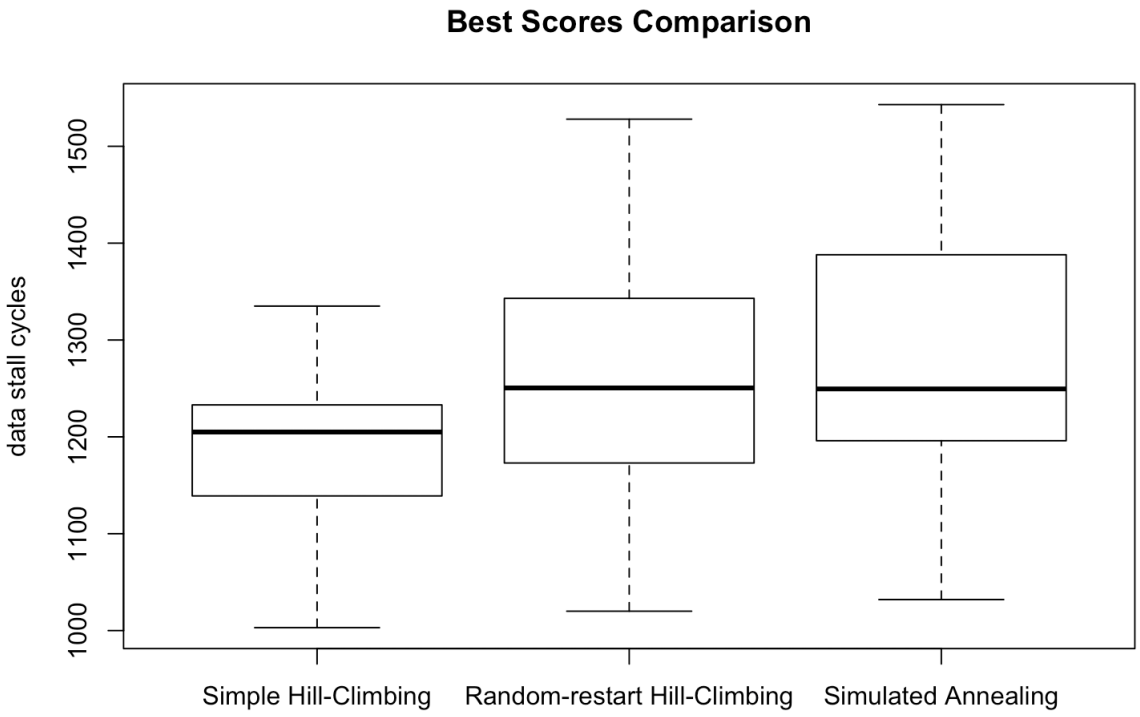

Figure 5.1: Data stall cycle comparison *Random-restart Hill-Climbing vs Simulated Annealing vs Simple Hill-Climbing*

As seen from Table 5.1 we can see a 'medium effect' [Coe02] for comparisons between Random-Restart Hill-Climbing (RRHC) against Simple Hill-Climbing (HC) and Simulated Annealing (SA) against HC. The results indicate a statistically significant difference between HC and RRHC, and between HC and SA. However, there is no statistically significant difference between RRHC and SA.

Table 5.1: Wilcoxon signed-rank test (p) & Effect Size (es) - Data Stall Cycles

	p	es
Simple HC vs RRHC	0.03325	-0.06
Simple HC vs SA	0.02607	-0.73
RRHC vs SA	0.9	-1.0

As for the score gain factor the results are shown in Figure 5.2. The median score gain achieved in case of RRHC is 36% compared to the initial score. Looking at the Simulated Annealing approach we can see moderately better median score gain of 41% while simple hill-climbing shows a median gain of 38%. When comparing the score gain alone it is not evident if a particular algorithm fares better than the other

as the difference is marginal and the spreads do considerably overlap.

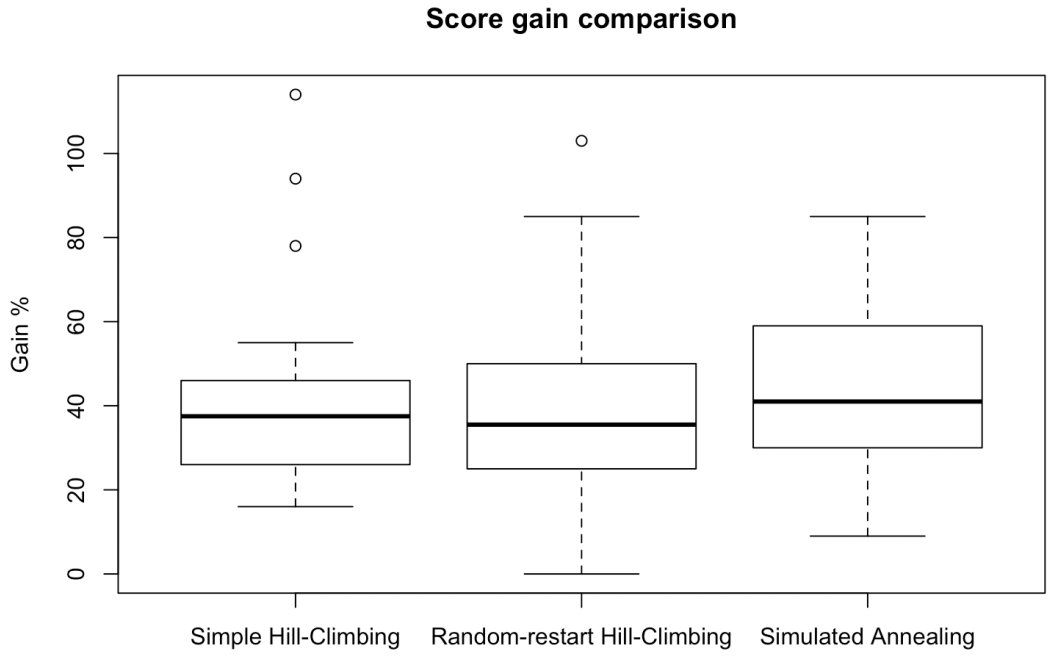

Figure 5.2: Gain comparison *Random-restart Hill-Climbing vs Simulated Annealing vs Simple Hill-Climbing*

Analysing the Wilcoxon signed-rank test results for score gain from Table 5.2, there are no statistically significant differences between the approaches. We can see a 'small effect' [Coe02] between RRHC and SA. When we compare RRHC against HC and SA we see no real effect. Accordingly, by virtue of score gain measurement alone it is not possible to consider one algorithm better than the others.

Table 5.2: Wilcoxon signed-rank test (p) & Effect Size (es) - score gain & AUC

	score gain		AUC	
	p-value	effect-size	p-value	effect-size
RRHC vs SA	0.2486	-0.25	0.6757	-0.7
Simple HC vs RRHC	0.5944	0.16		
Simple HC vs SA	0.5393	-0.09		

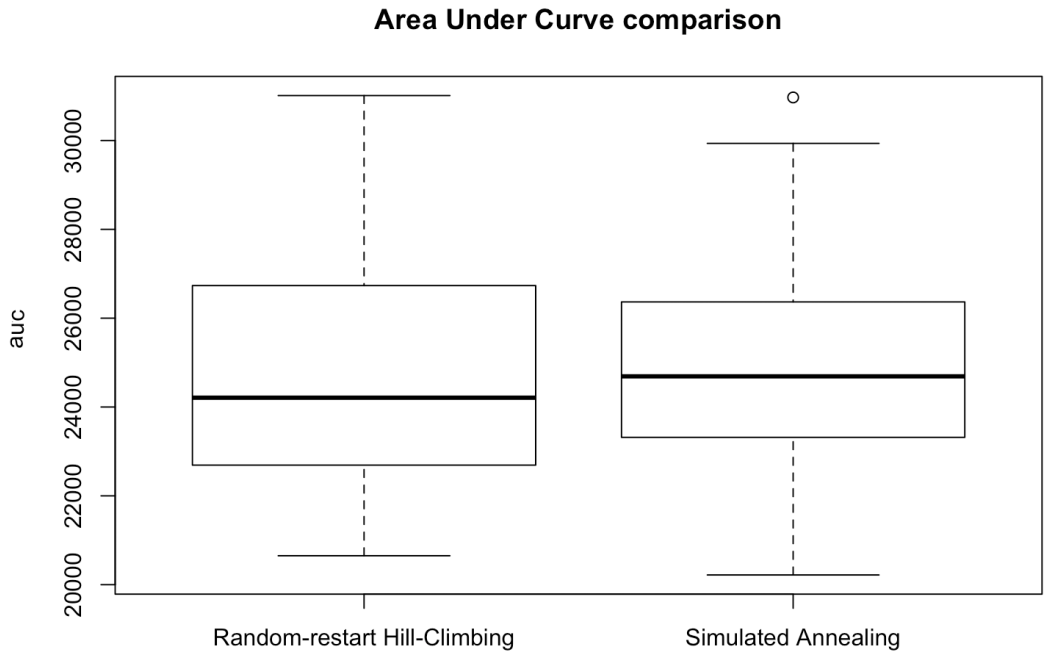

Figure 5.3: Area under curve for best scores found across payloads evaluated.

Figure 5.3 compares the AUC for these 2 advanced algorithm variants to assess how quickly these could generate higher data stall cycles. The results show that Simulated Annealing has a moderately higher median area under the curve of 24690 when compared with random-restart hill-climbing which has 24209.

Figure 5.4 shows how the optimal solutions are converging for each algorithms as seen from the area under the curve measurements. We analysed how quickly an algorithm was converging on the optimal solution in the experiment. From the figure we can see that RRHC converges earlier in the 2 hour fixed budget at 52 min (median value across 30 experiments) however ends up with a relatively lower median for the overall AUC measurement(when compared against Simulated Annealing). In case of Simulate Annealing we observe that the optimal solution is converged much later (at 104 min) than RRHC. However it converges to a better solution than RRHC, which is reflected in the larger median AUC value for Simulated Annealing.

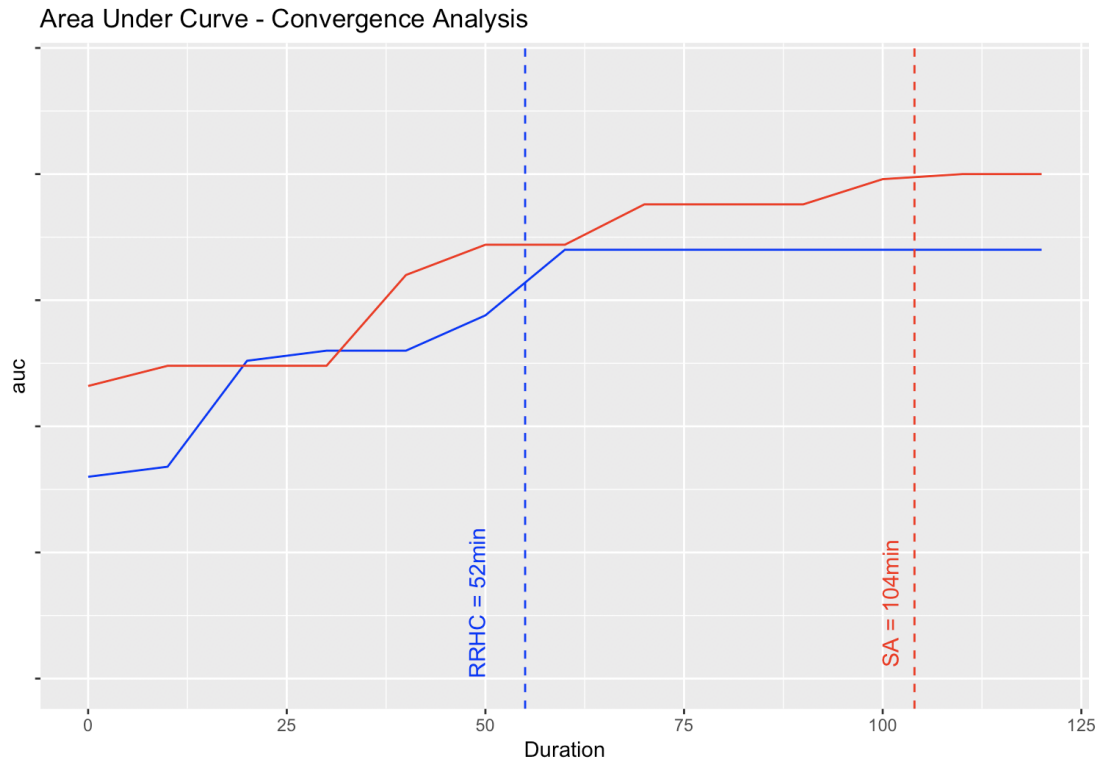

Figure 5.4: Area under curve - *RRHC vs SA - convergence analysis.*

From these results we conclude that both Simulated Annealing and random-restart hill climbing perform similarly with comparable median AUC scores. From the Table 5.2 the p-value of 0.6757 indicates that this is again not statistically significant result to assert one algorithm is better than the other based on the area under the curve metric comparison.

Conclusion: The current search-space is shaped in such a way that local-maxima are problematic. More advanced search algorithms like random-restart hill-climbing and Simulated Annealing are capable of generating higher stress conditions when compared with simple hill-climbing, but the improvements are relatively small. The relative benefits of adopting random-restart hill-climbing or Simulated Annealing are similar, with a marginal advantage to Simulated Annealing.

5.4 Threats to Validity

Threats to Internal validity : RQ2 shares the same threats to validity as outlined for RQ1 in Section 4.4.

Threats to External validity : RQ2 shares the same threats to validity as outlined for RQ1 in Section 4.4.

5.5 Summary

In this chapter we detail an empirical study done on a 64-bit Arm platform to assess if advanced search-based algorithms can overcome limitations of simpler search-based approach in stress testing CCI. While the study shows marginal improvements with advanced search algorithms compared to simple hill-climbing, it is hard to conclude whether one of the advanced algorithms is better than the other.

This empirical study reiterates the suitability of applying more advanced search-based algorithms in CCI testing. So far we looked at algorithms that relies on a single objective fitness function. We know that the stress on a complex IC component in its operational setup can be affected by and be manifested in multiple ways in a full system context. This paves way for investigating how multi-objective algorithms can be used for testing IC components from a system context which is discussed in the next chapter.

Chapter 6

Improving IC Testing with Multi-Objective Algorithms

In the previous chapters we established the suitability of search-based algorithms in stress testing IC components from system level. These focused on search algorithms relying on single objective fitness function. In a complex component like CCI, the stress can manifest in multiple ways and be measured using many different mechanisms. To adequately test ICs in terms of these various measures, search-based IC testing will have to employ search algorithms that are capable of optimising multi-objective fitness functions. This chapter explores the adoption of multi-objective search algorithms in the testing of IC components from a system level.

In Section 6.1 we argue why SPEA2 along with a variant of Simulate Annealing using composite fitness function can be a suitable multi-objective algorithm choice for solving this problem. In Section 6.2 we present the empirical study along with the evaluation methodology followed. Section 6.3 details the results along with the threats to validity.

This chapter is developed based on the insights gained from the STVR journal publication (See Page 15), which explores how multi-objective algorithms can be effectively used in IC testing. It builds on the findings from the empirical study, which applies SPEA2 and a variant of Simulated Annealing using a composite fitness function that maximises multiple objectives while generating better stress tests for IC components .

6.1 Use of Multi-Objective SBST Algorithms

The stress on a component like CCI can be affected by many factors and can manifest itself in a multitude of ways. In a practical scenario when we test CCI at system level, we would want to consider these multiple aspects. This chapter focuses on the

application of multi-objective search-based algorithms to this testing problem and investigates whether this can improve IC testing. Let us first look at the algorithms that are used as part of this empirical study.

Input:
p_sz - Size of the Population
a_sz - Size of the Archive Population
mp_sz - Size of the Mating Pool
$P\_crossover$ - Probability for crossover operation
$P\_mutation$ - Probability for mutation operation
Output: Nondominate Solution Set
begin
  $Population \leftarrow initialisePopulation(p\_sz)$
  $Archive \leftarrow null$
  $MatingPool \leftarrow null$
  while $\neg terminationCondition()$ **do**
    $evaluateObjectives(Population)$
    $fitnessAssignment(Population, Archive)$
    $Archive' \leftarrow getNonDominate(Population, Archive)$
    if $sizeOf(Archive') > a\_sz$ **then**
      $truncateArchive(Archive')$
    else
      $Archive' \leftarrow copyDominate(Population, Archive)$
    end
    $MatingPool \leftarrow tournamentSelection(Archive')$
    $Population \leftarrow$
     $crossoverAndMutate(MatingPool, P\_crossover, P\_mutation)$
  end
  return $getNonDominate(Population, Archive')$
end

Algorithm 4: Dynamic Test Data Generation with SPEA2

SPEA2 is one of the prominent elitist population based multi-objective search algorithms [ZLT01]. It is proven to be more efficient than other alternatives [ZLT01], which made us choose this as option for the multi-objective search algorithm experiment.

SPEA2, like other population based evolutionary algorithms is classed as an extension of Genetic Algorithms. To apply it, it is necessary to define an encoding, the genome, the crossover and the mutation operations. In this research we do represent the search-space in terms of the same three attributes used in previous experiments: PS, SP and AC (See Section 3.3).

Any individual in the population is defined based on the three payload attributes: PS, SP and AC. For crossover operation we define a 'single-point crossover', where the the parent candidates will swap the respective payload attributes (See Fig 6.1). We apply a crossover probability and when that is favourable the crossover operation

is performed. In case of mutation we defined specific mutation operators for each of the individual payload attributes. When mutation happens based on the mutation probability being favourable the following can occur:

- *PS* mutation operation - we increment the size attribute with a predefined step as mutation operation. The step size is set as 1024.
- *SP* mutation operation - we change the SP assignment to a value different from the current one based on the possible list of: Unconstraint, Relative sparse, Dense and Very Dense (See Table 3.2).
- *AC* mutation operation - we change the AC assignment to a value different from the current one based on the possible list of: config1, config2 and config3 (See Table 3.3 and Table 3.4).

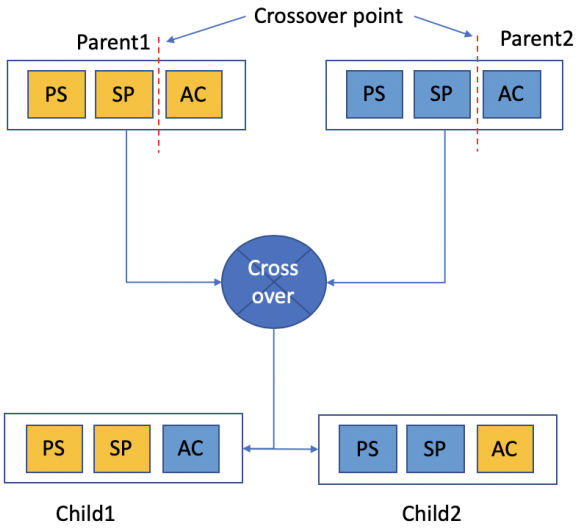

Figure 6.1: Crossover Operation

An overview of the approach is provided in Algorithm 4. The algorithm follows the general SPEA2 approach as described in Page 36. SPEA2 starts from an initial population with random solution candidates. SPEA2 evaluates the fitness of the individual solution candidates as per a fitness assignment strategy described by Zitzler *et al.* which takes into account the dominance of the solution candidate in the population and the computed density function [ZLT01]. Based on the fitness assignment it will generate a non-dominating set of solution candidates (See Page 37) as the archive population. From the archive population SPEA2 performs generational evolution by creating a mating pool based on the tournament selection method [BT96]. From this mating pool it evolves the next generation by performing crossover and mutation operation. The generational evolution is repeated until the termination condition is reached at which point the non-dominating set of solutions are returned.

The algorithm uses the following auxiliary functions:

- $initialisePopulation(p\_sz)$ initialises the population with a random set of candidates equal to the p_sz.
- $terminationCondition()$ evaluates the condition for terminating the experiment.
- $evaluateObjectives(Population)$ evaluates the multiple objectives set for each candidate in the population.
- $fitnessAssignment(Population, Archive)$ performs the fitness assignment to each candidate considering both raw fitness score and density function.
- $sizeOf(Population)$ computes the number of candidates in the population.
- $truncateArchive(Population)$ truncates the Archive population considering the distance to nearest candidate to maintain the Archive size.
- $copyDominate(Population, Archive)$ copies the dominate candidates from population to the Archive to maintain the Archive size.
- $tournamentSelection(Population)$ performs binary tournament selection with replacement to fill the mating pool for generational evolution.
- $crossoverAndMutate(Population, P\_crossover, P\_mutation)$ performs crossover operation from the mating pool based on the crossover probability set and performs mutation on the child candidate based on mutation rate set.
- $getNonDominate(Population, Archive)$ returns the non-dominate set of solutions which forms the solutions in the pareto front.

6.2 Methodology

We know that the CCI component's behaviour can be affected by multiple aspects and the stress on the component can manifest themselves in multiple ways. This raises the need to look at multi-objective search-based algorithms as outlined in Section 2.5:

- RQ3 - Does multi-objective search algorithm generate better stress tests compared to single objective search algorithms when testing CCI components?

Evaluation Criteria for RQ3: Multi-objective vs Single objective search

For SPEA2, we use data stall cycles and energy measurement as the two objectives of the multi-objective fitness function.

For our single objective search algorithm, we opt for the same Simulated Annealing algorithm used for RQ2. However, to ensure that there is a suitable comparison, we provide two versions – one fitness function for each of the objectives used in the SPEA2 implementation:

- SA.PMU - where Simulated Annealing uses data stall cycles as the fitness function.
- SA.ENERGY - where Simulated Annealing uses energy measurement as the fitness function.

As we are trying to compare the single objective search algorithm against the multi-objective version, we adapted the simulate annealing algorithm to start recording the a secondary fitness function in each of the above 2 versions. With this adaptation, Simulated Annealing will maximise the 'primary' fitness function but along with that will report a secondary fitness function at every iteration. Do note that the secondary fitness function does not drive the state-space exploration in this single objective case.

Typically multi-objective algorithms aim to optimise problems affected by competing objectives. In the current study we have both data stall cycles and energy measurement as the 2 objectives. In case of CCI testing both these objectives are not considered as directly linked but rather orthogonal to each other. These measures are not competing objectives in case of CCI because when more data stall cycles occur naturally this means CCI performing more operations which then translate to higher energy consumption.

Various methods for comparison of multi-objective and single-objective search algorithms exists [IND06], including the use of scalar fitness functions, where multiple fitness scores are combined with a weighting factor. We extended the use of Simulated Annealing algorithm to define a composite fitness function where equal weightage is given to both data stall cycles and energy measurement scores. We call this as SA-Composite algorithm variant. This variant transforms the Simulated Annealing algorithm to guide the search-space navigation by considering multiple objectives. Hence SA-Composite is treated as part of the multi-objective option along with SPEA2 when evaluated against the single objective variants.

Let us discuss the details of the methodology we devised for the comparisons used as part of RQ3.

Since the multi-objective SPEA2 algorithm would give a number of solutions at the pareto front (equal to the archive population size) instead of a single solution,

we save the top 'n' solutions from Simulated Annealing run. Here 'n' is limited by the archive population size of SPEA2 algorithm. This approach is taken for both versions of the Simulated Annealing experiment (SA.PMU and SA.ENERGY) where they are treated as the single objective baselines and for SA.Composite where it is treated as part of the multi-objective option evaluated.

Now with a group of solution candidates available for comparison from both multi-objective algorithm and from the single objective versions, we can use the recorded fitness function values for this purpose. We compute AUC measurement to compare the algorithm variants. Figure 6.2 shows how the AUC from 2 sets of solutions from the pareto front can be used in comparisons, where f1 and f2 are the two fitness functions.

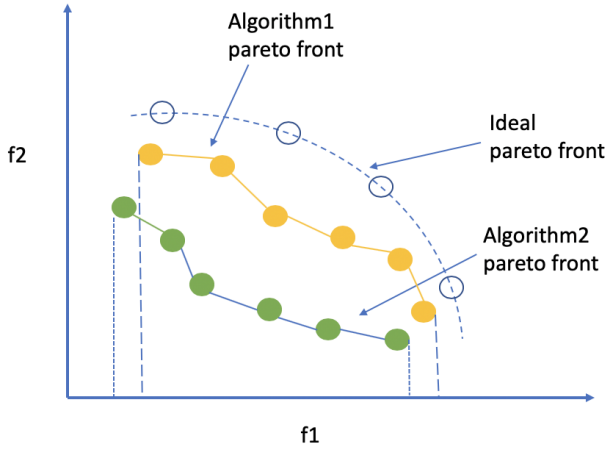

Figure 6.2: AUC comparison based on solutions at pareto front: AUC(Algorithm1) > AUC (Algorithm2)

We expect SPEA2 to have a larger AUC as it should maximise both energy and data stall cycles as opposed to the baselines, which are maximising a single objective. The same behaviour from SPEA2 is expected for the SA.Composite algorithm variant. Another reason is for SPEA2's ability to avoid clustering of solutions at the pareto front. Similar to RQ2, we also check for the statistical significance of the comparisons and compute the effect size.

As in RQ1 and RQ2, these experiments are executed for 30 iterations and given a 14 hour fixed execution budget. We increased the search budget compared to previous empirical studies so as to accommodate the higher execution time [Mut+03] needed for population based evolutionary algorithms such as SPEA2. The search budget for RQ3 is determined based on the results from parameter tuning exercise (See Table 6.1) done for the SPEA2 algorithm. In the case of SPEA2 the population size, the archive population size, the mating pool size, the choice of candidate selection methodology all affected the increase in execution time. In order to evaluate the algorithm variants, we measured the maximum data stall cycles and energy

measurements generated from these experiments.

For the evaluation of multi-objective search-based algorithms (as in RQ3), a 64-bit system called Juno is used as detailed in Section 2.3.2. We use performance monitor counters to measure the data stall cycles and energy measurement from the on-board energy meters as a means to measure the stress on the CCI component.

The test input representation as outlined in Section 3.3 has different configurations in the experimental setup based on the target platform. The details of payload attributes as customised for our Juno setup can be seen in Page 53 for Payload Size and Sparsity and in Page 54 for Actor Profile.

In order to finalise on the search algorithm parameters for SPEA2, we adopted an exhaustive grid search approach [BB12] with the following tunables in consideration as in Table 6.1:

Table 6.1: SPEA2 parameter tuning attributes

population attributes			crossover rate			mutation rate		
p_sz	*a_sz*	*mp_sz*						
26	8	10						
58	15	20	0.5	0.7	0.9	0.05	0.1	0.25
80	20	30						

Here, the *p_sz* parameter is the population size from which the generational evolution happens. The *a_sz* parameter is the archive population size where the archive holds the selection of elitist candidates which are used in the generational evolution process. Finally, the *mp_sz* parameter indicate the mating pool size which controls the number of mating candidates selected for the generational evolution. Both crossover rate and mutation rate indicate the probability by which the crossover and the mutation operations happen during the generational evolution process. The search grid used is detailed in Appendix B.

The aim of SPEA2 based implementation is to generate a group of test cases which are in the pareto optimal front that can maximising both the data stall cycles and the energy measurement indicating maximum stress on the CCI component. From the parameter tuning experiment we observed that each of the parameters - population attributes, crossover rate and mutation rate - had an effect when varied during the parameter tuning experiment. The following parameters led to the best results and hence chosen for the experiment: *p_sz* of 26, *a_sz* of 8, *mp_sz* of 10, crossover rate of 0.5 and mutation rate of 0.25. These are set as part of the values used in the auxiliary functions relied on by Algorithm 4.

In case of SA.Composite we defined a composite fitness function that relied on both the data stall cycles and energy measurement scores. However these scores are on different scales. Thus we converted the raw scores to a normalised scale of 1 to

100. A larger scale was chosen to ensure that the sensitivity of the measurement does not get lost during the normalisation process. The normalised value of the individual scores are computed as:

$$normalised\_score = 1 + \frac{(raw\_score - lower\_scale) * (100 - 1)}{upper\_scale - lower\_scale}$$

where raw_score is the actual score for the individual objective with $lower\_scale = 0$ and

$$upper\_scale = average\_score + 3 * standard\_deviation$$

The composite fitness function is computed as a simple summation of the $normalised\_score$ for both the objectives with an equal weightage given. We consider both the measures exhibiting stress conditions in a similar manner hence decided to give equal weightage.

6.3 Results

We first consider the the plots comparing the data stall cycles (see Figure 6.3) and energy measurement (see Figure 6.4) from the experiments. Higher stress is manifested as larger data stall cycles and more energy spent. Thus an algorithm which is able to maximise both these factors is seen as a better one to generate stress on CCI component. SPEA2 or SA.Composite (multi-objective options) will be considered worthwhile if either of them can provide better or on-par results when compared with single objective algorithm baselines (SA.PMU and SA.ENERGY). This is despite the natural expectation on single objective algorithms to have an innate advantage in maximising its single objective.

From Figure 6.3 we see Simulated Annealing with the composite fitness function (SA.Composite) as the best performing one with a media value of 1461 followed by SA.PMU (media value 1377), SPEA2 (media value 1327) and SA.Energy (median value 1170).

Analysing the median values we note that SA.PMU is only 3.76% better than SPEA2 and we can treat SPEA2 to have comparable ability to generate stress based on data stall cycles. Also note that SPEA2 has a higher median score than SA.ENERGY and when it comes to upper quartile SPEA2 has a better score than both the single objective variants. But more interestingly, SA.Composite (against the expectations) comes on top of the SA.PMU in case of data stall cycles (when considering both median values and upper quartile values).

From Figure 6.4 we see Simulated Annealing with energy measurement as fitness function (SA.ENERGY) as the best performing one with a median value of 729

followed by SA.Composite (media value 728) and both SPEA2 and SA.PMU tied with the same median value of 727.

Analysing the median values we note that SA.ENERGY is only 0.14% and 0.28% better than SA.Composite and SPEA2 respectively. Thus we can treat SA.Composite and SPEA2 to have comparable ability to generate stress based on energy measure. When we look at the upper quartile, both SA.Composite and SPEA2 have a better score than both the single objective variants (SA.PMU and SA.ENERGY).

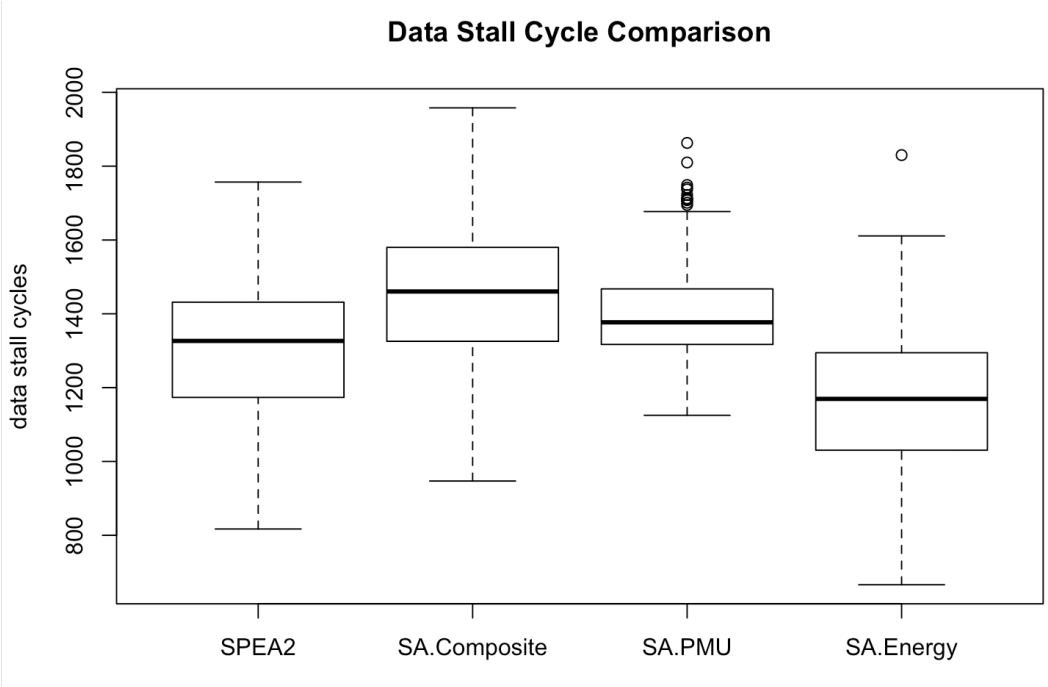

Figure 6.3: Data Stall Cycles Comparison *SPEA2 vs Simulated Annealing - Composite FF vs Simulated Annealing - PMU FF vs Simulated Annealing - ENERGY FF*

On the contrary of our expectations SA.Composite outperformed even the single objective algorithm variant (SA.PMU) when data stall cycles measure is considered and is at a comparable level to single objective algorithm (SA.Energy) when energy measurement is considered. We also note that the relative improvement of single objective algorithm variants over SPEA2 is observed to be at a minor scale (less that 4%) in each case.

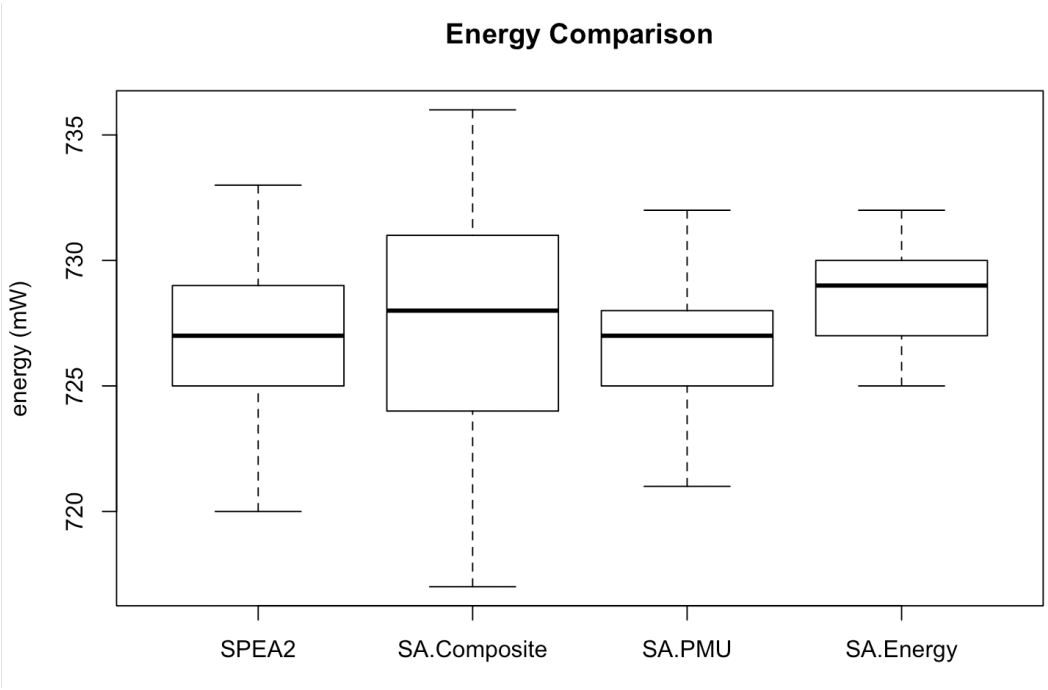

Figure 6.4: Energy Comparison *SPEA2 vs Simulated Annealing - Composite FF vs Simulated Annealing - PMU FF vs Simulated Annealing - ENERGY FF*

In order to consider either SPEA2 or SA.Composite (the multi-objective options) as worthwhile, we expect them to outperform SA.PMU in terms of maximising energy measurement and SA.ENERGY in terms of maximising data stall cycles. At the same time we expect a comparable measure on the objective which the single objective algorithm can innately maximise. We note that indeed SPEA2 outperform SA.ENERGY in terms of generating higher data stall cycles and holds on to equal capability by generating the same energy measurement when compared with SA.PMU. However SA.Composite is even better than SPEA2, in that it outperformed SA.PMU on the data stall cycles objective and obtained a comparable result with SA.Energy on the energy measurement fitness function. This result shows that the multi-objective algorithms (SPEA2 and SA.Composite) are indeed a worth-while option in maximising the generation of both data stall cycles and energy measurement compared to the single objective algorithm variants (SA.PMU and SA.ENERGY).

As the energy is measured on a milli-watt (mW) scale, we expect poorer statistical significance for the comparisons using energy measurement compared to those using data stall cycles. For the same reason, the effect size computed - to gauge the difference in the measurement - is expected to be smaller for comparisons using energy while medium to large effects are expected for comparisons using data stall cycles. We did an exploratory assessment for the above made assumptions based on the statistical significance test and effect size computations, which are discussed

Table 6.2: Wilcoxon signed-rank test (p) & Effect Size (es) - Data Stall Cycles & Energy Comparisons

	data stall cycles		energy	
	p-value	effect-size	p-value	effect-size
SPEA2 vs SA.PMU	6.026e-08	-0.6	0.3686	0.05
SPEA2 vs SA.ENERGY	4.802e-12	0.72	3.524e-14	-0.77
SA.PMU vs SA.ENERGY	<2.2e-16	1.36	<2.2e-16	-0.92
SA.Composite vs SPEA2	4.113e-15	0.77	0.05	0.16
SA.Composite vs SA.PMU	0.0002	0.29	0.017	0.2
SA.Composite vs SA.ENERGY	<2.2e-16	1.43	0.0028	-0.38

below.

Table 6.2 shows the results of evaluating the statistical significance and the effect size of the experiments conducted as part of RQ3. While the experiments using data stall cycles as fitness function show statistical significance and noticeable effect size [Coe02], for those experiments with energy measurement as the fitness function, the results observed are mixed - some shows statistical significance while others do not.

Figure 6.5 shows the results of the comparing the algorithms in terms of efficiency computed in terms of AUC (See Page 85). It shows that SA.Composite and SPEA2 (the 2 multi-objective variants) fares better than the rest with a median value of 322174 and 296333 respectively. This is when compared against the single objective baselines of SA.ENERGY with 269907 and SA.PMU with 133283.

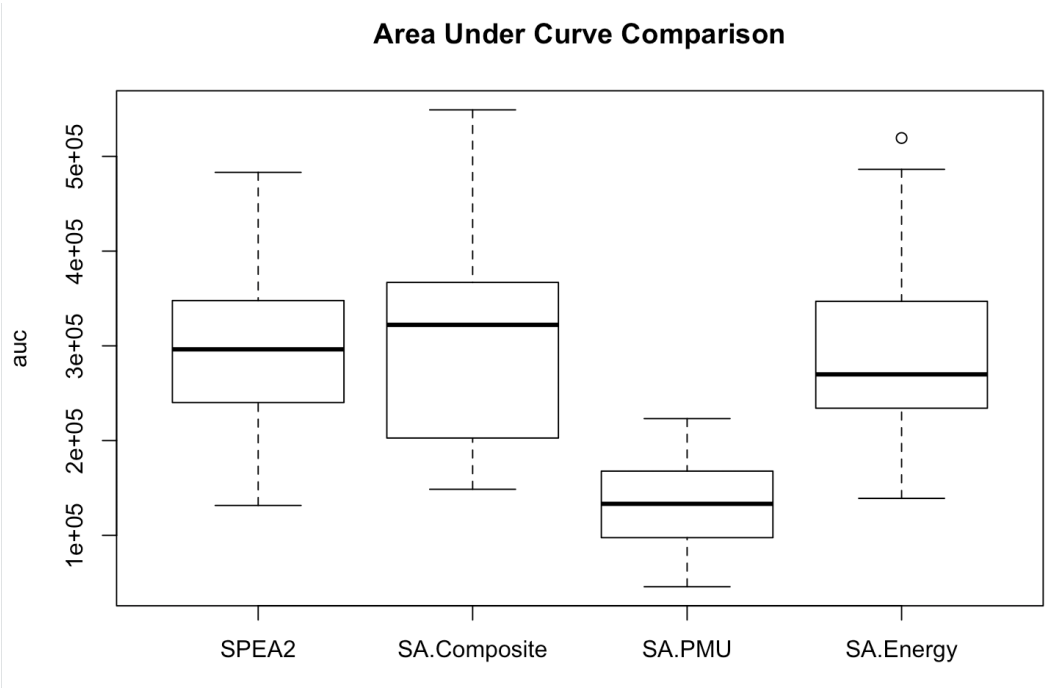

Figure 6.5: Area Under Curve Comparison *SPEA2 vs Simulated Annealing - Composite FF vs Simulated Annealing - PMU FF vs Simulated Annealing - ENERGY FF*

Table 6.3: Wilcoxon signed-rank test (p) & Effect Size (es) - AUC

	p-value	effect-size
SPEA2 vs SA.PMU	1.266e-10	2.09
SPEA2 vs SA.ENERGY	0.959	-0.08
SA.PMU vs SA.ENERGY	2.04e-12	-2.05
SA.Composite vs SPEA2	0.59	0.15
SA.Composite vs SA.PMU	7.957e-11	2.01
SA.Composite vs SA.ENERGY	0.67	0.07

Table 6.3 shows the results of evaluating the statistical significance and the effect size of the experiments conducted as part of RQ3. The AUC computation shows mixed results for statistical significance and computed effect size. They show

statistical significance with a 'medium' to 'large' effect for comparisons involving data stall cycles. For the comparisons involving energy measurement some show no statistical significance and no noticeable effect, while others on the contrary.

Conclusion: Considering both data stall cycles and energy measurement, we observe that the multi-objective variants (SPEA2 or SA.Composite) are able to either improve or be on-par for the two fitness functions evaluated against the single objective baselines. Since stress on CCI can be affected by multiple factors and manifest itself in more than one way, the multi-objective algorithm's ability to either improve or be on-par for both the fitness functions is encouraging. This makes the case for using multi-objective search-based algorithms in addressing this particular testing challenge.

6.4 Threats to Validity

Threats to Construct validity : In RQ3, we use data from PMU counters (data stall cycles) and Hardware Monitor (energy measurement) as fitness functions in the case of multi-objective and single objective algorithm variants. We expect these to indicate the measure of 'stress' on the CCI component. Energy measurements are susceptible to ambient temperature in the operational environment. Without having a temperature controlled environment, the use of energy measurement could pose a risk of threat to the validity of a suitable fitness function being identified to measure stress. In our study the experiments are conducted in a single location. While this does not provide the same mitigation as a temperature controlled environment, it reduces the effect of this threat to validity.

Threats to Internal validity : RQ3 shares the same threats to validity as outlined for RQ1 in Section 4.4.

Threats to External validity : RQ3 shares the same threats to validity as outlined for RQ1 in Section 4.4.

6.5 Summary

In this chapter we detailed the empirical study with a 64-bit Arm platform to assess if multi-objective search-based algorithms can provide a more practical solution in the testing of deeply embedded IC components in a system context. From the experiments we note that, multi-objective search algorithm variants (SPEA2 or SA.Composite) are able to improve the 2 objectives collectively better than the single objective algorithm. Both SA.Composite and SPEA2 provided a similar levels of uplift for both the data stall cycles and higher energy measurements as opposed

to Simulate Annealing algorithm using single objective (SA.PMU or SA.ENERGY) where they are able to provide the uplift only to the single objective it is maximising.

As the stress conditions on CCI when exercised from system level can manifest in multiple ways the application of multi-objective search-based approach provides a flexible means to address the testing challenges involved.

Chapter 7

Conclusion and Future Work

This thesis has investigated the suitability of search-based testing methodology in addressing testing challenges posed by 'deeply embedded' IC components in a full system context.

We analyse the results obtained from this research on how suitable search-based approaches are in solving the specific IC testing challenge in Section 7.1. We also discuss in Section 7.2 the relevance of this research in addressing the evolving IC testing challenges and captures some potential future directions for the research in this area at Section 7.3.

7.1 CCI Testing Improvements with SBST

The testing challenges posed by modern ICs with deeply embedded components is typical of the recent trend in electronic system designs - an increase in functional capability but with ever reducing testability. This situation demands us to look at the strengths of search-based software testing methodology, which can be beneficial over traditional testing approaches. In this research we focused on the application of search-based algorithms in addressing testing challenges pertaining to IC testing.

While search-based approaches present a wide range of options, the success of applying search-based algorithms are dependent on the identification of a suitable search-space representation, a good fitness function and a resultant fitness landscape that is suitable for the search algorithm. This fact is very much evident from the early part of this research, where we focused on identifying the suitable search-space model for this IC testing problem. This research exploits some existing hardware capabilities in the form of either PMU counters or energy measurement blocks to be part of the fitness function definition. Unlike in most scenarios where search-based approaches are applied, here finalising the fitness function for CCI testing is more straight-forward task - we rely on hardware counters or energy measurement units to indicate the stress on the CCI component. Even though we have an easy option

for defining the fitness function, the search-space modeling exercise proved to be tricky when a simple search grid from which memory operations are performed did not yield positive results. An eventual search-space model based on key attributes that affects the stress on CCI component in terms of Payload Size, Actor Count and Sparsity proved to be a more suitable search-space model for this testing problem.

In RQ1 we looked at the basic question of whether search-based algorithms (with a suitable search-space model finalised) can generate better tests that can stress the CCI component from a full system context. Even with the simplest of search-based algorithms - hill-climbing - we observed that search algorithms are able to generate tests that stress the CCI component better than what random testing could achieve. This paved way for the possibility of moving away from current scenario of having to rely on specialised hand-crafted tests for CCI testing. Establishing the capability of search-based algorithms to automatically generate test payloads - that can stress the CCI component - leads to better testing capabilities at the system testing phase. This capability not only leads the way for better testing in case of CCI, but in general establishes a way for better testing of similar core components which share the same testability challenges of deeply embedded components. The suitability of using search-based algorithms is underpinned by the positive results that showed considerable improvement over random tests when evaluating RQ1. This showcases its wider potential in similar test scenarios.

Even though we established the suitability of search-based algorithms with a simple local search algorithm as hill-climbing, in a real testing context the test demands go beyond what single objective search algorithms can address. As there are multiple aspects that influence the testing from system level and when one assess the stress on the platform that can be manifested in multiple ways, we see multi-objective search-algorithms as the better option for addressing this testing problem.

As part of RQ1 we did observe the local maxima issue, which is relevant to local search algorithms like hill-climbing. In the subsequent evaluation of RQ2 we see that more advanced search-based methods like random-restart hill-climbing and simulated annealing are able to address the local maxima issue - even though the scale of improvements over hill-climbing is marginal. Beyond addressing limitations of single objective search algorithms in this testing problem, our focus is to see if multi-objective search algorithms can satisfy the testing demands that are close to reality. With this view we conducted the empirical study that evaluates RQ3 to establish the benefits of using multi-objective search-based algorithms as opposed to single objective approaches in the context of stress testing deeply embedded IC components. Nejati *et al.* [NB14] shows the use of multi-objective algorithms to optimise non-functional system properties which emboldens us to pursue this direction.

This empirical study do confirm that multi-objective search algorithms can address the real-world IC testing demands at system level. The results from RQ3 give more credence to the suitability of this approach in addressing the practical testing demands posed by CCI testing from a full system context. With the complexity of deeply embedded components evolving over time, the empirical studies conducted as part of this thesis provides the confidence that search-based algorithms can be successfully applied in solving the testing challenges relevant to IC testing. This research also shows that by focusing on search-based algorithms, it can open up a large number of avenues available from the SBST research area, which could be suitable for testing problems relevant in IC testing and beyond.

7.2 Implications for Future IC Testing

The testing challenges posed by modern ICs is evolving over time. Looking at CCI family of products, we can see that they are evolving in complexity and usage context. The need for supporting large data mining application workloads and data accelerators have led to the evolution of chip designs to support these use-cases with the latest generation of cache coherency management hardware. There are system designs which now have not just Level1 or Level2 cache, but also a Level3 cached described as the system cache.

Newer applications and workloads have demanded systems to be designed for high data throughput and ability to run data analytics on-the-go, while data is getting shared across various systems. To support such advanced use-cases the Cache Coherent Interconnect for Accelerators (CCIX) Consortium [CCI] defines a new specification that the CCIX products have to support. Arm CoreLink^{TM}CMN-600 Coherent Mesh Network [Arme] is a CCIX product offering, with agile system cache support and multi-chip connectivity. This brings the testing challenges related to cache management at a system level to even higher level. We now have multiple systems (no longer intra-system, but inter-system data sharing) sharing data in the cache and the issues around testability are more profound than ever. We believe that the search-based approaches assessed as part of this thesis will have continued relevance in the future testing of complex IC components.

In this research we focused on the testing challenges posed by a hardware component called CCI which is inherently a 'deeply embedded' component. Acknowledging the critical nature of this component's behaviour in the overall system performance, a case is made for doing improved testing of this deeply embedded component from a system perspective. The aim is to go beyond the current state of play where specialised hand-crafted tests are the only means to test this. Even though the testability issues highlighted in Section 2.4 are in the context of deeply embedded

components like CCI, these are typical of other components too, especially when it comes to observability and information loss from sub-components in a full system context. Here CCI being a hardware component (without direct interfaces to higher level software), it gives an impression of being relevant to only hardware, however this can well be applied on any software component which is part of a larger system. In a complex system, we can see sub-components similar to that of CCI irrespective of whether it is hardware or software component exhibiting similar testability challenges. Thus the testability issue addressed in this research does not limit its relevance to CCI in particular or 'deeply embedded' hardware components in general. It has a wider relevance especially in a system testing context.

In this research we exploited some of the hardware functional blocks - that are already present - as part of the solution. Looking at hardware engineering as an engineering discipline, it has a very mature approach towards design for testability. This results in additional logic blocks to be incorporated in hardware designs that supports testing, be it self-test capabilities or additional logic to probe internal operational states etc. Often these test capabilities are exercised at individual component level in early phases of testing, but are seldom exploited for testing from system level. Sometimes they end up being only being used for debugging and never exploited for testing in the full system context. This research makes the case for looking at similar existing functional blocks that could help improve current testing efforts at system level. Potentially hardware designers can do more to support system level testing with similar functional blocks in future. Software engineering disciple should emulate the good work done by hardware engineering in the area of 'design for testability' thus improving system level testing capabilities in future.

7.3 Future Research Direction

In this research we started by focusing on using a single objective fitness function targeting a single PMU event which is followed by the use of multi-objective approach with PMU event and energy measurement as part of the fitness function. The successful generation of stress tests with these approaches as demonstrated on both 32-bit TC2 and 64-bit Juno platforms indeed confirm the suitability of this approach to similar testing challenges with deeply embedded hardware components. The proposed fitness function and search-space representation is evaluated to be suitable for this testing problem. We are able to establish that the search-based approach is indeed a suitable choice for addressing testing challenges posed by situations similar to that of stress testing deeply embedded IC components. Moreover multi-objective algorithms are better suited than single objective variants in this case.

Multi-objective search algorithms are most useful when there are conflicting objectives to be maximised. In the current study we focused on data stall cycles and energy measurement as the two objectives and they are orthogonal to each other, but none-the-less both measures indicate the underlying stress on the platform. This study can be extended to look at evaluating how some conflicting objectives would be used as part of the fitness function while using SPEA2. There are many additional PMU events which have the potential for use in the fitness function. Hence exploring additional PMU events for defining a more effective fitness function is a natural extension to this study. Do note that CCI-400 can record 4 PMU events in parallel without incurring any performance penalty. Hence exploring multiple PMU events is a natural extension to this research. Further extensions to this study are possible by focusing on evaluating the suitability of other prominent multi-objective algorithms like PESA [CKO00] and NSGA2 [Deb+00]. In case of SA.Composite we provided equal weightage to both the fitness functions. Future research could investigate the effect of varying these weightages in improving the faster convergence to optimal solution.

An innate problem with the application of more complex population based search-based approaches is the cost of experimentation in terms of the total execution time. With deeply embedded components tested at a system level we have a need for test payload executions to be repeated due to the large variances in the measurements. Hence exploring ways to reduce the total execution time for the experiments is definitely a future direction this research can look into. This also includes exploring further into SA.Composite variant (which is not a population based approach) to see if it can coverage at optimal solutions with lesser search budget.

The high data throughput use-cases addressed by CCIX specification is definitely an opportunity to leverage the search-based software testing methodologies in addressing the testing challenges there. It will be very interesting to extend this study to look at how effectively we can stress test this new hardware component from CCI family (CMN600) that deals with high speed and high throughput data from a full system context - especially when the cache data is shared between multiple systems. Definitely the evolving technology landscape do provide more opportunities to further explore the direction of this research.

Appendix A

Early Search-Space Model Trials

A.1 Genetic Algorithm based Experiments

A.1.1 Algorithm

The following Algorithm 5 outlines the implementation of the genetic algorithm in arriving at the best candidate to generate the maximum data stall cycles.

```
Input:
P_p, P_c - Parent and Child Population
T_p - Tournament Pool
E_g - Elite group
P_1, P_2 - Parent individuals for Crossover
Output: The optimal candidate maximising the fitness score
begin
    P_p = P_c = empty
    P_p ← seedPopulation()
    while !exhausted do
        /* Generating the child population                    */
        for i= 1 till populationsize do
            /* Select first parent                            */
            for j= 1 till tournamentsize do
                T_p ← randomIndividual(P_p)
            end

            P_1 ← tournamentSelecton(T_p)
            /* Select second parent                           */
            for j= 1 till tournamentsize do
                T_p ← randomIndividual(P_p)
            end

            P_2 ← tournamentSelecton(T_p)
            P_c ← crossover(P_1, P_2, singlepointcrossover)
        end
        /* Perform mutation operation                         */
        P_c ← mutatePopulation(P_c, mutationrate)
        /* Select the elite children                          */
        E_g ← groupElite(P_c)
        /* Produce next generation based on elitist replacement
           strategy                                           */
        P_p' ← generationalReplacement(P_c, E_g, elitistreplacement)
        if populationScore(P_p') > populationScore(P_p) then
          P_p ← P_p'

        else exhausted ← true

    end
    return bestIndividual(P_p)
end
```

Algorithm 5: Dynamic Test Data Generation with Genetic Algorithm

A.1.2 Results

A series of experimental runs are done with the current implementation of the evolutionary test framework. The aim is to identify the test vectors that would increase

the data stall cycle events on CCI in case of TC2 platform. This should indicate whether the data stall rates on the given platform is within acceptable limits or not. For the current experiment the population size is set as 30. 'Tournament selection' is used as method for candidate selection that is to be then used for cross over operations and the tournament size is set to 5. A single-point crossover method is adopted. The crossover rate is set to 100% and mutation rate is experimented with values varying between 0.5% to 0.001%. An Elitist generational replacement strategy is adopted with an elitist group size set to 5.

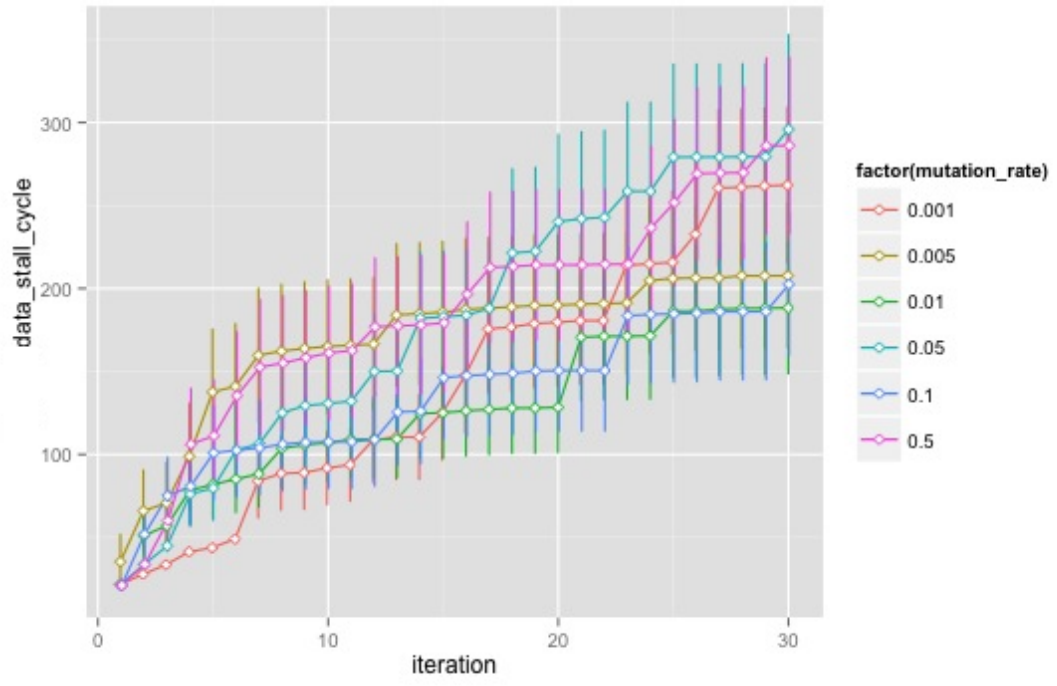

Figure A.1: Effect of Mutation rate.

The above plot outlines the effect of mutation rate on the genetic algorithm. The mutation rate is varied across 0.001, 0.005, 0.01, 0.05, 0.1 and 0.5. It shows the standard error of the mean plot for the elite group fitness score over generations. The plot indicates that certain mutation rates favour the evolutionary algorithm execution.

Based on the above experiment, a favourable mutation rate of 0.05% is chosen and the following shows the standard error of mean plot for 100 iterations with chromosome length of 1024 (1 KB of data):

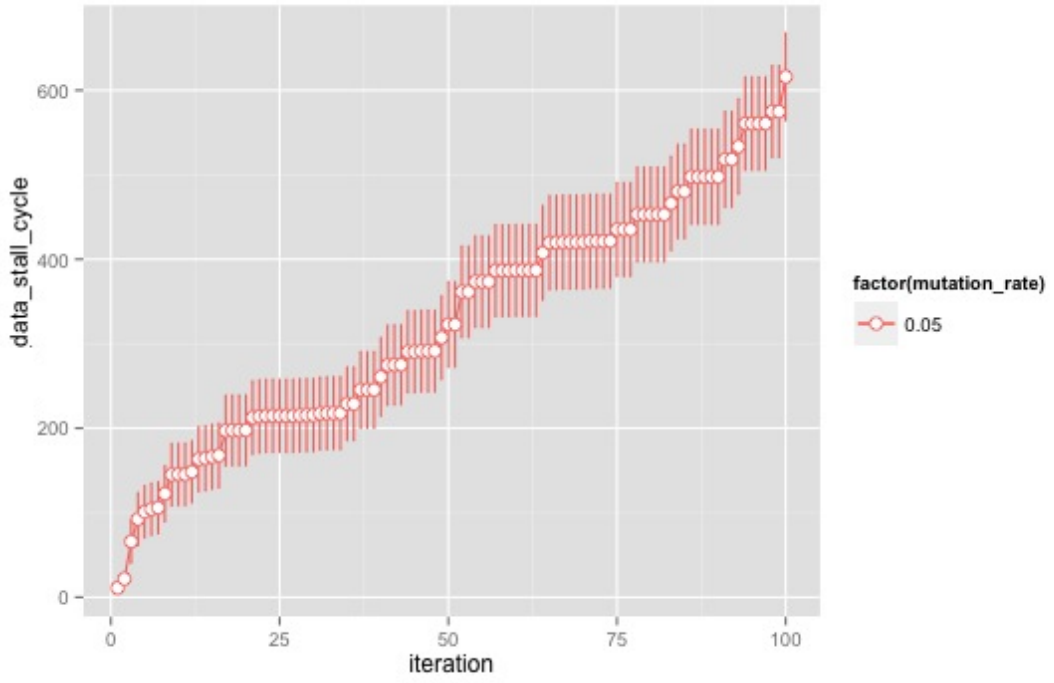

Figure A.2: GA with mutation rate set to 0.05.

Appendix B

Algorithm Parameter Evaluation

B.1 Grid Approach

The following outlines the grid used for evaluating the SPEA2 algorithm parameters. Experiments were run against each grid configuration and the results compared to arrive at the parameter selection to be used for the experiments.

Table B.1: SPEA2 Parameter Evaluation - Grid Approach

Config	**population attributes (p_sz,a_sz,mp_sz)**			**crossover rate**			**mutation rate**		
	28,8,10	**58,15,20**	**80,20,30**	**0.5**	**0.7**	**0.9**	**0.05**	**0.1**	**0.25**
1	x			x			x		
2	x				x		x		
3	x					x	x		
4	x			x				x	
5	x				x			x	
6	x					x		x	
7	x			x					x
8	x				x				x
8	x					x			x
10		x		x			x		
11		x			x		x		
12		x				x	x		
13		x		x				x	
14		x			x			x	
15		x				x		x	
16		x		x					x
17		x			x				x
18		x				x			x
19			x	x			x		
20			x		x		x		
21			x			x	x		
22			x	x				x	
23			x		x			x	
24			x			x		x	
25			x	x					x
26			x		x				x
27			x			x			x
7_A	58,8,10 x			x					x

A special configuration Config7_A was also defined to see the effect of increasing population size while the rest being constant on one of the favorable configs.

Appendix C

Implementation Project

C.1 Project Details

The following provides some additional details about the implementation of the test framework [Elj] that is used for the experimentation.

Table C.1: Project - LoC Details

Language	files	blank	comment	code
C	12	1582	1629	5484
XML	6	0	0	1166
Python	10	177	321	546
C/C++ Header	21	169	435	531
make	5	35	4	92
Sum:	54	1963	2389	7819

Bibliography

[Ede97] Alan Edelman. "The Mathematics of the Pentium Division Bug". In: *SIAM Review* 39 (1997), pp. 54–67.

[Wik] Wikipedia. *Intel Pentium F00F Bug*. URL: https://en.wikipedia.org/wiki/Pentium_F00F_bug.

[Koc+19] Paul Kocher et al. "Spectre Attacks: Exploiting Speculative Execution". In: *40th IEEE Symposium on Security and Privacy (S&P'19)*. 2019.

[Lip+18] Moritz Lipp et al. "Meltdown: Reading Kernel Memory from User Space". In: *27th USENIX Security Symposium (USENIX Security 18)*. 2018.

[Ber07] A. Bertolino. "Software Testing Research: Achievements, Challenges, Dreams". In: *Future of Software Engineering (FOSE '07)*. May 2007, pp. 85–103. DOI: 10.1109/FOSE.2007.25.

[SWH11] M. Staats, M. W. Whalen, and M. P. E. Heimdahl. "Programs, tests, and oracles: the foundations of testing revisited". In: *2011 33rd International Conference on Software Engineering (ICSE)*. May 2011, pp. 391–400. DOI: 10.1145/1985793.1985847.

[ISOa] ISO. *ISO/IEC 25010:2011 - Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models*. URL: https://www.iso.org/standard/35733.html.

[Bar+15] Earl T. Barr et al. "The Oracle Problem in Software Testing: A Survey". In: *IEEE Transactions on Software Engineering* 41 (2015), pp. 507–525.

[Rot+99] G. Rothermel et al. "Test case prioritization: an empirical study". In: *Proceedings IEEE International Conference on Software Maintenance - 1999 (ICSM'99). 'Software Maintenance for Business Change' (Cat. No.99CB36360)*. Aug. 1999, pp. 179–188. DOI: 10.1109/ICSM.1999.792604.

[Di +15] Daniel Di Nardo et al. “Coverage-based regression test case selection, minimization and prioritization: a case study on an industrial system”. In: *Software Testing, Verification and Reliability* 25.4 (2015), pp. 371–396. DOI: 10.1002/stvr.1572. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/stvr.1572. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/stvr.1572.

[YH10] Shin Yoo and Mark Harman. “Regression testing minimisation, selection and prioritisation: A survey”. In: *Software Testing, Verification, and Reliability* (2010).

[Hut+94] M. Hutchins et al. “Experiments on the effectiveness of dataflow- and control-flow-based test adequacy criteria”. In: *Proceedings of 16th International Conference on Software Engineering*. May 1994, pp. 191–200. DOI: 10.1109/ICSE.1994.296778.

[HGW04] M. P. E. Heimdahl, D. George, and R. Weber. “Specification test coverage adequacy criteria = specification test generation inadequacy criteria”. In: *Eighth IEEE International Symposium on High Assurance Systems Engineering, 2004. Proceedings.* Mar. 2004, pp. 178–186. DOI: 10.1109/HASE.2004.1281742.

[Raj06] A. Rajan. “Coverage Metrics to Measure Adequacy of Black-Box Test Suites”. In: *21st IEEE/ACM International Conference on Automated Software Engineering (ASE'06)*. Sept. 2006, pp. 335–338. DOI: 10.1109/ASE.2006.31.

[ISOb] ISO. *ISO/IEC 26262-1:2018 - Road vehicles — Functional safety — Part 1: Vocabulary*. URL: https://www.iso.org/standard/68383.html.

[Tei12] Jürgen Teich. “Hardware/Software Codesign: The Past, the Present, and Predicting the Future”. In: *Proceedings of the IEEE* 100.Special Centennial Issue (May 2012), pp. 1411–1430. DOI: 10.1109/jproc.2011.2182009. URL: https://doi.org/10.1109/JPROC.2011.2182009.

[Rob15] Tiziano Villa Robert Brayton Luca P. Carloni. “Design Automation of Electronic Systems: Past Accomplishments and Challenges Ahead”. In: *Proceedings of the IEEE* 103 (11 Oct. 2015). DOI: 10.1109/JPROC.2015.2487798. URL: https://doi.org/10.1109/JPROC.2015.2487798.

[Hua+11] Chung-Yang Huang et al. “SoC HW/SW verification and validation”. In: *16th Asia and South Pacific Design Automation Conference (ASP-DAC 2011)* (2011), pp. 297–300.

[CNR13] Alessandro Cimatti, Iman Narasamdya, and Marco Roveri. "Software Model Checking SystemC". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 32 (2013), pp. 774–787.

[JHS99] Jih-Kwon Peir, W. W. Hsu, and A. J. Smith. "Functional implementation techniques for CPU cache memories". In: *IEEE Transactions on Computers* 48.2 (Feb. 1999), pp. 100–110. ISSN: 0018-9340. DOI: 10.1109/12.752651.

[AA17] Z. Al-Waisi and M. O. Agyeman. "An overview of on-chip cache coherence protocols". In: *2017 Intelligent Systems Conference (IntelliSys)*. Sept. 2017, pp. 304–309. DOI: 10.1109/IntelliSys.2017.8324309.

[BW88] J. -. Baer and W. -. Wang. "On the inclusion properties for multi-level cache hierarchies". In: *[1988] The 15th Annual International Symposium on Computer Architecture. Conference Proceedings*. May 1988, pp. 73–80. DOI: 10.1109/ISCA.1988.5212.

[Arma] Arm. *Arm CoreLink CCI-400 Cache Coherent Interconnect - Technical Reference Manual*. URL: http://infocenter.arm.com/help/topic/com.arm.doc.ddi0470k/DDI0470K_cci400_r1p5_trm.pdf.

[Armb] Arm. *TestChip2 - part of Arm Versatile Express product family*. URL: http://www.arm.com/products/tools/development-boards/versatile-express/index.php.

[Armc] Arm. *Juno ADP - part of Arm Versatile Express product family*. URL: https://www.arm.com/files/pdf/DDI0515D1a_juno_arm_development_platform_soc_trm.pdf.

[Lin] Linaro. *Linaro - A non-profit organisation working on open source software for Arm based platforms*. URL: http://www.linaro.org.

[Mar06] Alessandro Cimatti Marco Bernardo. *Formal Methods for Hardware Verification*. Springer, Berlin, Heidelberg, 2006. DOI: https://doi.org/10.1007/11757283.

[Meh17] Ashok B. Mehta. *Constrained Random Verification*. Springer, Cham, 2017. DOI: https://doi.org/10.1007/11757283.

[The+13] G. Theodorou et al. "Software-Based Self Test Methodology for On-Line Testing of L1 Caches in Multithreaded Multicore Architectures". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 21.4 (Apr. 2013), pp. 786–790. ISSN: 1063-8210. DOI: 10.1109/TVLSI.2012.2191000.

[The+14] G. Theodorou et al. “Software-Based Self-Test for Small Caches in Microprocessors”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 33.12 (Dec. 2014), pp. 1991–2004. ISSN: 0278-0070. DOI: 10.1109/TCAD.2014.2363387.

[You+16] J. You et al. “Red Baron: Near/post-silicon SoC cache coherence stress tester”. In: *2016 IEEE Dallas Circuits and Systems Conference (DCAS)*. Oct. 2016, pp. 1–4. DOI: 10.1109/DCAS.2016.7791135.

[VM95] J. M. Voas and K. W. Miller. “Software testability: the new verification”. In: *IEEE Software* 12.3 (May 1995), pp. 17–28. ISSN: 0740-7459. DOI: 10.1109/52.382180.

[She+15] Du Shen et al. “Automating Performance Bottleneck Detection Using Search-based Application Profiling”. In: *Proceedings of the 2015 International Symposium on Software Testing and Analysis*. ISSTA 2015. Baltimore, MD, USA: ACM, 2015, pp. 270–281. ISBN: 978-1-4503-3620-8. DOI: 10.1145/2771783.2771816. URL: http://doi.acm.org/10.1145/2771783.2771816.

[ATF09] Wasif Afzal, Richard Torkar, and Robert Feldt. “A Systematic Review of Search-based Testing for Non-functional System Properties”. In: *Inf. Softw. Technol.* 51.6 (June 2009), pp. 957–976. ISSN: 0950-5849. DOI: 10.1016/j.infsof.2008.12.005. URL: http://dx.doi.org/10.1016/j.infsof.2008.12.005.

[BPS03] André Baresel, Hartmut Pohlheim, and Sadegh Sadeghipour. “Structural and Functional Sequence Test of Dynamic and State-based Software with Evolutionary Algorithms”. In: *Proceedings of the 2003 International Conference on Genetic and Evolutionary Computation: PartII*. GECCO’03. Chicago, IL, USA: Springer-Verlag, 2003, pp. 2428–2441. ISBN: 3-540-40603-4. URL: http://dl.acm.org/citation.cfm?id=1756582.1756738.

[Can+05] Gerardo Canfora et al. “An Approach for QoS-aware Service Composition Based on Genetic Algorithms”. In: *Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation*. GECCO ’05. Washington DC, USA: ACM, 2005, pp. 1069–1075. ISBN: 1-59593-010-8. DOI: 10.1145/1068009.1068189. URL: http://doi.acm.org/10.1145/1068009.1068189.

[Del+05] Concettina Del Grosso et al. “Improving Network Applications Security: A New Heuristic to Generate Stress Testing Data”. In: *Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation*.

GECCO '05. Washington DC, USA: ACM, 2005, pp. 1037–1043. ISBN: 1-59593-010-8. DOI: 10.1145/1068009.1068185. URL: http://doi.acm.org/10.1145/1068009.1068185.

[WK09] J. Wegener and P. M. Kruse. "Search-Based Testing with in-the-loop Systems". In: *2009 1st International Symposium on Search Based Software Engineering*. May 2009, pp. 81–84. DOI: 10.1109/SSBSE.2009.15.

[LW10] F. Lindlar and A. Windisch. "A Search-Based Approach to Functional Hardware-in-the-Loop Testing". In: *2nd International Symposium on Search Based Software Engineering*. Sept. 2010, pp. 111–119. DOI: 10.1109/SSBSE.2010.22.

[Mut+11] F. Mutter et al. "Model-Driven In-the-Loop Validation: Simulation-Based Testing of UAV Software Using Virtual Environments". In: *2011 18th IEEE International Conference and Workshops on Engineering of Computer-Based Systems*. Apr. 2011, pp. 269–275. DOI: 10.1109/ECBS.2011.30.

[CS16] Davi Ferreira de Castro and Davi AntÃ´nio dos Santos. "A Software-in-the-Loop Simulation Scheme for Position Formation Flight of Multicopters". en. In: *Journal of Aerospace Technology and Management* 8 (Dec. 2016), pp. 431–440. ISSN: 2175-9146. URL: http://www.scielo.br/scielo.php?script=sci_arttext&pid=S2175-91462016000400431&nrm=iso.

[BLS05] Lionel C. Briand, Yvan Labiche, and Marwa Shousha. "Stress Testing Real-time Systems with Genetic Algorithms". In: *Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation*. GECCO '05. Washington DC, USA: ACM, 2005, pp. 1021–1028. ISBN: 1-59593-010-8. DOI: 10.1145/1068009.1068183. URL: http://doi.acm.org/10.1145/1068009.1068183.

[Ale+15] Stefano Di Alesio et al. "Combining Genetic Algorithms and Constraint Programming to Support Stress Testing of Task Deadlines". In: *ACM Trans. Softw. Eng. Methodol.* 25.1 (Dec. 2015). ISSN: 1049-331X. DOI: 10.1145/2818640. URL: https://doi.org/10.1145/2818640.

[QM12] X. Qin and P. Mishra. "Automated generation of directed tests for transition coverage in cache coherence protocols". In: *2012 Design, Automation Test in Europe Conference Exhibition (DATE)*. Mar. 2012, pp. 3–8. DOI: 10.1109/DATE.2012.6176423.

[EN16] M. Elver and V. Nagarajan. “McVerSi: A test generation framework for fast memory consistency verification in simulation”. In: *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. Mar. 2016, pp. 618–630. DOI: 10.1109/HPCA.2016.7446099.

[Acl+15] J. P. Acle et al. “On the functional test of the cache coherency logic in multi-core systems”. In: *2015 IEEE 6th Latin American Symposium on Circuits Systems (LASCAS)*. Feb. 2015, pp. 1–4. DOI: 10.1109/LASCAS.2015.7250453.

[McM04] Phil McMinn. “Search-based Software Test Data Generation: A Survey: Research Articles”. In: *Softw. Test. Verif. Reliab.* 14.2 (June 2004), pp. 105–156. ISSN: 0960-0833. DOI: 10.1002/stvr.v14:2. URL: http://dx.doi.org/10.1002/stvr.v14:2.

[Har07] M. Harman. “The Current State and Future of Search Based Software Engineering”. In: *Future of Software Engineering (FOSE ’07)*. May 2007, pp. 342–357. DOI: 10.1109/FOSE.2007.29.

[JY04] Sheldon H. Jacobson and Enver Yücesan. “Analyzing the Performance of Generalized Hill Climbing Algorithms”. In: *Journal of Heuristics* 10.4 (July 2004), pp. 387–405. ISSN: 1381-1231. DOI: 10.1023/B:HEUR.0000034712.48917.a9. URL: https://doi.org/10.1023/B:HEUR.0000034712.48917.a9.

[YM93] Deniz Yuret and Michael de la Maza. “Dynamic Hill Climbing: Overcoming the limitations of optimization techniques”. In: *In The Second Turkish Symposium on Artificial Intelligence and Neural Networks*. 1993, pp. 208–212.

[NJ10] Alexander G. Nikolaev and Sheldon H. Jacobson. “Simulated Annealing”. In: *Handbook of Metaheuristics*. Ed. by Michel Gendreau and Jean-Yves Potvin. Boston, MA: Springer US, 2010, pp. 1–39. ISBN: 978-1-4419-1665-5. DOI: 10.1007/978-1-4419-1665-5_1. URL: https://doi.org/10.1007/978-1-4419-1665-5_1.

[PK98] Moon-Won Park and Yeong-Dae Kim. “A systematic procedure for setting parameters in simulated annealing algorithms”. In: *Computers & OR* 25 (1998), pp. 207–217.

[ZLT01] Eckart Zitzler, Marco Laumanns, and Lothar Thiele. *SPEA2: Improving the Strength Pareto Evolutionary Algorithm*. Tech. rep. Swiss Federal Institute of Technology (ETH), Zurich, 2001.

[ZT98] Eckart Zitzler and Lothar Thiele. *An Evolutionary Algorithm for Multiobjective Optimization: The Strength Pareto Approach*. Tech. rep. Swiss Federal Institute of Technology (ETH), Zurich, 1998.

[CKO00] David W. Corne, Joshua D. Knowles, and Martin J. Oates. "The Pareto Envelope-Based Selection Algorithm for Multiobjective Optimization". In: *Parallel Problem Solving from Nature PPSN VI*. Ed. by Marc Schoenauer et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 839–848. ISBN: 978-3-540-45356-7.

[Deb+00] Kalyanmoy Deb et al. "A Fast Elitist Non-Dominated Sorting Genetic Algorithm for Multi-Objective Optimization: NSGA-II". In: Springer, 2000, pp. 849–858.

[KCS06] Abdullah Konak, David Coit, and Alice E. Smith. "Multi-objective optimization using genetic algorithms: A tutorial". English (US). In: *Reliability Engineering and System Safety* 91.9 (Sept. 2006), pp. 992–1007. ISSN: 0951-8320. DOI: 10.1016/j.ress.2005.11.018.

[TM14] Mujahid Tabassum and Kuruvilla Mathew. "A Genetic Algorithm Analysis towards Optimization solutions". In: *International Journal of Digital Information and Wireless Communications (* 4 (Jan. 2014), pp. 124–142. DOI: 10.17781/P001091.

[Has+19] Ahmad Hassanat et al. "Choosing Mutation and Crossover Ratios for Genetic Algorithms-A Review with a New Dynamic Approach". In: (Dec. 2019), p. 390. DOI: 10.3390/info10120390.

[Nai15] Ravi Nair. "Evolution of Memory Architecture". In: *Proceedings of the IEEE* 103 (2015), pp. 1331–1345.

[Armd] Arm. *When to use Barrier instructions?* URL: http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.faqs/ka14041.html.

[EW16] Basil Eljuse and Neil Walkinshaw. "A Search Based Approach for Stress-Testing Integrated Circuits". In: *Search Based Software Engineering*. Ed. by Federica Sarro and Kalyanmoy Deb. Cham: Springer International Publishing, 2016, pp. 80–95. ISBN: 978-3-319-47106-8.

[EW18] Basil Eljuse and Neil Walkinshaw. "Comparison of Search-Based Algorithms for Stress-Testing Integrated Circuits". In: *Search-Based Software Engineering*. Ed. by Thelma Elita Colanzi and Phil McMinn. Cham: Springer International Publishing, 2018, pp. 198–212. ISBN: 978-3-319-99241-9.

[Elj] Basil Eljuse. *Project Source Code*. URL: https://doi.org/10.25392/leicester.data.11825652.v2.

[Goo] Google. *Android Debug Bridge:* URL: https://developer.android.com/studio/command-line/adb.

[FGG18] Mona Fronita, Rahmat Gernowo, and Vincencius Gunawan. "Comparison of Genetic Algorithm and Hill Climbing for Shortest Path Optimization Mapping". In: *E3S Web of Conferences* 31 (Jan. 2018), p. 11017. DOI: 10.1051/e3sconf/20183111017.

[RC95] Peter Ross and Dave Corne. "Comparing genetic algorithms, simulated annealing, and stochastic hillclimbing on timetabling problems". In: *Evolutionary Computing*. Ed. by Terence C. Fogarty. Berlin, Heidelberg: Springer Berlin Heidelberg, 1995, pp. 94–102. ISBN: 978-3-540-47515-6.

[Mac+10] A. MacFarlane et al. "An experimental comparison of a genetic algorithm and a hill-climber for term selection". In: *Journal of Documentation* 66.4 (2010), pp. 513–531. DOI: 10.1108/00220411011052939. URL: https://openaccess.city.ac.uk/id/eprint/1693/.

[Sak+14] Shinji Sakamoto et al. "A Comparison Study of Hill Climbing, Simulated Annealing and Genetic Algorithm for Node Placement Problem in WMNs". In: *J. High Speed Netw.* 20.1 (Jan. 2014), pp. 55–66. ISSN: 0926-6801.

[Sha+15] Sina Shamshiri et al. "Random or Genetic Algorithm Search for Object-Oriented Test Suite Generation?" In: *Genetic and Evolutionary Computation Conference (GECCO 2015)*. ACM, 2015, pp. 1367–1374.

[Jac02] Sheldon Jacobson. "Analyzing the Performance of Local Search Algorithms Using Generalized Hill Climbing Algorithms". In: (Jan. 2002). DOI: 10.1007/978-1-4615-1507-4_20.

[Coe02] Rohert J. Coe. "It's the Effect Size, Stupid: What effect size is and why it is important". In: *Proceedings of Annual Conference of the British Educational Research Association* (2002).

[BT96] T. Blickle and L. Thiele. "A Comparison of Selection Schemes Used in Evolutionary Algorithms". In: *Evolutionary Computation* 4.4 (Dec. 1996), pp. 361–394. ISSN: 1063-6560. DOI: 10.1162/evco.1996.4.4.361.

[IND06] H. Ishibuchi, Y. Nojima, and T. Doi. "Comparison between single-objective and multi-objective genetic algorithms: Performance comparison and performance measures". In: *2006 IEEE Congress on Evolutionary Computation, CEC 2006*. 2006, pp. 1143–1150. ISBN: 0780394879.

[Mut+03] A. Mutoh et al. *Reducing execution time on genetic algorithm in real-world applications using fitness prediction: parameter optimization of SRM control*. Dec. 2003. DOI: 10.1109/CEC.2003.1299624.

[BB12] James Bergstra and Yoshua Bengio. "Random Search for Hyper-parameter Optimization". In: *J. Mach. Learn. Res.* 13.1 (Feb. 2012), pp. 281–305. ISSN: 1532-4435. URL: http://dl.acm.org/citation.cfm?id=2503308.2188395.

[NB14] Shiva Nejati and Lionel C. Briand. "Identifying Optimal Trade-Offs between CPU Time Usage and Temporal Constraints Using Search". In: *Proceedings of the 2014 International Symposium on Software Testing and Analysis*. ISSTA 2014. San Jose, CA, USA: Association for Computing Machinery, 2014, pp. 351–361. ISBN: 9781450326452. DOI: 10.1145/2610384.2610396. URL: https://doi.org/10.1145/2610384.2610396.

[CCI] CCIX. *Cache Coherent Interconnect for Accelerators Consortium*. URL: https://www.ccixconsortium.com.

[Arme] Arm. *Arm CoreLink CMN-600 Coherent Mesh Network*. URL: https://developer.arm.com/ip-products/system-ip/corelink-interconnect/corelink-coherent-mesh-network-family/corelink-cmn-600.