BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INTELLIGENT SYSTEMS
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

APPLICATIONS OF FORMAL METHODS IN APPROXIMATE COMPUTING

VYUŽITÍ FORMÁLNÍCH METOD V PŘIBLIŽNÉM POČÍTÁNÍ

PHD THESIS
DISERTAČNÍ PRÁCE

AUTHOR **Ing. JIŘÍ MATYÁŠ**
AUTOR PRÁCE

SUPERVISOR **doc. RNDr. MILAN ČEŠKA, Ph.D.**
ŠKOLITEL

CO-SUPERVISOR **prof. Ing. TOMÁŠ VOJNAR, Ph.D.**
ŠKOLITEL SPECIALISTA

BRNO 2023

Abstract

As the Moore's Law ceases to hold, the ever increasing demand for high performance and lower power computer systems leads to the emergence of novel alternative computing paradigms. One of these paradigms is the so called *approximate computing* – a technique aiming to increase the efficiency of computations by introducing some errors into the computed results. This paradigm is mainly applicable in the error resilient applications – a class of applications where the absolute precision of the result is not critical, the most prominent of which include neural networks, multimedia and signal processing, or data mining. Naturally, techniques for approximate computing developed at various levels of the computing system architectures - the hardware, memory, operating system and software levels.

This thesis aims at the search-based design techniques for approximate arithmetic circuits. Circuit approximation is a crucial domain of approximate computing, as the approximate circuits can serve as the basic building blocks for larger systems and applications. We focus on the automated search-based approaches, which often work iteratively: in each iteration they 1) create the approximate candidates (synthesizer component), 2) evaluate their error with respect to the correct solution (analyser component). To successfully approximate complex circuits, the search based approaches usually need to perform a high number of iterations. Therefore, efficient synthesizer and analyser components are essential.

In order to improve the performance of the approximation process, we employ formal verification methods in both the synthesizer and analyser components of a circuit design loop implemented using Cartesian Genetic Programming. The evaluator component is accelerated with the utilisation of a novel construction of the specialised intermediate circuits called the approximation miters. The miters allow us to translate the error quantification procedure to a decision problem that can be evaluated using a SAT solver. We further enhance the performance of the search algorithm by integrating it with a verifiability driven strategy that guides the search towards promptly verifiable circuits and thus performs a significantly higher number of iterations, leading to a better quality of the final approximate solutions. We also improve the performance of the synthesizer component using an integration of satisfiability based local sub-circuit optimisation with the search algorithm. This effectively allows the search strategy to escape local optima and to further improve the quality of the solution. Finally, we propose a novel mutation operator tailored to circuit approximation that improves the overall performance of the approximation process.

The research presented in this thesis fundamentally improves the capabilities of the current search-based circuit approximation techniques and thus allows us to design approximate circuits with large bit-widths and complex structure (e.g. 32-bit multipliers or 128-bit adders) with the best known trade-offs between the power consumption and approximation error.

Keywords

Formal verification, approximate computing, approximate arithmetic circuits, Cartesian Genetic Programming, approximate equivalence checking.

Abstrakt

V minulosti se výkon počítačových systémů zvyšoval hlavně díky tzv. Mooreovu zákonu – každé dva roky se počet transistorů na čipu přibližně zdvojnásobí. V současné době tento zákon přestává platit a tak se objevují a vyvíjí nové alternativní výpočetní přístupy, které mají za úkol zrychlit a zefektivnit výpočetní systémy. Jedním z těchto přístupů je tzv. aproximované počítání, které se snaží urychlit a zefektivnit výpočty za cenu přijatelných nepřesností ve výsledcích. Tento přístup je aplikovatelný hlavně v oblastech, které jsou přirozeně odolné vůči chybám – např. neuronové sítě nebo zpracování multimédií. Techniky pro aproximované počítání se postupně vyvinuly na všech úrovních výpočetních systémů.

V rámci této práce se zaměřujeme na prohledávací algoritmy pro přibližný návrh hardwarových aritmetických obvodů. Aproximace aritmetických obvodů má velký potenciál, protože tyto obvody slouží jako základní stavební kameny větších systémů. Automatizované prohledávací aproximační algoritmy často pracují iterativně. V každé iteraci se nejprve vytvoří kandidátní aproximovaná řešení (pomocí komponenty zvané syntetizér), a poté se vyhodnotí jejich chyba vzhledem ke správnému řešení (komponenta analyzátor). Pro získání kvalitních aproximovaných obvodů musí prohledávací algoritmy vykonat velké množství těchto iterací. Proto je nutná vysoká efektivita syntetizéru i analyzátoru.

Abychom zvýšili výkonnost těchto komponent, zapojujeme do prohledávacího algoritmu založeném na Kartézském genetickém programování (CGP) metody formální verifikace. Analyzátor je akcelerován za použití speciálního obvodu zvaného aproximační miter, který nám umožňuje převést vyhodnocení chyby obvodu na rozhodovací problém a tento problém vyřešit pomocí nástrojů zvaných SAT solvery. Další zrychlení aproximačního algoritmu přináší nově navržená strategie, která uvaluje limit na prostředky, které může SAT solver využít při vyhodnocování chyby kandidátních řešení. Díky tomuto limitu je evoluční algoritmus motivován hledat rychle verifikovatelná řešení. Výsledkem je větší množství iterací prohledávacího algoritmu a tím pádem také vyšší kvalita výsledných aproximovaných obvodů. Použitý evoluční algoritmus se může během aproximace "zaseknout" v tzv. lokálních optimech. Navržené vylepšení syntetizéru integruje CGP a optimalizaci pod-obvodů využívající SAT solver umožňuje evolučnímu algoritmu uniknout z lokálních optim. Díky tomu může algoritmus dále zlepšovat řešení i v případech, v nichž by se původní varianta CGP již dále nezlepšila. Dalším navrženým vylepšením syntetizéru je nový mutační operátor pro CGP, vytvořený speciálně pro co nejefektivnější aproximaci obvodů.

Výsledky prezentované v rámci této dizertační práce výrazně vylepšují výkonnost prohledávacích algoritmů pro aproximaci aritmetických obvodů. Díky tomu můžeme získat aproximace obvodů velkých bitových šířek se složitou vnitřní strukturou (např. 32bitové násobičky nebo 128bitové sčítačky), které poskytují doposud nejlepší známý poměr mezi aproximační chybou a spotřebou elektrické energie.

Klíčová slova

Formální verifikace, přibližné počítání, aproximované aritmetické obvody, Kartézské genetické programování, přibližná ekvivalence.

Reference

MATYÁŠ, Jiří. *Applications of Formal Methods in Approximate Computing.* Brno, 2023. PhD thesis. Brno University of Technology, Faculty of Information Technology. Supervisor doc. RNDr. Milan Češka, Ph.D.

Rozšířený abstrakt

V současné době přestává platit Mooreův zákon, a tak se objevují a vyvíjí nové alternativní výpočetní přístupy, které mají za úkol zrychlit a zefektivnit výpočetní systémy. Jedním z těchto přístupů je tzv. aproximované počítání, které se snaží urychlit a zefektivnit výpočty za cenu přijatelných nepřesností ve výsledcích. Aplikace, ve kterých je tento přístup nejlépe aplikovatelný, se nazývají *přirozeně odolné vůči chybám.* Mezi tyto oblasti patří především zpracování signálu a multimédií, dolování z dat nebo neuronové sítě. Techniky pro aproximované počítání se postupně vyvinuly na všech úrovních výpočetních systémů – na úrovni výpočetního hardware, pamětí i počítačových programů.

V rámci této práce se zaměřujeme na aproximaci hardwarových aritmetických obvodů. Cílem aproximace je nalézt takové aritmetické obvody, které poskytují co nejlepší rovnováhu mezi chybou výpočtu a úsporou elektrické energie. Aproximace aritmetických obvodů má velký potenciál, protože tyto obvody slouží jako základní stavební kameny mnoha počítačových systémů. Pomocí menších efektivních aproximovaných komponent je tak možné sestavit komplexní aproximovaný systém.

V minulosti byly přibližné obvody navrhovány především manuálně. Manuální návrh ale není dostačující, protože vyžaduje expertní znalosti konkrétní implementace daného obvodu, a často neposkytuje adekvátní úspory vzhledem k velikosti chyb výpočtů. Proto se stále více rozvíjejí automatizované aproximační algoritmy, které si kladou za cíl rychle, efektivně, a bez nutnosti externích zásahů tvořit kvalitní aproximace hardwarových obvodů, a to bez ohledu na jejich funkci, implementaci a velikost. Předmětem této práce jsou tzv. *prohledávací aproximační algoritmy.* Převážná většina těchto algoritmů pracuje iterativně. V každé iteraci se nejprve vytvoří kandidátní aproximovaná řešení (pomocí komponenty zvané syntetizér), a poté se vyhodnotí jejich chyba vzhledem ke správnému řešení (komponenta analyzátor). Pro získání kvalitních aproximovaných obvodů musí prohledávací algoritmy vykonat velké množství těchto iterací. Proto je nutná vysoká efektivita syntetizéru i analyzátoru.

Jako základ pro prohledávací algoritmus pro aproximaci obvodů slouží v rámci této práce specializovaná varianta evolučních algoritmů, tzv. Kartézské genetické programování (CGP), do níž integrujeme metody formální verifikace. Analyzátor, který vyhodnocuje chybu aproximovaného řešení, je akcelerován pomocí speciálního obvodu zvaného aproximační miter. Tento obvod nám umožňuje převést vyhodnocení chyby aproximovaného řešení na problém splnitelnosti Booleovské formule, který je řešitelný pomocí moderních, vysoce efektivních programů zvaných SAT solvery. V rámci našeho výzkumu jsme navrhli optimalizované implementace miterů pro vyhodnocení maximální absolutní a maximální relativní chyby obvodu. Díky použití SAT solverů můžeme implementovat také další nově navrženou prohledávací strategii, která uvaluje limit na prostředky, které může SAT solver využít při vyhodnocování chyby kandidátního řešení. Díky tomuto limitu je evoluční algoritmus motivován hledat rychle verifikovatelná řešení. Výsledkem těchto vylepšení je zrychlení prohledávacího algoritmu a jeho lepší škálovatelnost vzhledem k velikosti aproximovaných obvodů. Navržená prohledávací strategie je první svého druhu, která dokáže úspěšně aproximovat složité obvody velkých bitových šířek (např. 32bitové násobičky) s formální zárukou na velikost aproximační chyby.

V další fázi našeho výzkumu jsme rozšířili strategii, která bere v potaz verifikovatelnost kandidátních řešení, o adaptivní složku. Díky tomu nemusí vývojář pevně stanovit limit prostředků pro vyhodnocení kandidátního řešení. Naopak, evoluční algoritmus sám dokáže určit nejvhodnější limit podle aktuálního průběhu evoluce. V rozsáhlém experimentálním vyhodnocení jsme ukázali, že adaptivní složka prohledávací strategie umožňuje dosáhnout

lepších aproximovaných řešení než pevné nastavení limitu, a to bez ohledu na typ a velikost aproximovaného obvodu a velikost aproximační chyby. Adaptivní strategie dále posouvá možnosti aproximačních algoritmů směrem k automatické aproximaci obecných obvodů.

Použitý prohledávací algoritmus se může během aproximace "zaseknout" v tzv. lokálních optimech. V lokálním optimu evoluce nedokáže pomocí dostupných mutačních operací najít vylepšení aktuálního kandidátního řešení a v dalších generacích již nepřináší žádná zlepšení. Navržené vylepšení syntetizéru, které integruje CGP a optimalizaci pod-obvodů využívající SAT solver, umožňuje evolučnímu algoritmu uniknout z lokálních optim. Díky tomu může algoritmus dále zlepšovat řešení i v případech, v nichž by se původní varianta CGP již dále nezlepšila. Prokládáním CGP aproximace a optimalizace pod-obvodů jsme dosáhli výrazně vyšších úspor během dlouhých aproximačních běhů.

V poslední fázi této práce jsme zkoumali mutační operátory pro CGP dostupné v literatuře a jejich výkonnost při aproximaci aritmetických obvodů. Na základě získaných znalostí jsme syntetizér vylepšili o nový mutační operátor pro CGP, vytvořený speciálně pro co nejefektivnější aproximaci obvodů. Tento operátor kombinuje dva operátory známé z literatury: 1) pro náš účel nejefektivnější obecný mutační operátor SAGM (Single active gene mutation – mutace jednoho aktivního genu) a 2) operátor který se v jednom kroku snaží eliminovat části aproximovaného obvodu. Dohromady tyto dva operátory tvoří kombinaci, která z počátku rychle eliminuje části obvodu, které nejsou potřebné k zachování požadované funkcionality, a poté kandidátní řešení dále vylepšuje pomocí náhodných mutací. Prezentované experimentální vyhodnocení demonstruje, že nově navržený mutační operátor výrazně (a statisticky významně) předčí výkonnost dosud známých operátorů.

Výsledky prezentované v rámci této dizertační práce výrazně vylepšují výkonnost prohledávacích algoritmů pro aproximaci aritmetických obvodů. Díky tomu můžeme získat aproximace obvodů velkých bitových šířek se složitou vnitřní strukturou (např. 32bitové násobičky nebo 128bitové sčítačky), které poskytují doposud nejlepší známý poměr mezi aproximační chybou a spotřebou elektrické energie.

Applications of Formal Methods in Approximate Computing

Declaration

I hereby declare that this thesis was prepared as an original work by the author under the supervision of Prof. Tomáš Vojnar and Assoc. Prof. Milan Češka. I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

. . . . . .
Jiří Matyáš
May 23, 2023

Acknowledgements

I thank my supervisors Assoc. Prof. Milan Češka and Prof. Tomáš Vojnar, my fellow colleague Dr. Vojtěch Mrázek, and the members of the EHW@FIT and VeriFIT research groups for their immense help and guidance during my doctoral studies.

Contents

1 Introduction **4**
1.1 Motivation . . . 4
1.2 Research Objectives . . . 6
1.3 Author's Contribution . . . 7
1.4 Thesis Outline . . . 8

2 State of the Art in Approximate Computing **10**
2.1 Overview of Approximate Computing Methodologies . . . 10
2.1.1 Approximate Storage . . . 11
2.1.2 Approximate Software . . . 11
2.1.3 Frequency and Voltage Scaling . . . 12
2.1.4 Functional Approximation of Hardware Circuits . . . 12
2.2 Approaches to Design of Approximate Circuits . . . 13
2.2.1 Preliminaries – Arithmetic Circuits . . . 13
2.2.2 Manually Designed Approximate Circuits . . . 16
2.2.3 Boolean Rewriting and Approximate High Level Synthesis . . . 18
2.2.4 Netlist Transformation . . . 19
2.2.5 Cartesian Genetic Programming . . . 22
2.3 Non-functional Circuit Metrics . . . 25
2.4 Error Metrics . . . 26
2.4.1 General Error Metrics . . . 26
2.4.2 Arithmetic Error Metrics . . . 27
2.5 Error Metrics Evaluation . . . 28
2.5.1 Simulation . . . 29
2.5.2 Formal Methods for Error Evaluation . . . 29

3 Scalable SAT-based Approximate Equivalence Checking **34**
3.1 Motivation . . . 34
3.2 Search-Based Design of AACs . . . 35
3.2.1 Problem Formulation . . . 35
3.2.2 Cartesian Genetic Programming . . . 36
3.2.3 Verifiability-Driven Search Strategy . . . 37
3.3 SAT-based WCAE Evaluation . . . 38
3.3.1 Checking Worst Case Absolute Error . . . 38
3.3.2 The Proposed Miter Construction . . . 39
3.4 WCAE Experimental Evaluation . . . 41
3.4.1 Experimental Setup . . . 41
3.4.2 16-bit Approximate Multipliers . . . 42

3.4.3 Complex Multipliers . . . 45
3.4.4 Approximate Adders . . . 45
3.5 SAT-based WCRE Evaluation . . . 46
3.5.1 Checking Worst Case Relative Error . . . 47
3.5.2 Variants of the WCRE Miter . . . 48
3.6 WCRE Experimental Evaluation . . . 49
3.6.1 Comparison of the WCRE Miters . . . 49
3.6.2 Circuit Approximation . . . 50
3.7 Conclusion . . . 52

4 Adaptive Verifiability Driven Search Strategy 53
4.1 Motivation . . . 53
4.2 Adaptive Verifiability-driven Optimisation . . . 54
4.3 Evaluation of the Proposed Adaptive Search Approach . . . 56
4.3.1 Computational Setup . . . 57
4.3.2 CGP Parameters . . . 58
4.3.3 Comparison of Adaptive Strategies . . . 60
4.3.4 Reduction of Randomness (Q1) . . . 63
4.3.5 Versatility of Adaptive Strategies (Q2) . . . 65
4.3.6 A Comparison of Adaptive and Fixed-limit Strategies (Q3) . . . 71
4.3.7 Comparison with State-of-the-art Techniques (Q4) . . . 72
4.4 Conclusion . . . 74

5 ADAC - a Framework for Automatic Approximation 75
5.1 Architecture and Implementation . . . 76
5.1.1 Integration to the ABC Tool. . . . 77
5.2 Error Evaluation Methods . . . 77
5.2.1 Bit-parallel Circuit Simulation . . . 77
5.2.2 BDD-based Evaluation . . . 77
5.2.3 SAT-based Evaluation . . . 78
5.3 Error Evaluation Performance . . . 78

6 StS-based Synthesis for Exact and Approximate Circuits 80
6.1 Motivation . . . 80
6.2 StS-based Circuit Approximation . . . 81
6.2.1 A Monolithic Approach . . . 81
6.2.2 Sub-circuit Approximation . . . 82
6.2.3 Evolutionary Approximation with StS-based Optimisation . . . 83
6.3 Experimental Evaluation . . . 84
6.3.1 Performance on Small Circuits . . . 85
6.3.2 Performance on Complex Circuits . . . 85
6.4 Conclusion . . . 87

7 Novel Mutation Operator for Approximate Circuit Design 88
7.1 Motivation . . . 88
7.2 Mutation Operators in CGP . . . 89
7.3 Towards Efficient Mutation . . . 90
7.3.1 Single Active Gene Mutation Operators . . . 90
7.3.2 Node Deactivation Operators . . . 91

7.3.3 Combined Mutation Operators 92
7.4 Experimental Setup 92
7.5 Experimental Results 93
7.5.1 Existing Mutation Operators 94
7.5.2 Mutation vs. Deactivation Operators 95
7.5.3 The Combined Operator SagTree 98
7.6 Conclusion 99

8 Conclusion **100**

Bibliography **102**

Chapter 1

Introduction

The following chapter contains a brief introduction to the doctoral thesis which is the result of the author's studies at the Brno University of Technology during the years 2017-2023. First, we provide the current technical background and motivation for the research efforts. Afterwards, we identify the research goals and objectives to be achieved in the scope of the thesis. At the end, we briefly outline the contents of the rest of the document.

1.1 Motivation

In the past, the performance of computer systems had been steadily increased according to the Moore's Law. Thanks to the shrinking feature size of the complementary metal-oxide-semiconductor (CMOS) components, the number of transistors on a chip roughly doubled every two years. This led to a decrease of the chip's voltage and current and subsequently to increased working frequencies. However, with the sizes of transistors reaching the units of nanometers, power dissipation has become a significant barrier that prevents this scaling from continuing.

At the same time, the recent years saw the energy efficiency of computer systems become one of the biggest challenges in the computer industry. The prolongation of the battery life of mobile devices is one of the critical concerns for consumer electronics. Similarly, the energy consumption of data centres accounts for a major part of their operational costs.

To satisfy the high demand for computer systems with increased computing performance and power efficiency, various new paradigms have been researched. These new methodologies usually exploit specific features of the targeted applications to increase the throughput and reduce the energy consumption. The paradigms have been developed at all levels of the computer systems – the circuit, architecture, operating system, and software levels.

Approximate computing has been established as an emerging research field whose principal goal is to increase the computational performance and reduce the system resource demands. It achieves this goal by relaxing the requirement that its underlying computations are always performed correctly. Approximate computing makes use of the fact that some applications are *error resilient*, that is, producing acceptable results even though the computations are performed with a certain measure of error. Error resilient applications are often related to human perception or have statistical nature – generally areas where the exactness of results tends to not be critical. Prominent applications for approximate computing include image and multimedia processing, signal processing, data mining, machine learning, neural networks, and scientific computations. In error resilient applications,

the error can be used as a design metric and traded for the area on the chip, power consumption, or runtime. Chippa et al. [22] claim that almost 80% of the runtime is spent in procedures that could be approximated.

Several major approaches for approximate computing have been proposed, the most prominent of which include: *CPU voltage reduction* lowering the power consumption at the expense of causing sporadic arithmetic errors, *approximate storage* working with approximate data representation, or *software approximation* employing code simplification techniques or randomised algorithms.

The research efforts described in this thesis build mainly on an approximation technique called the *functional approximation*. The key idea of this approach is to implement the system with a functionality that slightly differs from the original specification, provided that the error is acceptable and the resource consumption (e.g. power consumption or circuit size) is reduced adequately. Such an approximate solution is usually obtained by a procedure that modifies the original (hardware or software) implementation. We will further focus on functional approximation in the context of *combinational hardware circuits*, which is the research focus of this thesis. Combinational circuits, such as adders or multipliers, are especially interesting for approximation, because they are widely used in almost every computer system. Approximate versions of the these circuits can serve as the basic building blocks of larger approximate systems used in many different applications. For example, approximate combinational circuits have recently been utilised in the acceleration of hardware neural networks [3], image compression [1], and digital signal processing [131].

Originally, the functional approximation was performed manually and required an expert knowledge of the initial exact circuit. The expert modified the non-critical parts of the circuit to obtain resource savings at the cost of the introduction of some errors into the circuit's behaviour. Since manual redesign of complex circuits is very demanding and usually does not produce adequate results, automated methods have been designed to functionally approximate both combinational and sequential hardware circuits. These automated methods typically consist of two core components: (i) a *synthesizer* generating candidate designs approximating the original circuit, and (ii) an *analyser* evaluating the quality of the candidates, namely, quantifying the approximation error. The goal of the functional approximation is to find a set of solutions that feature different trade-offs between the error and other design metrics. The set should ideally closely approximate the so-called Pareto optimal front.

With regards to the synthesizer and analyser components, we can generally define the fundamental research challenges as follows. A high quality automated approximation procedure requires a synthesizer that generates suitable candidate solutions and can explore a wide range of possible solutions at the same time. To be efficient, the procedure also needs an analyser that can quickly evaluate the error of the candidate solutions and scales well for larger problem instances.

Several systematic methods in hardware design have recently been developed to approximate energy-critical hardware components. These methods represent different approaches to the synthesizer component. They build on specially designed heuristic procedures that iteratively modify the original correct solution to obtain Pareto fronts of solutions representing the trade-offs between the error and the design metrics. The most well known systematic methodologies are SALSA [125], SASIMI [124] and ABACUS [81]. Generally, these methodologies use heuristic procedures that try to identify the parts of the system that are redundant or can be approximated. However, the space of approximate solutions reachable this way is quite limited.

Recent research has demonstrated [119] that evolutionary and search-based synthesizers are well capable of generating high-quality Pareto fronts of approximate solutions. However, a great number of candidate solutions has to be generated and evaluated in the process. A new solution is usually promptly generated, while the candidate evaluation takes most of the computational time. Therefore, it is essential to evaluate the quality of the candidate solution as quickly as possible. An exact evaluation of the candidate solutions for complex systems is usually replaced by statistical testing [49]. However, as shown in [21, 132], many safety-critical applications favour *provable error bounds* on the resulting approximate circuits, and thus the statistical testing is not sufficient.

To be able to provide bounds on the approximation error, one can, in theory, simulate the circuit on all possible inputs. Unfortunately, such an approach does not scale beyond circuits with more than 12-bit operands even when exploiting modern computing architectures [78]. A similar scalability problem does, in fact, emerge even when using evolutionary optimisation of circuits while preserving their precise functionality. To solve the problem in that case, applications of *formal verification methods* [24, 92, 115] have been proposed. Naturally, attempts to use formal verification methods – including binary decision diagrams (BDDs) [114], Boolean satisfiability (SAT) solving [126], model checking [20], or symbolic computer algebra employing Gröbner bases [34] – have appeared in the design of approximate circuits too. However, these approaches did still not scale beyond approximation of multipliers with 8-bit operands and adders with 16-bit operands.

Overall, there is a decent amount of specialised ad hoc solutions for approximate circuit design offering good results in narrow fields. Only a few general methodologies have been developed but usually they do not supply the designer with satisfactory approximate designs or have a very limited scalability.

1.2 Research Objectives

The principal aim of this thesis is to improve the automated search-based methods for designing high-quality approximate arithmetic circuits. We will focus on enhancing of both of the key steps affecting the performance of the design process:

1. improvements of the **synthesizer** (the algorithm that creates the candidate approximate solutions), and

2. improvements of the **analyser** (the procedure that evaluates the quality of a given candidate approximate solution).

The synthesizer should generate solutions that provide a good trade-off between the approximation error and resource savings, and quickly converge to high quality solutions. With an improved synthesizer, the search based algorithm should require a lower amount of iterations needed to converge to an acceptable solution. While it is not possible for a stochastic search based algorithm to always reach the same high quality solution, the algorithm with a good synthesizer should consistently reach solutions with similar quality.

The analyser should be able to quickly and efficiently evaluate the approximation error of various circuit instances. Ideally, the analyser should support multiple commonly used error metrics and its evaluation speed should scale well for larger bit widths. A powerful analyser allows us to perform more iterations of the search-based method and therefore find better solutions.

The improvements of both synthesizer and analyser should bring us closer to the ideal approximation method: one that is fully automated, can produce high quality approximations quickly, can approximate various circuit instances, scales well for larger circuit instances, and consistently finds solutions of similar quality.

1.3 Author's Contribution

The author's research contributions that were created in the course of their doctoral studies are summarised below. For each item of the list, we also provide the ranking or impact factor of the corresponding conference or journal in the year of publication. The publications comprise the core of this thesis. The author of this thesis took an important part in the formulation of the research ideas, designing methodological approaches, and writing of the articles below. The author played an essential role in the implementation and experimental evaluation of the proposed methodologies.

Conference papers

[13] M. Češka, J. Matyáš, V. Mrazek, L. Sekanina, Z. Vasicek and T. Vojnar. **Approximating Complex Arithmetic Circuits with Formal Error Guarantees: 32-bit Multipliers Accomplished.** In: Proceedings of the 36th IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pages 416–423. IEEE (2017). Rank A in CORE2017. (60 citations according to Google Scholar in March 2023.)

[16] M. Češka, J. Matyáš, V. Mrazek and T. Vojnar. **Satisfiability Solving Meets Evolutionary Optimisation in Designing Approximate Circuits.** In: Proceedings of the 23rd International Conference on Theory and Applications of Satisfiability Testing (SAT), pages 481–491. Springer International Publishing (2020). Rank A in CORE2020.

[18] M. Češka, M. Češka sr., J. Matyáš, A. Pankuch and T. Vojnar. **Approximating Complex Arithmetic Circuits with Guaranteed Worst-Case Relative Error.** In: Proceedings of the 17th International Conference on Computer Aided Systems Theory (Eurocast), pages 482–490. Springer Verlag (2020). Rank B3 in Qualis.

[12] M. Češka, J. Matyáš, V. Mrazek and T. Vojnar. **Designing Approximate Arithmetic Circuits with Combined Error Constraints.** In: Proceedings of the 24th Euromicro Conference on Digital System Design (DSD), pages 785–792. IEEE (2022). Rank B in CORE2021.
This work was finished and published during the writing of this thesis and further extends some of its topics. For the sake of brevity and better coherence, we do not include the results published in this paper in the thesis.

Journal papers

[15] M. Češka, J. Matyáš, V. Mrazek, L. Sekanina, Z. Vasicek and T. Vojnar. **Adaptive verifiability-driven strategy for evolutionary approximation of arithmetic circuits.** In: Applied Soft Computing, Volume 95, Number 106466. Elsevier (2020). Impact Factor 6.65 (Web of Science, 2020) and quartile Q1 according to SJR.

[17] M. Češka, J. Matyáš, V. Mrazek, L. Sekanina, Z. Vasicek and T. Vojnar. **SagTree: Towards Efficient Mutation in Evolutionary Circuit Approximation.** In: Swarm and Evolutionary Computation, Volume 69, Number 100986. Elsevier (2022). Impact Factor 10.26 (Web of Science, 2022) and quartile Q1 according to SJR.

Tool papers

[14] M. Češka, J. Matyáš, V. Mrazek, L. Sekanina, Z. Vasicek and T. Vojnar. **ADAC: Automated Design of Approximate Circuits.** In: Proceedings of the 30th International Conference on Computer Aided Verification (CAV), pages 612–620. Springer International Publishing (2018). Rank A* in CORE2018.

1.4 Thesis Outline

In this section, we briefly outline the structure of the thesis:

Chapter 2 contains a summary of the background knowledge for the area of approximate computing. The chapter overviews the various existing approaches to approximate computing, such as voltage overscaling, approximate storage, and approximate software. A major part of the chapter is dedicated to the focus of this thesis – the field of approximate arithmetic circuits and their design methodologies. These approaches range from manually designed circuits to advanced automatic approaches, e.g. Boolean rewriting or netlist transformation methods. Subsequently, the chapter focuses on both non-functional metrics (e.g. circuit area, delay, etc.) and functional metrics (also error metrics, e.g. Hamming distance, error rate) of approximate arithmetic circuits. Finally, we discuss the existing techniques for error metric evaluation that include full simulation, partial simulation, and utilisation of formal verification techniques (binary decision diagrams, satisfiability solving, etc.).

In **Chapter 3**, we introduce the techniques that aim at the improvement of the existing error evaluation approaches based on Boolean satisfiability solving (SAT). We present novel miter constructions for the evaluation of the worst case absolute error and the worst case relative error using SAT and a newly designed verifiability driven search strategy. We implement the presented techniques into a framework based on the Cartesian Generic Programming and show that they significantly improve the capabilities of the search based approximate arithmetic circuit design. This chapter is based mainly on our work published in [13, 18].

In **Chapter 4**, we continue the development of scalable approximation algorithms with SAT-based error evaluation by introducing the adaptive verifiability driven search strategy. This strategy is meant to provide a good performance in the approximation of arithmetic circuits independently of the actual type and size of the approximated circuit. We propose several settings for the adaptive approximation framework, choose the best ones and then present an extensive experimental evaluation of the verifiability driven strategies. The evaluation is performed on both the traditional and more complex arithmetic circuits of various bit widths. Finally, we show the improvements in the quality of the approximate solutions the adaptive strategy brings in comparison to the research presented in the previous chapter. This chapter contains the work published in [15].

Chapter 5 contains a description of the ADAC tool, which implements the algorithms and techniques shown in Chapters 3 and 4 and is utilised for the presented experimental evaluations. ADAC is an extension of the academic hardware synthesis and verification tool ABC and makes use of the ABC's internal circuit representation structures and powerful

solving algorithms. After a description of the ADAC structure and inner workings, the chapter also contains a short experimental section showing the speed of the various error evaluation methods available in ADAC. A tool paper about ADAC was published in [14].

In **Chapter 6**, we present another extension of the CGP based approximation algorithm. We combine a satisfiability based exact synthesis optimisation approach with the approximation techniques described in Chapters 3 and 4. The exact optimisation rewrites parts of the approximated circuit and helps the evolutionary algorithm to escape local optimums, thus allowing for further improvements of the quality of the approximate solutions. The material in this chapter is based on our work [16].

Chapter 7 contains our work that also tries to improve the approximation algorithm, albeit by different means than the other chapters. While in Chapters 3 and 4, we aimed at the speed up of the error evaluation part of the approximation process, in this chapter we try to improve the mutation operators utilised in the CGP algorithm. We firstly summarise the various CGP mutation operators utilised in the literature and then propose a novel operator tailored to the task of arithmetic circuit approximation. In the experimental evaluation, we focus both on the speed of convergence of the mutation operators as well as on the quality of the final approximate solutions. The contents of this chapter are based on our work published in [17].

Chapter 8 summarises the contents of the thesis and also mentions some promising research directions for the future work.

Chapter 2

State of the Art in Approximate Computing

In this chapter, we provide an introduction of the state of the art of the various techniques and approaches used in the area of approximate computing. After this initial broad overview, we focus more closely on the topics crucial to this thesis, namely the functional approximation and search based methods for the design of the approximate arithmetic circuits. The performance of these methods is directly affected by the performance of the candidate circuits evaluation. Therefore, in the following sections, we also survey existing methods for the evaluation of the error of approximate circuits – simulation as well as alternative approaches based on formal methods.

Various approaches have been proposed to address the problem of rapidly growing energy consumption of modern computer systems. As one of the most promising energy-efficient computing paradigms, approximate computing has been introduced [73]. Approximate computing intentionally introduces errors into the computing process in order to improve its energy-efficiency. This technique targets especially applications featuring an intrinsic error resilience property where significant energy savings can be achieved. The inherent error resilience means that it is not always necessary to perform the precise and usually resource-expensive computations. Instead, much simpler approximate computations may be used to solve a given problem without any significant degradation in the output quality. Multimedia signal processing and machine learning represent typical examples that allow the quality to be traded for power, but approximate computing is not limited to those applications only.

2.1 Overview of Approximate Computing Methodologies

Many fundamentally different approaches have recently been introduced under the term of approximate computing. The literature on the subject covers the whole computing stack, integrating areas of microelectronics, circuits, components, architectures, networks, operating systems, compilers, and applications. Approximations are conducted for embedded systems, ordinary computers, graphics processing units, and even field-programmable gate arrays. A good survey of existing techniques can be found, for example, in [73, 130].

2.1.1 Approximate Storage

Approximate RAM: the techniques belonging to this category modify the parameters and behaviour of RAM modules to decrease their energy consumption. As a result of these changes, the values retrieved from the memory are not guaranteed to exactly correspond to the values stored. The work published in [99] presents an energy saving SRAM cache using lowering voltage supply and power-gating. Another work [23] focuses on eDRAM and proposes a technique for saving the refresh energy in the frame buffers for video applications. The technique trades off pixel data accuracy for energy savings in the frame buffers. Similarly, another approach [65] divides the data into critical and non-critical parts, which are stored in different memory modules. The module with the non-critical data can feature a lower refresh rate that consumes less energy but can introduce errors into the stored data.

Load value approximation: rather than approximating the stored values themselves, with this approach, the approximation happens when a load miss in a cache occurs. After such a miss, the data would need to be fetched from the next level cache or memory, which is a time consuming operation. Instead of waiting for the data to load, an estimator provides a predicted block of data. Load value approximation hides the cache miss latency and allows the processor to continue without waiting for the data to load. The technique was successfully employed in various applications for both CPUs and GPUs [69, 108].

2.1.2 Approximate Software

The *loop perforation* approach reduces the computational complexity of programs by skipping some iterations of selected loops. Several common computational patterns suitable for loop perforation were identified in [100]. These patterns include Monte Carlo simulation, iterative refinement and search space enumeration. The loop perforation algorithm explores the combinations of possible loop skipping in the source program and tries to maximise the performance within acceptable result quality bounds.

Skipping tasks and memory accesses: these approaches selectively skip tasks, memory accesses or input portions to achieve better efficiency while not breaching the bounds on the quality of results. An example of such approach is Paraprox [90] – pattern-based approximation for data parallel applications. In the publication, the authors examine six common programming patterns (scatter/gather, map, scan, reduction, stencil, and partition) suitable for execution on multi-core architectures. For example, the reduction pattern is approximated using sampling and its output is therefore computed using only a subset of the original data. Similarly, in the scan pattern, the scan is performed only on the beginning of the input array to produce an intermediate result and the final result is then computed from the intermediate one without considering the rest of the input.

Multiple inexact program versions: in [5] and [91] the authors present techniques to create approximate versions of programs for general CPUs and GPUs, respectively. Using the correct versions of the program and the desired result quality as inputs, the approaches first generate multiple versions of the approximated program using e.g. loop perforation or fusion of CUDA threads. Then a training and optimisation algorithm examines the quality and efficiency of the created approximations and improves and identifies the ones providing the best trade-offs. During runtime, the frameworks periodically monitor the quality of the computed approximate results and update the approximate program if the result quality falls below a user defined threshold.

Neural network based accelerators [31] expose the neural network's (NN) parallelism and can be used to increase the efficiency of the accelerated operations. The accelerators

work by identifying the approximable parts of the code and then training a suitable neural network to mimic the functionality of the code region. NN accelerators can be executed on general purpose CPUs or specialised hardware components (neural processing units) for increased efficiency.

2.1.3 Frequency and Voltage Scaling

Frequency and voltage scaling utilise the correctly working digital circuits and cause inaccuracies in the computations by tuning the frequency and voltage parameters. In *voltage over-scaling*, the inaccuracies are introduced by scaling the supply voltage of the logic gates of the circuit. A lower voltage supply causes timing errors in some non critical paths of the circuit, creating acceptable approximate results. The main disadvantage of this approach is, that the error characteristic of such approximate operation is affected by parametric variations and is therefore nondeterministic.

Most of the works published in this area provide methods designed to approximate computations in specific applications. In [44, 98], the authors approximate digital filters. To improve the energy versus quality of results trade-off, these works utilise error correction mechanisms (e.g., specialised error correction blocks or redundant components). In [43], the authors focus on approximation of arithmetic units. Instead of utilising dedicated logic to mitigate the approximation errors, the quality of the result is controlled using expert knowledge of the statistical properties of arithmetic operands. While these publications promise good trade-offs between energy savings and result quality, they are only applicable in their specific domains and cannot be easily modified to fit other applications.

In [67], the authors present a more general methodology to approximate any sequential circuits using voltage scaling. The designer controls the maximum acceptable error by defining the maximum error probability at each register of the given circuit. The framework then examines the paths of the circuit, determines which paths fail at which voltage and finally estimates the error occurrence probability in the circuit's registers. Using this information, the framework can infer voltage supply levels for each timing path.

Frequency upscaling works on similar principles to voltage scaling, but instead of lowering the voltage supply, this approach investigates the performance of circuits subjected to a higher input frequency. The increase of the input frequency can lead to an improvement of the overall processing speed of the system. However, when the input frequency exceeds the maximal operational frequency of the circuit, timing violations can be introduced along the circuit's longest paths. In [60], the authors experimented with full adder cells and examined the errors at their outputs when the input frequency was increased beyond the maximal correctly operating frequency. The error rate of the sum and carry bits gradually grew with growing input frequency. The work published in [47] extends this approach and proposes a mathematical model that analyses the behaviour of adders under frequency upscaling.

2.1.4 Functional Approximation of Hardware Circuits

In the previous section, we have seen that the frequency and voltage scaling approaches make use of correctly working circuits and introduce errors into the computations by changing the external parameters of voltage supply or frequency of inputs. The approximate circuits take the approximate computing paradigm one step further – their principal goal is to design circuits whose function differs from the original specification. This approach is also often called the *functional approximation* of hardware circuits.

Functional approximation is an approach where the original circuit is replaced by a less complex one which exhibits some errors but improves the non-functional circuit parameters such as power consumption or chip area. Circuit approximation can be formulated as a multi-objective optimisation problem where the error and non-functional circuit parameters are the conflicting design objectives. Since the resulting approximate circuits are still common circuits, they can be implemented using the standard circuit design flow. In contrast to the frequency and voltage scaling, the approximate circuits designed using functional approximation feature deterministic error characteristics. This allows an easier analysis of the behaviour of the approximate circuits in the target application.

Functional approximation of circuits can be performed manually, but the current trend is to develop fully automated functional approximation methods that can be integrated into computer-aided design tools for digital circuits. The fully-automated methods typically employ various heuristics to identify the circuit parts suitable for approximation.

Even though almost any hardware circuit can be approximated, much of the research effort focuses on the *approximate arithmetic circuits*. Such circuits can be used as the basic building blocks in key applications relevant for approximate computing. Prominent examples are signal, image, and video processing circuits (such as filters, discrete transforms, and motion estimation blocks [97]), or the multiply-accumulate-transform structures of artificial neurons in neural networks (consuming about 50% of the total power in neural network accelerators [53]).

In this section, we provided an overview of the state of the art in the area of approximate computing and put the main focus of this work – the functional approximation of arithmetic circuits – into the context of this research field. In the next section, we will study the approximate circuits and the various approaches to their design in greater depth.

2.2 Approaches to Design of Approximate Circuits

There are many diverse approaches to designing approximate circuits based on the various goals of the designers and target applications. A survey published in [51] provides a broad overview of methods utilised in the design of approximate circuits, as well as the characterisation of multiple classes of approximate circuits and their performance in various applications. While some existing approaches try to design approximate circuits manually, this technique becomes impractical as the research moves to larger and more complex circuits. This leads to a growing interest in the area of automatic methods for the design of approximate circuits (also called *approximate logic synthesis* – ALS). An ideal design method would be able to approximate a wide range of different circuits while being fully automated. A recent survey on approximate computing [93] divides the current research efforts in approximate logic synthesis into three categories: (1) approximate high level synthesis, (2) Boolean rewriting, and (3) netlist transformation. In the following subsections, we will first take a look at the inner workings of the basic arithmetic circuits. Afterwards, we explain the basic concepts of the common approaches to approximate circuit design and mention some of the key research works in the area.

2.2.1 Preliminaries – Arithmetic Circuits

The arithmetic circuits that are most often approximated in literature are undoubtedly the binary adders and multipliers. The basic building blocks for these circuits are the so called half adder (HA) and full adder (FA). The gate level schematic of HA is visualised in

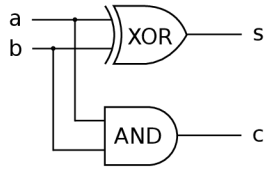


Figure 2.1: A gate level representation of a half adder.

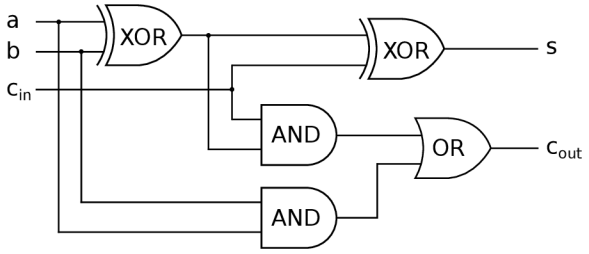


Figure 2.2: A gate level representation of a full adder.

Fig. 2.1 and the schematic of FA is shown in Fig. 2.2. A HA computes the sum of two single binary digits, a FA computes the sum of three input single binary digits. The result of both circuits is represented by a 2-bit binary number. The lower bit of the result is denoted as *sum* (s), the higher bit is denoted as *carry* (c).

The basic circuit designs used to add two n-bit binary numbers a and b are the ripple-carry adder (RCA) and the carry lookahead adder (CLA). An n-bit RCA is constructed by connecting n full adders. The FA on i-th position performs the sum of the i-th bits of input – a_i and b_i and the carry output of the previous FA c_{i-1}. If the input carry of the addition operation is expected to be zero, we can substitute the first FA with a HA. The structure of RCA is rather simple, as can be seen in Fig. 2.3. However, RCAs are relatively slow, because each FA has to wait for the carry of the previous FA to be computed. The critical path of the circuit spans from the inputs of the first HA to the outputs of the last FA.

To reduce the delay of the adder circuit, a faster way to add two binary digits was designed in the form of CLA. The CLA structure computes two signals P and G for each bit position. Depending on the values of P and G for each position, the carry in that position is either 1) generated (both signals are in logical 1), 2) propagated from the less significant position (exactly one signal is 1), or 3) killed (both signals are 0). Once P and G signals are computed, we can determine the carry values in each position and subsequently compute the result. The structure of CLA is more complex than that of RCA, but the added complexity leads to a lower computational delay. The delay and circuit size of RCA increase

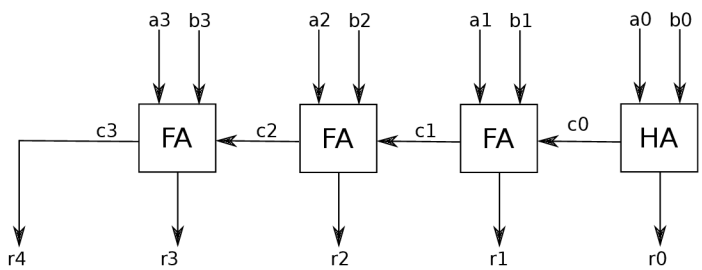


Figure 2.3: A 4-bit ripple carry adder consisting of full adder and half adder blocks.

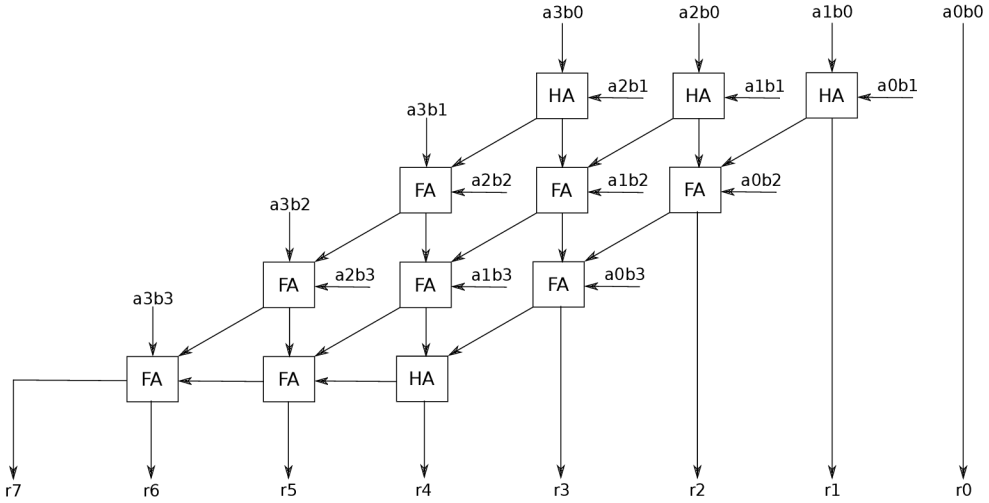


Figure 2.4: A drawing of the partial product accumulation and final addition in a 4-bit array multiplier. The multiplier is composed of half adder and full adder blocks. The generation of the partial products is omitted from the figure – the partial products generated using AND gates serve as the inputs in the drawing.

proportionally with n ($O(n)$). On the other hand, the delay of CLA is roughly logarithmic ($O(log(n))$), while its size is linearithmic ($O(n * log(n))$). For input widths exceeding 32 bits, the classic CLA structure becomes inefficient due to the increasing fan-in of the gates computing the P and G signals. Therefore, lookahead structures with multiple levels have been introduced to increase the speed and reduce the circuit area. Prominent examples of these CLA implementations are the Kogge-Stone adder [58] and Brent-Kung adder [8]. Other adder designs include e.g. carry-skip adder, carry-save adder or carry-select adder. The inner workings and characteristics of these adders are thoroughly addressed in [30].

The other most commonly approximated arithmetic circuit is the binary multiplier. Multipliers exists in both signed and unsigned variants, in this work, we focus mainly on the latter. An n-bit binary multiplier is an arithmetic circuits that takes two n-bit binary numbers a and b and computes their product – a $2n$-bit binary number – as the result. A slower multiplier variant with a less complex structure is the shift-and-add multiplier, that iteratively shifts and accumulates partial results. Such a multiplier requires multiple cycles to compute the result. On the other hand, the faster single cycle multiplier is a purely combinational circuit. In a typical combinational multiplier, the multiplication process consists of three steps: partial product generation, partial product accumulation, and carry propagate addition. The partial products (PP) can be computed by simply using an AND gate ($PP_{i,j} = a_i \; and \; b_j$). The partial product accumulation is the most costly operation of the three steps and takes up most of the multiplier's area. The structures commonly used to achieve partial product accumulation are: 1) adder array [82], 2) Wallace tree [128], and 3) Dadda Tree [25].

The structure of the multiplier with adder array is based on the shift-and-add algorithm. Because an n-bit multiplier has to complete the computation in one cycle, it needs n adders to accumulate the partial products. The carry signals propagate through the adder array in a diagonal direction. An example array multiplier is visualised in Fig. 2.4. We can see that the area of the multiplier is quadratic in the number of input bits ($O(n^2)$. The delay of array multiplier is roughly linear in n ($O(n)$).

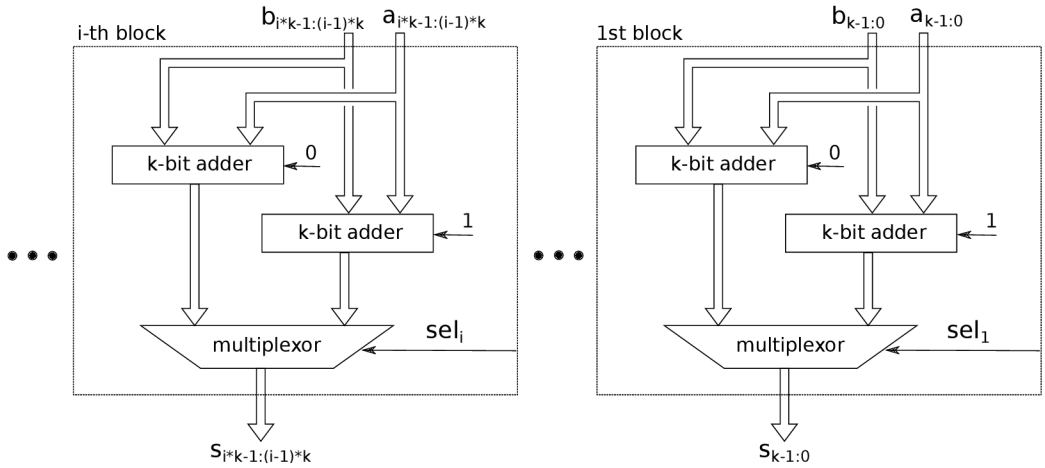


Figure 2.5: An illustrative diagram of approximate carry select adder.

To reduce the time needed to accumulate the partial products, tree structures of adders were implemented in Wallace and Dadda multipliers. These structures reduce the number of partial products until there are only two numbers left, which are then accumulated using a conventional adder. The tree structure reduces the circuit's delay, but requires a larger amount of gates, resulting in a bigger circuit area and power consumption.

2.2.2 Manually Designed Approximate Circuits

The original works in the area of approximate circuits focused on manual circuit design. The researchers selected a specific hardware circuit, usually an arithmetic circuit such as adder, multiplier, or divider and then tried to change the structure of the circuit in such a way, that only slightly modifies the functionality while bringing significant resource savings. The result of these research efforts was either a single inexact version of the original arithmetic unit (e.g. approximate 2-bit multiplier), or an approximate implementation with configurable bit width (e.g. k-bit approximate adder).

The simplest approach to designing approximate adders and multipliers is circuit truncation. The truncated circuit only computes n most significant bits of the result and the rest are hard wired with logical zeros. The logic evaluating the omitted lower bits can be eliminated from the circuit. Additionally, the eliminated logic can instead be substituted with a simpler correction / compensation logic that reduces the truncation error.

Approximate Adders

The literature contains many approximate adder designs. The smallest approximable arithmetic circuit units are the half adder (providing the sum of two input bits), and the full adder (computing the sum of three input bits). The half and full adders serve as a basic building block for more complex arithmetic circuits. The authors of [85] propose several manually designed gate-level implementations of an approximate full adder. The approximate full adder cells can be used to construct an optional bit width approximate adder. For example, one of the proposed designs is based on the structure of RCA. For an n-bit approximate adder of this design, the lower k bits of the result are computed using approximate FAs, while the higher $(n - k)$ bits are computed using standard FAs [66].

Larger approximate adders can be divided into categories based on the method used to modify the circuit structure and shorten its critical path. These categories include speculative adders, segmented adders and approximate carry select adders [49]. *Speculative adders* [127] exploit the fact that the length of the carry chain utilised in most computations is much shorter than the whole bit width of the adder. Therefore, the carry bits for each sum bit can be predicted using a certain number of lower bits. This introduces some extra logic but at the same time significantly shortens the critical path of the circuit. The scheme used in *segmented adders* [74] divides the n-bit adder into a number of smaller sub-adders. During the addition, the sub-adders operate in parallel with their carry inputs either fixed, or generated by some auxiliary logic. The *approximate carry select adders* [28, 57] utilise the structure of a carry select adder, illustrated in Fig. 2.5. Similarly to segmented adders, the n-bit circuit is also divided into m smaller k-bit adder blocks. The addition in each block is performed twice – each of the blocks contains two k-bit adders, one with input carry equal to 1, the other with input carry 0. A multiplexor then selects which of the results is correct using a selection signal *sel*. The value of *sel* is determined by the carry prediction logic of the previous block. The structure of the prediction logic is similar to the carry generation logic of the carry lookahead adder. Again, this technique shortens the critical path of the circuit while introducing errors in computations where the carry prediction is inaccurate.

Approximate Multipliers

Multiplier circuits usually consist of three stages – partial product generation, partial product accumulation and final addition. One can introduce approximations in the generation of the partial products [59, 66], in the tree used for the accumulation of the partial products [6], or in the components [63] that accumulate the partial products.

In [66], the authors create an algorithmic approach to approximate the partial product accumulation of an array multiplier. The resulting approximate implementations are called Broken Array Multipliers (BAMs). The BAM approach is similar to truncation and can be applied to multipliers of universal bit widths. The designer can select the target approximation error by specifying the *horizontal breaking level* and *vertical breaking level* parameters, both with values between zero and the multiplier's bit width. The least significant bits (the ones below the breaking level) of the partial products are then omitted in both vertical and horizontal fashion, resulting significant in power and area savings. Fig. 2.6 illustrates the structure of a 7-bit multiplier with horizontal breaking level 2 and vertical breaking level 5. Each circle of the figure represents a single adder cell accumulating the partial products. The adder cells omitted from the final approximate circuit are coloured red, the cells present in the final circuit have green colour. The outputs of all excluded cells are considered to be 0. With increasing horizontal and vertical breaking level values, more adder cells are left out from the circuit, resulting in a smaller approximate multiplier with a larger approximation error. A similar technique can be used to build approximate multipliers from other basic multiplier implementations (such as Wallace and Dadda multipliers).

More complex approximate circuits can be constructed by using the *composition* of approximate elementary blocks. For example, a 2-bit multiplier was approximated in [59] and then used as a building block of more complex multipliers. This strategy can be improved, e.g., by configurable lossy compression of the partial product rows based on their progressive bit significance [84]. Similarly, approximate full adders were composed to create large approximate adder implementations in [85].

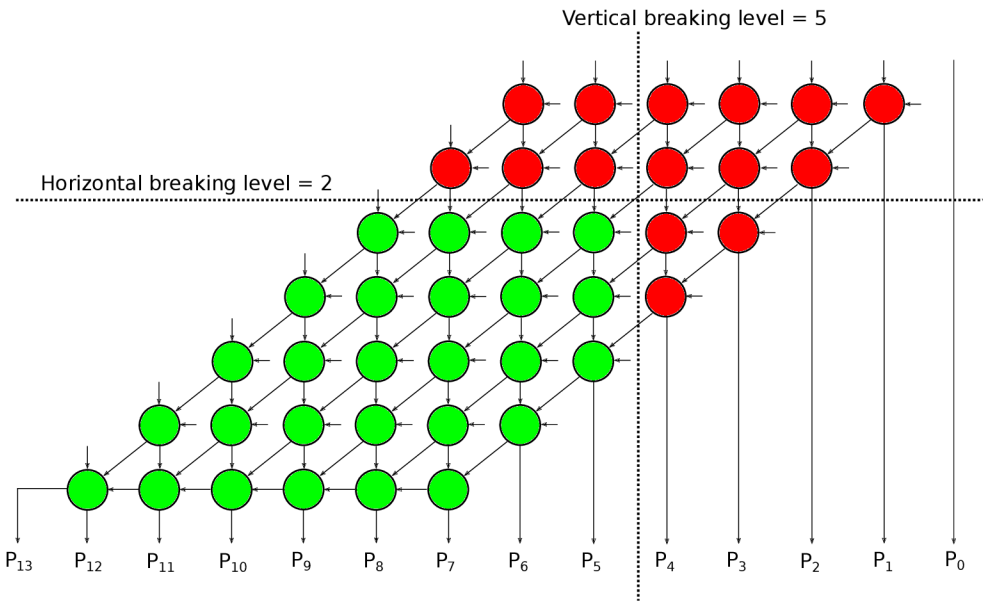


Figure 2.6: A diagram of the structure of a Broken Array Multiplier with horizontal breaking level = 2 vertical breaking level = 5.

Quality Configurable Circuits

All the circuits mentioned so far in this section featured constant behaviour with known error characteristics. However, in some applications, a dynamic adjustment of the approximation error and power consumption might be desirable. The concept of *quality configurable circuits* uses elementary circuits composed in such a way that their error can be modified online using several configuration bits in order to dynamically reduce the power consumption. The configuration bits can (dis)connect some pre-selected parts of the circuit. The main advantage of the quality configurable adders [96] and multipliers [97] is the fact, that the circuit can behave as a correct implementation and can also dynamically change the approximation error based on the currently desired quality of result.

2.2.3 Boolean Rewriting and Approximate High Level Synthesis

Boolean rewriting and approximate high level synthesis (AHLS) are automated approaches to approximate logic synthesis with a higher level of abstraction. These methods do not rely on any electrical components or implementation technology and thus are more abstract than i.e. a gate level representation. We will provide only a brief overview of these methods, as they are not the principal aim of this thesis. For further information, readers should refer to e.g. the survey [93].

To utilise the Boolean rewriting techniques in circuit approximation, we first need to describe the functionality of the circuit using a formal Boolean representation – for example a Boolean formula, binary decision diagram (BDD), or and-inverter graph (AIG). This representation is then modified by the rewriting algorithm to create an approximate representation. Optionally, the approximate version can then be mapped into a target technology to precisely determine its non-functional parameters, such as circuit area or power consumption.

The Systematic methodology for Automatic Logic Synthesis of Approximate circuits (SALSA) [125] is one of the first approaches that address the problem of approximate synthesis using Boolean optimisation. The authors mapped the problem of approximate synthesis into an equivalent problem of the traditional logic synthesis: the *don't care-based optimisation*. The approach introduced in SALSA was extended in ASLAN [86] to also include sequential circuits, where the error behaviour of the approximate circuit has to be computed over multiple sequential cycles.

Logic rewriting using BDDs [35] and AIGs [21] starts by transforming the circuit representation into BDD or AIG, respectively. BDD rewriting then tries to identify nodes of the diagram that can be replaced by other nodes, or hardwired logical '1' or '0' while still satisfying the desired quality of results. Iterative application of these transformations gradually reduces the size of the BDD. Similarly, the AIG rewriting approach also achieves size reduction by replacing parts of the graph (the so called *cuts*) by their approximate versions.

Logic rewriting using Boolean matrix factorisation was introduced in BLASYS [42]. In this approach, the circuit's function is represented by a matrix M that contains the truth values of the circuit's outputs. Matrix M is then factored into two smaller matrices B and C, whose product approximates the original matrix. The sizes of B and C (and thus the amount of approximation) are controlled by the *factorisation degree* parameter. Matrices B and C are then used to synthesise the approximate solution.

The goal of approximate high-level synthesis is to implement efficient approximations of circuits described using hardware design languages, such as Verilog or C. One of the most well known methodologies for AHLS is ABACUS [80]. ABACUS first parses the input Verilog design to create an internal abstract representation. Then it tries to simplify and approximate the design by using several transformation operators: (1) truncating least significant bits of signals, (2) replacing exact arithmetic operations with approximate ones, (3) transformation of arithmetic expressions, (4) replacing variables with constants, and (5) transformation of loops. As the enumeration of the whole state space of transformation applications is not feasible, ABACUS executes the transformations in a random fashion and creates multiple approximate designs. In the end, a Pareto front of designs providing the best trade-offs between their accuracy and power consumption can be identified.

Another example of AHLS is autoAx [76], which utilises a library of approximate components to build approximate implementations of filters and accelerators. AutoAx provides an effective search algorithm that can select and combine approximate components to create high quality approximate designs.

2.2.4 Netlist Transformation

The netlist transformation methods for approximate logic synthesis change the structure of the hardware circuits similarly to the manual design approaches we described earlier. However, as netlist transformation is a part of ALS, one of its main aims is full automation. Also, the general-purpose approximation methods aim at automatically approximating circuits independently of their structure.

In this family of approaches, the Boolean function we want to approximate is already mapped in a *netlist*, which is a directed acyclic graph with electronic components as nodes and wires as edges. The netlist mapping can be obtained by one of the existing tools for hardware synthesis (e.g. Yosys – an open source hardware synthesis and verification tool). One of the most commonly used types of netlists is the gate-level netlist – a forward prop-

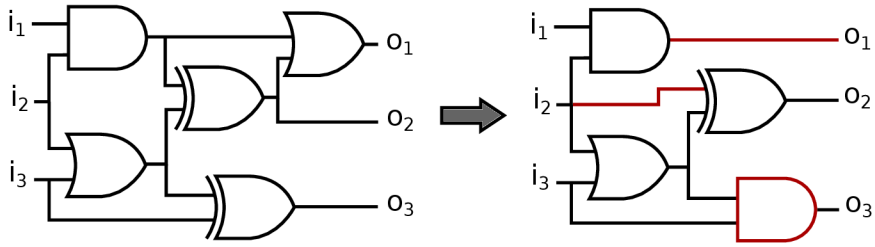


Figure 2.7: An example of netlist transformation.

agating network of logical gates such as AND, OR, NOT, etc. The methods for netlist transformation can remove gates of the netlist, add new gates, or even change the interconnections between the gates. An example netlist and its transformed variant are shown in Fig. 2.7.

The design objective of netlist transformation is usually defined as follows: given an original, correctly working circuit c and an error threshold T, find an approximate circuit (or a set of approximate circuits) c' that satisfies the error threshold T and provides area or energy savings with regards to c. For pruning methods, the netlist of c' is a subgraph of c, as some of the gates (nodes) and connections (edges) were pruned. Other approaches are able to create new structures in the netlist during the approximation process, so the subgraph assertion of pruning does not necessarily hold.

Netlist Pruning

Netlist pruning methods try to simplify the approximated circuit by removing some of its gates. An example of the netlist pruning approaches is Gate-Level Pruning (GLP) [94]. GLP employs a greedy approach to gate elimination. The decision to prune a certain gate is based on two parameters of the gate: significance and activity. The significance is first assigned to the circuit's outputs (the most significant bit has the highest significance) . The netlist is then traversed from the outputs to inputs and the significance of each gate is computed as the sum of the significance of its successors. The gate's activity is obtained through gate-level simulation on a representative sample of the circuit's inputs. The algorithm iteratively removes the gate with lowest significance-activity-product. After each gate removal, the approach evaluates the functional parameters of the circuit and examines, whether the approximate implementation satisfies a predetermined error threshold. Once the threshold is reached, the algorithm terminates and produces the final approximate solution.

Another proposed netlist pruning method called Circuit Carving [2] disposes of the iterative approximation approach. Instead, it enumerates all the possible pruned versions of the original circuit and tries to find the version satisfying the error threshold with a minimal number of gates. To explore the design space, the algorithm utilises a binary search tree to represent the subgraphs of the original circuit. The nodes in each level of the tree correspond to a single gate. The edges leading from nodes are labeled 1 and 0. Edge 1 denotes inclusion of the corresponding gate in the subgraph, while edge 0 denotes exclusion of the gate from the subgraph. The paths from the root of the tree to the leaves enumerate all possible subsets of gates of the original circuit. Since the number of the subsets is exponential in the original number of gates, the authors employ several methods to reduce search space. These are (1) exclude the subsets that violate error threshold and (2) exclude the subgraphs that cannot be minimal (there exists smaller circuit with the same error

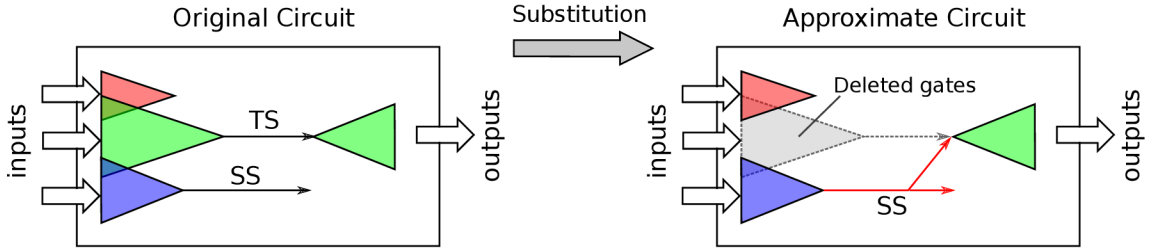


Figure 2.8: Illustration of signal substitution and simplification performed by SASIMI. TS denotes the target signal, SS denotes the substitute signal.

or better error). Although the reduction methods prune the search tree considerably, the approach is still not able to enumerate the whole search space for larger designs.

Greedy Netlist Transformation

The Substitute-And-SIMplIfy (SASIMI) [124] approach is a step further from the pruning techniques. SASIMI also removes parts of the approximated netlist to achieve area and energy savings, but also creates new gate interconnections in the process. The approach tries to identify signal pairs in the circuit that exhibit the same value with a high probability, and substitutes one (substitute signal) for the other (target signal). These substitutions introduce functional approximations. The unused logic that was previously exclusively used to compute the target signal can be eliminated from the circuit, which results in area and power savings. When selecting the signal to substitute, the algorithm takes into account the potential error caused by the substitution and the potential resource savings. Also, the substitutions must not create cycles in the circuit logic. An example of a signal substitution and the resulting circuit simplification is visualised in Fig. 2.8 An extension of the described substitution approach can be utilised to design configurable approximate circuits with multiple quality modes (one of these modes might even be accurate).

Search Based Netlist Approximation

The main limitation of the techniques based on the variants of probabilistic or deterministic pruning is the inability to generate novel circuit structures. None of the previously described techniques allows one to replace a part of the original circuit with a sub-circuit that does not form a part of the original circuit. This limitation considerably restricts the space of the possible solutions as shown in [77]. In order to address this issue and improve the quality of the obtained approximate circuits, various artificial intelligence techniques have been applied to design the approximations.

In order to approximate gate-level digital circuits, Sekanina and Vasicek employed a specialised variant of the Cartesian Genetic Programming [116, 119]. As shown in [77], this approach is able to produce high-quality approximate circuits that are unreachable by the traditional approximation techniques. As a result, a comprehensive library EvoApproxLib consisting of 8-bit adders and multipliers was built using multi-objective CGP.

Statistically Certified Approximate Logic Synthesis (SCALS) [64] represents another search based netlist approximation approach. The framework creates approximate versions of an original circuit by applying exact and approximate transforms to its netlist. The exact transforms include balancing, refactoring and rewriting of a part of the netlist. The approximate transformations can remove gate interconnections, change gate functionality

or even add new gates to the netlist. After some transformations are applied, the framework performs hypothesis testing of the error constraints, providing statistical guarantees on the approximation error.

The proposed search-based approaches share a common idea — they map the problem of approximate synthesis to a search-based design problem. An automated circuit approximation procedure is seen as a multi-objective search process. In the process, a circuit satisfying user-defined constraints (describing the desired trade-off between the quality and other electrical parameters) is sought within the space of all possible implementations. The approximation process typically starts with a fully-functional circuit and a target error. A heuristic procedure (e.g. an evolutionary algorithm) then gradually modifies the original circuit. This procedure is typically repeated iteratively in order to improve the current approximate implementation in the subsequent steps.

The modification can affect either the node function (e.g. an AND node can be modified to an inverter or vice versa), node input connection, or primary output connection. It is thus able to not only disconnect gates but also to introduce new gates (by activating redundant gates). Both can happen by changing either the primary output connection or node input connection or by changing the node function resulting in an increase/decrease of its arity.

2.2.5 Cartesian Genetic Programming

Cartesian Genetic Programming (CGP), mentioned in the previous section as one of the search based methods for netlist transformation, is one of the key focuses of this thesis. Therefore, we provide a detailed introduction of CGP in this section. CGP is a form of genetic programming where the candidate solutions are represented as a string of integers of a fixed length that is mapped to a directed acyclic graph [71]. This integer representation is called a *chromosome*. CGP can efficiently represent common computational structures including mathematical equations, computer programs, neural networks, and digital circuits. The candidate solutions are typically represented in a two-dimensional array of programmable two-input nodes.

Since its introduction, CGP remains one of the most powerful techniques in the domain of evolutionary-based logic synthesis and optimisation [71, 72]. Its adoption in the area of approximate computing seems to be natural because circuit approximation can be formulated as a multi-objective optimisation problem where the error and non-functional circuit parameters are the conflicting design objectives [119, 77].

Algorithm Parameters

CGP models a candidate circuit having n_i primary inputs and n_o primary outputs as a two dimensional array with r rows and c columns and thus $r * c = n_n$ configurable nodes. Each node has n_a inputs and represents a single gate with up to n_a input connections. Two-input and single-output nodes are typically used. To avoid feedback connections, the inputs can be connected either to the output of a node placed in the previous L columns, for some given L, or directly to primary inputs. The function of a node can be chosen from a set Γ consisting of $|\Gamma| = n_f$ functions. Depending on the function of a node, some of its inputs may become redundant. In addition to that, some of the nodes may become redundant because they are not referenced by any node connected to a primary output. The fixed number of nodes n_n does not mean that all the nodes are effectively used. This property is a direct consequence of the redundant encoding used in CGP, which enables the existence of

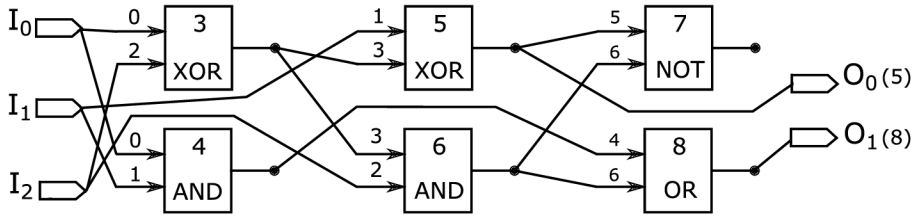


Figure 2.9: Full adder represented by CGP. Chromosome: (0, 2, 2) (0, 1, 0) (1, 3, 2) (3, 2, 0) (5, 6, 3) (4, 6, 1) (5, 8), node functions: AND (0), OR (1), XOR (2), NOT (3). The adder is encoded as five active nodes, gate 7 does not participate in in the computation of the circuit's outputs – it is *inactive*.

neutral mutations. According to many studies, this neutrality is important for an effective search in CGP [123].

Circuit Representation

The candidate circuits are encoded as follows. Each primary input as well as each node is associated with a unique index. Each node is encoded using $n_a + 1$ integers $(x_1, \cdots, x_{n_a}, f)$ where the first n_a integers denote the indices of its fan-ins, and the last integer determines the function of that node. Every candidate circuit is encoded using $n_n(n_a + 1) + n_o$ integers where the last n_o integers specify the indices corresponding with each primary output. Fig. 2.9 illustrates the CGP encoding on an example full adder circuit.

Search Algorithm

The most common search technique used in connection with the CGP is inspired by the $(1 + \lambda)$ evolutionary strategy [71] where λ corresponds with the number of new candidate solutions generated from a single parental solution. In circuit optimization and approximation, the initial population is seeded by the original circuit to be optimized or approximated. Every new population consists of the best circuit chosen from the previous population and its λ offspring created using a mutation operator. Either point or probabilistic mutation is used in the standard CGP. Point mutation is typically preferred because it is easier to implement and more efficient than using a probabilistic mutation [72]. Crossover is not used in the standard CGP because it was found that crossover has little effect on the efficiency of CGP [72].

The selection of the individuals is based on a fitness function evaluating the quality of the candidate circuits, i.e. the precision of the circuit and its size. An individual with the best fitness is selected as the new parent. In the case that there are more individuals with the same fitness, the individual that has not served as a parent will be selected as the new parent. This procedure is typically repeated for a predefined number of iterations (also known as generations). The CGP algorithm written in pseudocode is included in Algorithm 1.

Problem Formulation

Various approximation strategies have been developed in the context of the evolutionary-based approximate computing [95]. The majority of the currently available methods are error-oriented in the sense that all logic optimizations leading to an approximate solution

Algorithm 1 Cartesian Genetic Programming pseudocode.

Input: CGP configuration parameters, fitness function, initial solution p, termination criterion.

Output: The highest scoring solution p found.

```
1: while <termination condition is not satisfied> do
2:     P ← {p} ∪ {λ offspring of p created by mutation}
3:     evaluatePopulation(P)
4:     b ← selectHighestScoringCandidate(P)
5:     if fitness(b) ≤ fitness(p) then
6:         p ← b
7: return The best candidate found p.
```

are constrained by a predefined *error criterion*. It means that the error determining the quality of candidate approximate circuits is used as a design constraint. Only the non-functional circuit parameters such as power consumption, delay, or area on the chip are subject to the optimization and are considered in the cost function. Depending on the application, the error can be expressed by various metrics such as the error magnitude, the average error magnitude, error probability, or Hamming distance [95].

The majority of the work in the literature formulates the design of approximate circuits as a single-objective optimization problem despite its multi-objective nature [87]. The multi-objective CGP is a very promising approach due to its ability to identify non-dominated solutions in each CGP generation and the possibility to provide many useful compromise solutions [119]. In practice, however, we are typically interested in several (predefined) design targets only; for example, approximate implementations have to be developed for a few error levels known in advance. Then it is usually computationally less expensive to execute a single objective CGP optimizing a given parameter several times and having the remaining ones as the constraints [95].

Given a candidate circuit C, the fitness function $f(C)$ of the single-objective error-oriented method is defined as follows:

$$f(C) = \begin{cases} \text{cost}(C) & \text{if error}(G, C) \leq \mathcal{T}, \\ \infty & \text{otherwise.} \end{cases} \tag{2.1}$$

Here, cost(C) denotes the cost of the candidate approximate circuit C in terms of electrical properties that are of interest in a particular context (for example, in our experiments, we use the area on a chip) and error(C) is the value of an error metric that reflects the quality of the approximation with respect to the original accurate implementation G. The values of cost and error have to be evaluated in each CGP generation for every new candidate solution. Since CGP only performs slight changes in the candidate solutions in each generation, the algorithm needs a high number of generations to converge to solutions with significant resource savings [118]. It is therefore critical for the performance of the approximation process that the error and cost computation is as efficient as possible. The notions of cost and error of the candidate circuit will be described in greater detail in the following sections.

2.3 Non-functional Circuit Metrics

The goal of approximate circuit design is to introduce inaccuracies into the circuit's functionality to improve the non-functional characteristics of the circuit. These characteristics often include non-functional metrics such as circuit area, delay, or power consumption. The non-functional metrics depend heavily on the target technology chosen to implement the circuit – the delay and power consumption of the same functionality implemented in FPGAs, 500 nm CMOS (complementary metal-oxide semiconductors) and 45 nm CMOS technologies will vary enormously. The target fabrication technology is chosen by the circuit manufacturer based on multiple criteria such as price, availability, required performance, etc.

Given a circuit description (e.g. a netlist), current hardware synthesis tools (also electronic design automation – EDA tools) can map the description to an implementation in the chosen target technology. Each target technology usually describes a library of basic components (logic gates, half adders, full adders, etc.) from which EDA tools build the circuits. Moreover, the synthesis tools are also able to provide multiple implementations, often with various trade-offs between the circuit delay and power consumption. These variants can be suitable for different target applications – high performance (low delay) circuits are desired in real-time machine learning systems [46], while the power efficient variants aim at embedded and mobile devices to provide longer battery life. To identify the overall most efficient implementations, various compound metrics have been proposed, such as power-delay product (PDP), area-delay product (ADP), and energy-delay product (EDP).

The automated approaches for circuit approximation mentioned in the previous section have to determine the non-functional metrics of the proposed approximate solutions in order to identify the best solution. As these approaches (such as SASIMI or CGP) often work iteratively, the non-functional metrics have to be evaluated in each iteration of the approximation process. The circuit area and delay can be accurately estimated using EDA tools such as Synopsys DC and PrimeTime-PX. However, the estimation process using EDA tools is computationally intensive and time consuming. Utilisation of EDA tools in each iteration would slow down the iterative approximation techniques, significantly reducing the explored search space and therefore the quality of resulting approximate solutions.

To avoid this problem, the iterative approaches make use of less accurate, but much faster estimations of the solution's non-functional parameters. The simplest estimation of the circuit area is to count the number of components the circuit contains (i.e. number of gates in case of gate level netlist transformation). This approach relies on the assumption, that if the netlist of the approximate solution contains less gates than the original circuit, its representation in the target technology will also be smaller. The estimation can be further refined by selecting a specific target technology and taking into account its library of components. The library usually specifies the size and delay of its components and these parameters can differ wildly among the components – a NOT gate can be many times smaller than a XOR gate. We can therefore estimate the size of an approximate circuit's netlist as the sum of the sizes of its gates in the target technology.

Analogously to the area estimation, we can predict the delay of an approximate circuit by counting the number of gates along its critical path (longest path from the inputs to the outputs). Again, this estimation can be further improved by considering the component delay in a given target technology. More advanced estimation methods include for example the simulation of the circuit's netlist and counting the switching activity of its gates to predict the circuit's power consumption [79].

The estimation methods for determination of the non-functional parameters are less accurate because they do not take into account the various optimisations and transformations performed by the hardware synthesis tools. However, they allow us to significantly speed up the approximation process.

2.4 Error Metrics

Accurately assessing the error of an approximate solution is one of the key steps of approximate logic synthesis. Moreover, different target applications of approximate circuits require different approximate behaviour. Therefore, various metrics describing the error of approximate circuits have been proposed and shown suitable for different application domains. The error metrics can be divided into two categories: (1) general error metrics, such as Hamming Distance or error rate, and (2) arithmetic error metrics, such as mean absolute error or worst case absolute error. The general error metrics are more suitable for general logic approximation, while arithmetic error metrics are mainly utilised in the approximation of arithmetic circuits. In this section, we will summarise the various error metrics employed in the literature to design approximate circuits.

The first step of the evaluation of the error of an approximate solution with regards to the correctly working solution is to evaluate the error for a single input combination. For a correct circuit G, further denoted as the *golden circuit*, which computes a function f_G, and its approximation C, computing a function f_C, where $f_G, f_C : \{0,1\}^n \rightarrow \{0,1\}^m$, we compute the error e_x for an input combination $x \in \{0,1\}^n$ as follows:

$$e_x = ||f_G(x) - f_C(x)|| \tag{2.2}$$

where the minus sign represents the difference between the outputs of the two functions. From this formula, we can derive an equation for input independent error metric E, which is computed from the values of e_x across all possible inputs x.

2.4.1 General Error Metrics

The metrics belonging to the general category view the inputs and outputs of the golden circuit and the approximate circuit as sequences of bits and do not give them any further interpretation. These metrics also usually treat all the output bits as having the same importance (also sometimes called weight).

The error rate (ER, also known as error probability) is a basic error metric that measures the ratio of input vectors for which the output of the approximate circuit is inaccurate (the approximate output differs from the correct output). This metric does not take into account the number of different bits – outputs differing in one bit or multiple bits all contribute the same to the error value. Error rate can be computed as follows:

$$\mathrm{ER}(C,G) = 2^{-n} \sum_{x \in \{0,1\}^n} \begin{cases} 1 & \text{if } f_C(x) \neq f_G(x), \\ 0 & \text{otherwise.} \end{cases} \tag{2.3}$$

In many cases, a more suitable metric is the Hamming Distance (HD). This error metric counts the number of flipped bits in the corresponding approximate and exact outputs. We can measure either the worst case Hamming Distance (HD_{wc}) which represents the maximum number of flipped bits for a single input combination:

$$\mathrm{HD}_{wc}(C,G) = max_{x\in\{0,1\}^n} \sum_{i=1}^{m} (f_C(x) \oplus f_G(x))_i, \tag{2.4}$$

or the average Hamming Distance (HD_{avg}), which gives us the average number of inaccurate bits in the outputs across all input combinations:

$$\mathrm{HD}_{avg}(C,G) = \frac{\sum_{x\in\{0,1\}^n} \sum_{i=1}^{m} (f_C(x) \oplus f_G(x))_i}{2^n} \tag{2.5}$$

The average Hamming Distance was utilised to design approximate circuits by the authors of [125, 42].

2.4.2 Arithmetic Error Metrics

The outputs of arithmetic circuits (adders, multipliers, etc.) are interpreted as signed or unsigned integers. Therefore, when approximating such circuits, the general error metrics are not sufficient to accurately express the difference in the behaviour of the golden and approximate solutions. For example, an output with a single bit flip in the most significant bit has a Hamming Distance of 1 from the correct output, but when interpreted as unsigned integers, the difference is 50 % of the output range.

Methods for approximation of arithmetic circuits often utilise arithmetic error metrics to obtain high-quality approximate solutions. There exist several arithmetic error metrics characterising different types of errors such as the worst-case error or the mean error. The worst-case error is essential when guarantees on the worst behaviour of the approximate circuits are required. On the other hand, the mean error better describes the overall error behaviour of the approximate solution. Since the values of some of these metrics can be very large for circuits with wide outputs, the error values are sometimes relativized by the range of the output (e.g. divided by 2^{32} for a 16-bit multiplier).

The worst-case behaviour is typically captured either by the *worst-case absolute error* (WCAE) or by the *worst-case relative error* (WCRE). The worst-case absolute error measures the maximum difference between the outputs of the candidate and the golden solution across all input combinations. We define its relativized variant as follows:

$$\mathrm{WCAE}(C,G) = \frac{\max_{x\in\{0,1\}^n} |\mathrm{int}(f_G(x)) - \mathrm{int}(f_C(x))|}{2^m} \tag{2.6}$$

where $\mathrm{int}(x)$ denotes the integer representation of a bit vector x and $|i|$ denotes the absolute value of an integer i. With the above relativized metrics definition, we can measure the error value in per cent relative to the function's output range.

The worst-case relative error is similar to WCAE, but relates the error value for a given input combination by the magnitude of the expected (correct) output. The definitions of WCRE slightly differ in the literature – depending on the behaviour in the cases where the expected output is equal to 0. To avoid division by zero, such cases can be either ignored, or the denominator is set to 1. We define WCRE as follows:

$$\mathrm{WCRE}(C,G) = \max_{x\in\{0,1\}^n} \frac{|int(f_G(x)) - int(f_C(x))|}{max(int(f_G(x)), 1)} \tag{2.7}$$

The mean absolute error (MAE) is an average based metric similar to WCAE. Unlike WCAE, MAE averages the values of arithmetic error across all input combinations. It

therefore captures the overall error behaviour of the approximate circuit.

$$\mathrm{MAE}(C,G) = \frac{\sum_{x\in\{0,1\}^n} |\mathrm{int}(f_G(x)) - \mathrm{int}(f_C(x))|}{2^n} \tag{2.8}$$

The mean squared error (MSE) is similar to MAE, but squares the computed error value for each input combination. The squaring of the error values results in weighing large error values more than small ones (i.e. MSE prefers solutions featuring small errors for multiple input combinations to solutions with large errors for a few inputs).

$$\mathrm{MSE}(C,G) = \frac{\sum_{x\in\{0,1\}^n} (\mathrm{int}(f_G(x)) - \mathrm{int}(f_C(x)))^2}{2^n} \tag{2.9}$$

Similarly to the worst case metrics, where WCAE has its relative counterpart WCRE, we can use the relative error distance to modify the MAE equation to define the mean relative error (MRE). As with WCRE, we use 1 in the denominator for the input combinations where $f_G(x) = 0$.

$$\mathrm{MRE}(G,C) = 2^{-n} \sum_{x\in\{0,1\}^n} \frac{|int(f_G(x)) - (f_C(x))|}{max(int(f_G(x)), 1)} \tag{2.10}$$

In addition to the above mentioned metrics that measure the error magnitude, we can also examine other properties of the approximate solution's error distribution. One of them is the mean of the error distribution. The mean tells us, whether the approximate solution tends to under or overapproximate the correct result. When the error distribution mean is close to 0, the error introduced over multiple approximate computation steps has the potential to average out. For example, truncated multipliers always underapproximate the correct result and therefore their error distribution is heavily biased.

$$\mathrm{Avg}(C,G) = \frac{\sum_{x\in\{0,1\}^n} (\mathrm{int}(f_G(x)) - \mathrm{int}(f_C(x)))}{2^n} \tag{2.11}$$

Another property important especially for approximate multipliers is the ability to produce correct results when one of the input operands is 0. Approximate multipliers with this quality have been shown to provide superior performance in applications such as neural networks [78], where the majority of approximate operations is multiplication by 0, or image processing filters.

$$\mathrm{Acc0}(G,C) = \begin{cases} 1 & \text{if } \forall x \in \{0,1\}^n : f_G(x) = 0 \Rightarrow f_C(x) = 0 \\ 0 & \text{otherwise.} \end{cases} \tag{2.12}$$

2.5 Error Metrics Evaluation

The success of approximate design methods depends on many aspects. Among others, the efficiency and accuracy of the procedure evaluating the quality of candidate approximate circuits generated by a chosen heuristic procedure has a substantial impact on the overall efficiency. The quality of approximate circuits is typically expressed using one or several error metrics, which were introduced in the previous section. The error metrics formulas compute the error values by enumerating all possible input combinations. The number of input combinations grows exponentially with the number of the circuit's input bits. Therefore, the error evaluation slows down considerably as we move from simple to more complex

circuits. For large input bit widths, the enumeration of all possible input combinations is computationally infeasible. To mitigate this problem, two main groups of approaches have been utilised in the literature: (1) estimating the error value using a subset of input combinations or probabilistic error analysis and (2) using formal verification methods to evaluate or place bounds on the approximation error. These approaches will be further introduced in this section.

2.5.1 Simulation

Conceptually, the simplest approach to obtain precise error bounds of an approximate arithmetic circuit (AAC) is to simulate its function on all possible inputs. The time needed for such an evaluation can be significantly lowered on modern processors where n inputs can be evaluated in parallel (with n being the width of the word of the processor). Due to that, the approach has been successfully employed in the synthesis of small and mid-size AACs [121, 77]. However, even on state-of-the-art computer architectures, this approach has principal scalability limitations causing that it cannot be used to synthesise approximate circuits with more than 12-bit operands [78].

The search-based synthesis is, in general, computationally expensive (hundreds of thousands of iterations are typically evaluated). Hence, the error evaluation needs to be fast as it has a great impact on the scalability of the whole design process. In order to maintain reasonable scalability and avoid a computationally expensive exhaustive simulation, many authors simplify the problem and evaluate the quality of the approximate circuits by applying a subset of all possible input vectors. Monte Carlo simulation is typically utilised to measure the error of the output vectors with respect to the original solution [124, 80, 49]. Unfortunately, a small fraction of the total number of all possible inputs vectors is typically used. For example, 10^3 vectors were used to evaluate a perceptron classifier and less than 10^4 vectors were employed for a 16x16 block matcher in [80]; 10^8 vectors were used to evaluate 16-bit adders in [49].

It is clear that this approach cannot provide any guarantee on the error and makes it difficult to predict the behaviour of the approximate circuit under different conditions. Not only the obtained error value strongly depends on the chosen vectors but this approach may also lead to overfitting. Alternatively, the circuit error can be calculated using a statistical model constructed for elementary circuit components and their compositions [61, 68]. However, reliable and general statistical models can only be constructed in some specific situations.

2.5.2 Formal Methods for Error Evaluation

While formal methods of (exact) equivalence checking have been studied for decades, only a few formal approximate checking methods have been used in circuit approximation tools. Depending on the particular error metric (e.g., the mean error or the worst-case error), the error calculation is transformed to a decision problem and solved by means of SAT solving or binary decision diagrams (BDDs).

Recently, various applications of formal methods have been intensively studied in order to improve the scalability of the design process of correct as well as approximate circuits. For designing correct circuits (where one insists on preserving the original functionality but tries to optimize non-functional parameters), one can consider combinational equivalence checking based on modern SAT solvers, efficient BDD representations of circuits, or algebraic computation techniques combining polynomial representation of circuits with

a) truth table

a	b	c	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

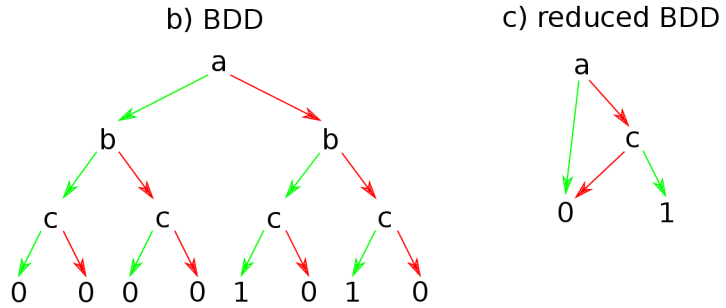


Figure 2.10: An example Boolean function represented by a) a truth table, b) a BDD and c) a reduced BDD. The green / red arrows visualise connections between parent and low / high successors, respectively. The internal nodes a, b and c represent Boolean variables.

logic reductions [24, 92]. For designing AACs, a more challenging notion of *relaxed* or *approximate equivalence checking* is needed. This notion requires the formal verification technique to quantify the approximation error or, alternatively, prove whether the error is below a certain threshold.

BDD-based Evaluation

Determining whether two Boolean functions are functionally equivalent is a classic problem solved by formal verification techniques. Many of the current formal verification approaches are based on the Binary Decision Diagrams [10] (BDDs). BDDs represent a Boolean function as a rooted, directed, connected, acyclic graph, which consists of internal Boolean decision nodes and Boolean result leaf (terminal) nodes. Each decision node n is labelled by a Boolean variable x and has two successors: the so called low successor l and high successor h. The edge from n to high successor h represents the assignment of Boolean 1 to variable x. Similarly, the edge from n to l represents the assignment of Boolean 0 to x. The terminal nodes are labelled with Boolean values 0 and 1. A path from the root node to a terminal node describes a variable assignment for which the represented Boolean function has value corresponding to the label of the terminal node. An example Boolean function represented by a truth table and a BDD is illustrated in Fig. 2.10.

A generic BDD can be transformed into ordered BDD (OBDD) by enforcing an ordering on the BDD's Boolean variables along all paths from the root to the leaves. Additionally, a BDD is called reduced BDD (RBDD) if there are no two nodes with identical successors and no two non-identical isomorphic subtrees in the BDD. A reduced ordered BDD (ROBDD) is a canonical representation of a Boolean function (for a given ordering, the ROBDD representing the Boolean function is unique).

The BDD representation provides effective ways to perform Boolean operations on Boolean functions, checking properties of Boolean functions (satisfiability, validity, etc.), and checking equivalence of Boolean functions. A significant advantage of BDDs is, that the operations on BDDs can be done without decompressing the represented functions. The operations on Boolean functions are performed via the *apply(op, f, g)* function. The function takes two ROBDDs f and g and a binary operator *op* and returns a new ROBDD representing the function f *op* g. Another operation on ROBDDs, *SAT-count*, computes the number of input assignments, for which the Boolean function results in logical 1. The

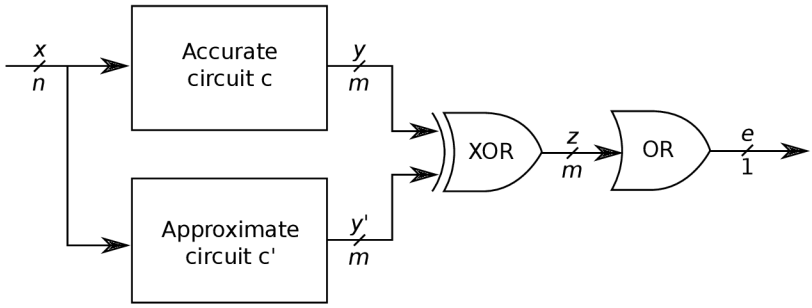


Figure 2.11: The schema of a miter for checking exact equivalence. The two circuits c and c' are functionally equivalent if and only if the output e has value 0 for all possible inputs x.

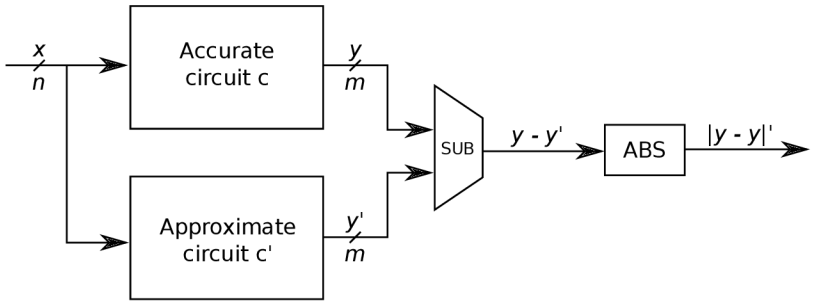


Figure 2.12: The schema of a miter for checking the arithmetic error of the approximate circuit. The SUB circuit is a subtractor that interprets the outputs y and y' as integer values and computes their difference.

SAT-count of an ROBDD can be computed in linear time with respect to the size of the diagram, which is a very important characteristic for circuit design.

BDDs have been traditionally used for exact equivalence checking during hardware synthesis and verification. Given two combinational circuits c and c' with m inputs and n outputs, we name the inputs of both circuits $x_1, ..., x_n$, and the outputs $y_1, ..., y_m$ and $y'_1, ..., y'_m$, respectively. To determine whether c and c' are functionally equivalent, we construct an auxiliary circuit called *miter*. The miter consist of the two examined circuits, has inputs $x_1, ..., x_n$ and m outputs z computed as $z_i = y_i \oplus y'_i$. Each of the outputs z_i can be represented by an ROBDD. The question of circuit equivalence is transformed to checking the SAT-count of the ROBDDs. If the ROBDD for z_i has SAT-count greater than 0, it means that there exists an assignment for which the output bits y_i and y'_i are different and therefore c and c' are not equivalent. One the other hand, c and c' are functionally equivalent, if and only if all of the ROBDDs have no satisfying assignments. Additionally, the BDDs for z_i can be combined together using the OR operation to create a single BDD representing the whole miter. The structure of such miter is visualised in Fig. 2.11.

The approach from strict equivalence checking on BDDs can be expanded to approximate equivalence checking to effectively evaluate the approximation error of an approximate circuit. The authors of [120] propose an algorithm to compute the Hamming Distance of an approximate circuit by summing the SAT-count values for each of the z_i ROBDDs. In a similar manner, the authors of [104] build on the concepts of approximation miter to design algorithms to determine the worst case and average error of an approximate circuit using ROBDDs. An example of the approximate miter computing the arithmetic error between

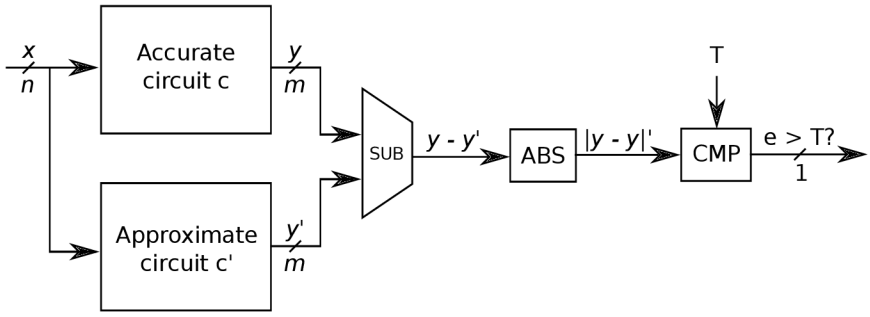


Figure 2.13: The schema of a miter that checks whether the worst case error of the approximate circuit c' exceeds a given error threshold value T. Parameter T is provided as an integer value and the CMP component compares it to the absolute value of the error.

the accurate and approximate circuit is shown in Fig. 2.12. The miter utilises a subtractor circuit to compute the difference between y and y'. An absolute value circuit then computes the absolute value of the difference. Such a miter allows BDD based verification techniques to evaluate both the worst-case and average-case error of the approximate circuit. An efficient BDD-based approach allowing one to guarantee the worst-case and the average-case arithmetic error of approximate adders up to 16-bit operands was proposed in [114]. An alternative approach that uses BDDs representing characteristic functions was employed in [21].

The canonical nature of ROBDDs makes them inefficient when representing certain classes of functions. Their efficiency is also very sensitive to the chosen variable ordering [27] – the size of the BDD representation of a function can vary dramatically based on the used ordering. Since finding an optimal variable ordering is an NP-hard problem, various heuristics are used to find high-quality orderings. Nevertheless, there exist certain functions for which the number of nodes is always exponential, independent of the chosen variable ordering, the most prominent one being the multiplication function [110]. Other examples of such functions include integer division, remainder and square root.

SAT-based Evaluation

Since BDDs have been shown to be inefficient in representing multiplication and some other functions, hardware designers utilise SAT solvers to verify the circuits representing these functions. There exist many powerful SAT solving tools able to prove the unsatisfiability of large Boolean formulae. The problem of circuit equivalence is translated to deciding whether a given Boolean formula is satisfiable or not.

In the strict equivalence checking scenario, a SAT solver is utilised to prove the functional equivalence of two circuits using the miter construction from Fig. 2.11. The miter circuit has a single output and can therefore be converted to a corresponding Boolean logic formula, usually in Conjunctive Normal Form. For a given input assignment, the evaluation of the formula corresponds to the value of the output bit. The two circuits composing the miter are functionally equivalent, if and only if the corresponding Boolean formula is unsatisfiable, i.e., there is no input assignment for which the outputs of the circuits differ. The answer is obtained by solving the formula using a SAT solver.

The advances in SAT solver performance led researchers to utilise them in approximate circuit design as well. While SAT solvers are able to handle larger approximate circuit instances than BDDs, they can be used only when a binary output is sufficient. This is

typically done for the worst-case error where one can ask whether the produced error is under a bound (error threshold) given by the designer as a parameter [78]. To answer this question, the approximation miters introduced earlier have to be modified to feature a single output. This is achieved by the inclusion of a comparator block, which compares the computed arithmetic error with the given error threshold value. An example of such miter circuit is shown in Fig. 2.13.

The exact value of a circuit's worst-case error can be evaluated by performing a binary search of the circuit's output range, i.e. running multiple SAT evaluations with various threshold values. In its basic form, the SAT based approach is unable to determine error metrics that aggregate the error value across all input combinations, such as mean average error or error rate.

The number of inputs for which an approximate circuit returns an incorrect result can be quantified using *SAT counting methods* (so-called #SAT solvers), that count the number of satisfying assignments of a given formula. The #SAT solving approach allows one to compute the error rate of a circuit using the miter construction from Fig. 2.11. However, despite the recent progress in the area of #SAT solvers (see, e.g., [19]), the #SAT problems encoding the error quantification are currently beyond the capabilities of state-of-the-art #SAT tools for more than small circuit instances.

Algebraic Polynomial Representation

An alternative method to arithmetic circuit verification was proposed in [36]. The authors design an approach to detect and diagnose errors in logic synthesis process by modelling the circuit using symbolic computer algebra. The method is based on representing the signals of the circuit in an algebraic polynomial representation and then computing differences between the polynomials. A non-zero resulting difference indicates a functional difference in the behaviour of the circuits.

This approach was extended in [34] to relax the strict requirements on the functional equivalence. The modified method can be used during an approximation process to determine, whether an approximate solution satisfies some bound on the approximation error. The computation of the error metric value is based on Gröbner Basis reduction. The golden circuit and the approximate circuit are represented as symbolic algebra polynomials. The specification polynomial is then divided by the approximate circuit polynomial, giving us the remainder R. When $R = 0$, the two polynomials are functionally equivalent. Otherwise, R represents the difference in behaviour between the polynomials and can be used to determine the approximation error. The authors of this approach utilise SMT (satisfiability modulo theories) solvers to compute the worst case error from R or solve a pseudo-Boolean formula constructed from R to quantify the mean squared error.

Chapter 3

Scalable SAT-based Approximate Equivalence Checking

This chapter is based on our work published in [13, 18]. In this part of the research, we try to improve the speed and scalability of the error evaluation phase of the approximation process. We focus mainly on the *worst-case absolute error* (WCAE) and *worst-case relative error* (WCRE) metrics and their evaluation using SAT solving techniques. To achieve better scalability, we propose novel approximation miter constructions as well as a new modification of the search algorithm called the *verifiability driven search strategy*. The chapter is structured as follows. Firstly, we introduce the concept of the verifiability driven search. Afterwards, we describe the improved miter construction for WCAE evaluation and show its performance in a detailed experimental evaluation. Finally, we extend the idea of the WCAE miter to evaluate the WCRE metric and follow with another experimental evaluation.

3.1 Motivation

In this chapter, we present a new method for designing complex approximate arithmetic circuits with formal bounds on the approximation error. The method uniquely integrates new formal techniques for approximate equivalence checking into search-based circuit optimisation by means of Cartesian Genetic Programming (CGP). The key idea is to employ a novel search strategy driving the search towards promptly verifiable approximate circuits. We have implemented the strategy within the ABC tool (see Chapter 5) and extended the underlying equivalence checking algorithm to support queries on the worst-case error. This extension builds on a new effective construction of miters, i.e. auxiliary circuits interconnecting the original correct circuit and its approximation such that their approximate equivalence can be checked.

In the first part of this chapter, we decided to optimise for the WCAE metric since its exact value can be important in time-critical and dependable systems (e.g., inverse kinematics in robot control [39]) or when complex approximate arithmetic circuits are constructed using less complex approximate building (circuit) blocks. The final error then depends on how the worst case error is propagated from low-level blocks to the result. Moreover, even in not so critical applications such as image processing, low average error but excessive worst-case error can produce unacceptable results [56]. Finally, our results suggest that there is also a high correlation between WCAE error and the mean absolute

error (MAE, Section 3.4). In the second part of this chapter, we extend the proposed approach to optimise the WCRE of approximate circuits.

While our primary motivation is to automatically approximate complex multipliers, our method is directly applicable to other arithmetic circuits too. The method is capable of providing Pareto fronts showing high-quality compromises between the circuit error and non-functional circuit parameters. Results are presented for approximate multipliers (with up to 32-bit operands) and adders (with up to 128-bit operands) and compared with several approximate circuits available in the literature. This is for the first time when such complex approximate arithmetic circuits with formally guaranteed error bounds have been presented.

Contributions The contributions presented in this chapter can be summarised as follows:

- We propose new WCAE and WCRE miter constructions allowing for efficient approximate equivalence checking tailored for search-based approximation of complex arithmetic circuits.
- We design a novel search strategy for synthesis of approximate circuits with formal error guarantees that integrates Cartesian Genetic Programming and the proposed approximate equivalence checking. Using a resource-limited verifier, the strategy drives the search towards promptly verifiable candidates and thus provides scalable approximation of complex circuits.
- We develop an implementation of the miter construction and the search strategy within the ABC tool and perform extensive experimental evaluation of our approach on large circuits including the approximation of 128-bit adders and 32-bit multipliers.
- Within several hours, we are able to construct high-quality Pareto sets of 128-bit adders and 32-bit multipliers with WCAE guarantees that represent the trade-offs between the circuit error and non-functional circuit parameters.

3.2 Search-Based Design of AACs

In this section, we present our novel approach to the search-based design of AACs combining the principles of CGP with the verifiability-driven search strategy that employs a fitness function based on the approximate equivalence checking.

3.2.1 Problem Formulation

First, we formalise the problem of designing approximate arithmetic circuits as a *single-objective optimisation problem*. Recall that the aim of the circuit approximation process is to improve the non-functional characteristics (such as the chip area, energy consumption, or delay) of the given circuit by introducing an error in the underlying computation.

The non-functional characteristics of the circuit depend on the target technology the circuit is synthesised for. Computing these characteristics precisely for every candidate solution would introduce a significant computation burden for the approximation process. Therefore, we approximate these characteristics by an estimated size of the circuit computed as follows. We assume that we are given a list of gates that can be used in the circuit and that each gate is associated with a constant characterising its size. The size of the particular gate is specified by the users and should respect the target technology (see Table 3.1 for the gates and their sizes used in our numerical trials). For a candidate circuit C, we then

define its size, denoted size(C), as the sum of the sizes of the gates used in C. As shown in [78, 113, 114], size(C) typically provides a good estimate for the chip area as well as for the power consumption.

The problem of finding the best trade-offs between the circuit size and the approximation error can be naturally seen as a multi-objective optimisation problem. In our approach, we, however, treat it as a series of single-objective problems where we fix the required values of the approximation error. This approach is motivated by the fact that the magnitude of the approximation error is usually given by the concrete application where the approximate circuits are deployed. Moreover, as shown in the study [118], optimising the chip size for a fixed error allows one to achieve significantly better performance compared to more general multi-objective optimisation producing Pareto fronts. The performance directly affects the time required to find high-quality approximation and is essential to scale to complex circuits such as 16-bit multipliers and beyond.

The key optimisation problem we consider is formalised as follows:

Problem: *For a given golden circuit G, error metric Error and an error threshold $\mathcal{T}$, our goal is to find a circuit C^* with the minimal size such that the Error$(G, C^*) \leq \mathcal{T}$.*

For the functionality given by the golden circuit G, the optimisation attempts to find the minimum total size of active gates and the corresponding connections, represented by the approximate circuit C^*, that guarantee the desired level of accuracy $\mathcal{T}$.

We emphasise that our aim is not to provide a complete algorithm that guarantees the optimality of C^*: such an algorithm clearly exists as the number of circuits with a given size is finite, and one can, in theory, enumerate them one by one. We rather design an effective search strategy that is able to provide high-quality approximations for complex arithmetic circuits having thousands of gates in the order of hours.

3.2.2 Cartesian Genetic Programming

CGP is a form of genetic programming where the candidate solutions are represented as a string of integers of a fixed length that is mapped to a directed acyclic graph [71]. This integer representation is called a *chromosome*. CGP can efficiently represent common computational structures including mathematical equations, computer programs, neural networks, and digital circuits. The candidate solutions are typically represented in a two-dimensional array of programmable two-input nodes.

In circuit approximation, the evolution loop starts with a *parent* representing a correctly working circuit. New candidate circuits are obtained from the parent using a *mutation operator* which performs random changes in the candidate's chromosome in order to obtain a new, possibly better candidate solution. In the next step, the algorithm evaluates the quality of each solution using a specified metric, called the *fitness function*. This function assesses important correctness and performance aspects of circuits. The candidate with the best fitness value is chosen as the parent of the next generation, the other solutions are removed and the evolution continues the next iteration by creating a new generation of candidate circuits. The whole loop is repeated until a termination criterion is met. For more details of CGP, refer to Section 2.2.5 or see [71].

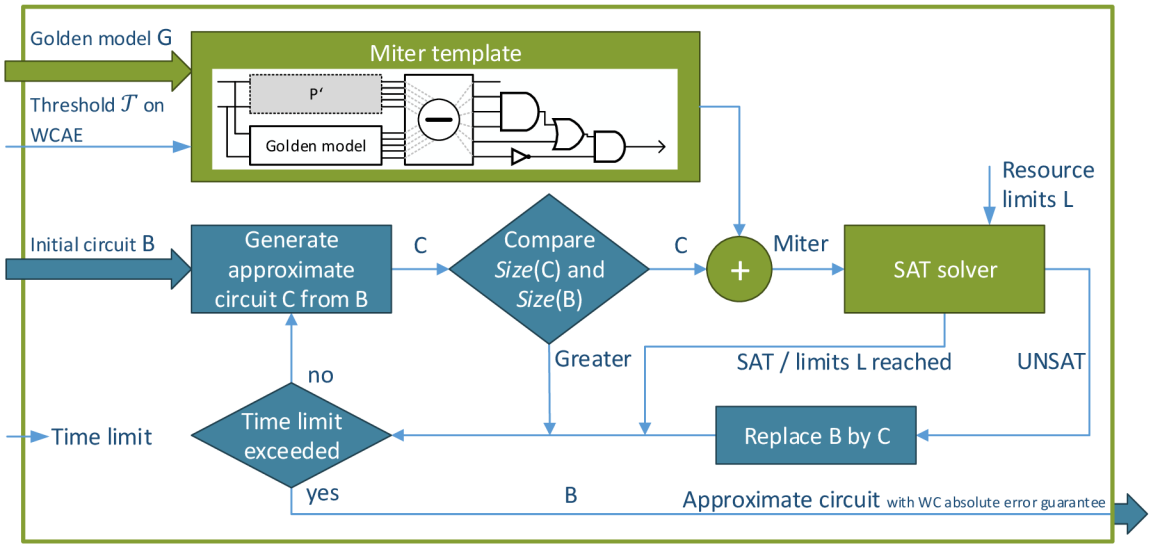


Figure 3.1: The main steps of the proposed verifiability-driven search scheme using CGP.

3.2.3 Verifiability-Driven Search Strategy

Our verifiability-driven search strategy can be viewed as a general concept improving the scalability of evolutionary design techniques. We presume the utilisation of SAT-based approximate equivalence checking to evaluate the approximation error in the evolutionary process. The problem formulation from Section 3.2.1 allows us to define the fitness function f in the following way:

$$f(C) = \begin{cases} \text{size}(C) & \text{if Error}(G, C) \leq \mathcal{T}, \\ \infty & \text{otherwise} \end{cases}$$

where size(C) denotes the size of the circuit C. Since the procedure deciding whether Error$(G, C) \leq \mathcal{T}$ (further denoted as SAT solver) represents the most time consuming part of the design loop, we avoid calling the procedure as much as possible. Therefore, we only call SAT solver for circuits C satisfying size$(C) \leq$ size(B) where B is the best solution with an acceptable error (i.e., Error$(G, B) \leq \mathcal{T}$) that we have found so far. Our experiments show that, during the evolution process, a significant set of candidate designs C does not satisfy the condition size$(C) \leq$ size(B) and thus their fitness can be easily assessed without calling the SAT solver.

Our experiments further indicate that a long sequence of candidate circuits B_i improving the size and having an acceptable error has to be typically explored to obtain a solution that is sufficiently close to C^*. Therefore, both the SAT and the UNSAT queries to SAT solver have to be as brief as possible. To this end, we use an additional criterion for the evaluation of the circuit C, namely, the ability of SAT solver to prove that Error$(G, C) \leq \mathcal{T}$ with a given limit L on the resources available to the underlying decision procedure. If the procedure fails to prove Error$(G, C) \leq \mathcal{T}$ within the limit L, we set $f(C) = \infty$ and generate a new candidate. The overall design loop using the verifiability-driven search strategy is illustrated in Fig. 3.1.

The inputs of the design process include: (1) the golden model G, (2) the threshold on the approximation error $\mathcal{T}$, (3) the initial circuit B having an acceptable error (it can be either the golden model or a suitable approximation we want to start with), and (4) the time

limit on the overall design process. The loop exploits the CGP principles; namely, it uses mutations to generate new candidate circuits C from the candidate circuit B representing the best approximation of the circuit C^* that we have found so far. The circuit C is then evaluated using the fitness function f as described above. If the candidate C belongs to the improving sequence (i.e., $\text{size}(C) \leq \text{size}(B)$ and $\text{Error}(G, C) \leq \mathcal{T}$), we replace B by C. The design loop terminates if the time limit is reached and B is returned as the output of the design process.

In our verifiability-driven search scheme, we use the resource limit L (as a parameter of the design loop) to drive the search towards candidates that can be promptly evaluated. We intentionally throw away improving candidates B_i that require greater resources and thus longer, but still feasible, verification time. The reason for this is the fact that by mutating these candidates we would most likely obtain solutions that would require the same or even longer verification times and thus finding the whole improving sequence would become time-infeasible. Instead, we require that every improving candidate B_i has to be verifiable using the resource limit L and thus drive the search towards candidates B_i that, for a given time limit on the overall design process, lead to longer improving sequences. Our experiments indicate that these sequences lead to candidate circuits that are closer to C^*. Since we are able to evaluate a much larger set of candidate circuits, we have a better chance to find a long improving sequence within the given time provided that it exists for the limit L.

The obvious disadvantage is that we possibly cut improving sequences that would lead to good solutions within the given design time. It can also happen that, for the limit L, no improving sequence exists, while it exists for a slightly greater resource limit. Despite of this limitation, our results clearly show that the proposed verifiability-driven search strategy allows us to utilise the given design time in a more efficient way compared to the standard evolution schemes.

3.3 SAT-based WCAE Evaluation

Various metrics describing the error of AACs have been proposed and shown suitable for different application domains. The most popular error metrics relevant especially to arithmetic circuits are the *worst-case absolute error* (WCAE) and the *mean absolute error* (MAE). For a correct circuit G, further denoted as the *golden circuit*, which computes a function f_G, and its approximation C, computing a function f_C, where $f_G, f_C : \{0,1\}^n \to \{0,1\}^m$, these metrics, relativized by the range of the output, are defined as follows:

$$\text{WCAE}(G, C) = \frac{\max_{x \in \{0,1\}^n} |\text{int}(f_G(x)) - \text{int}(f_C(x))|}{2^m},$$

$$\text{MAE}(G, C) = \frac{\sum_{x \in \{0,1\}^n} |\text{int}(f_G(x)) - \text{int}(f_C(x))|}{2^n},$$

where $\text{int}(x)$ denotes the integer representation of a bit vector x and $|i|$ denotes the absolute value of an integer i.

3.3.1 Checking Worst Case Absolute Error

To compute whether the WCAE is violated, we can adopt the concept of the approximation miter introduced in [126]. The general configuration of the approximation miter is shown in Fig. 3.2. The miter consists of the inspected approximate circuit C, the golden circuit G which serves as the specification, a subtractor, an absolute value block, and a comparator

which checks whether the error introduced by the approximation is greater than a given threshold $\mathcal{T}$. The output of the miter is a single bit which evaluates to 1 if and only if the error is violated, i.e. $\text{WCAE}(G, C) > \mathcal{T}$.

For a given input vector x, the subtractor calculates the difference between the output of the golden circuit, i.e. $f_G(x)$, and the output of the approximate circuit, i.e. $f_C(x)$. Let $d = \text{int}(f_G(x)) - \text{int}(f_C(x))$ be the error magnitude. A direct computation of the WCAE according to its definition leads to evaluating the expression $e = |d|$, i.e. the absolute difference of the error magnitude, which is performed by the absolute value block. The absolute difference is can be calculated for example by the means of a common two's complement subtractor (implemented using m full-adders with the first carry-in set to 1 and inverting each bit of the subtrahend) followed by a circuit determining the absolute value (computed using m half-adders and m XOR gates).

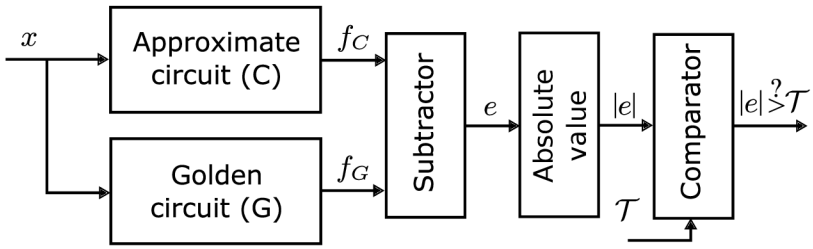


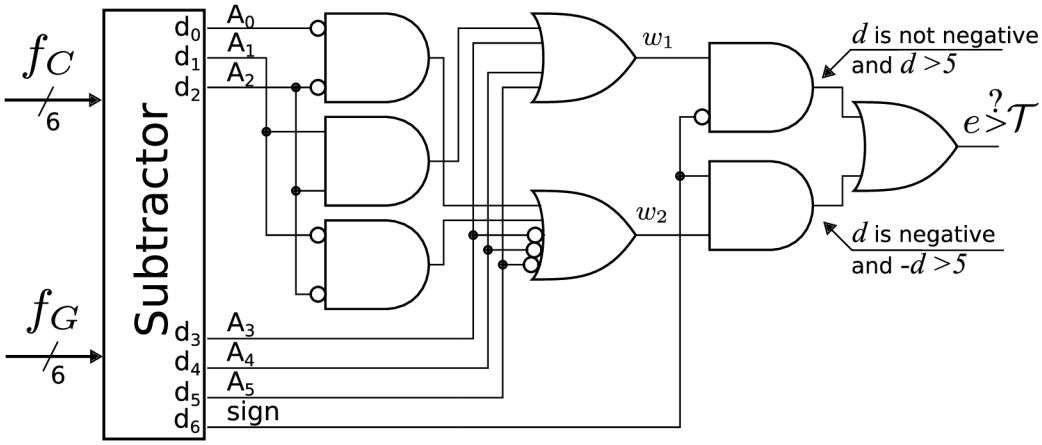
Figure 3.2: Approximation miter for the worst case absolute error analysis. Showing a typical scenario where $e(x) = |f_G(x) - f_C(x)|$.

3.3.2 The Proposed Miter Construction

Miters so far used in the literature compute the absolute value of the difference between f_G and f_C ($|f_G - f_C|$). The computation is usually performed in two steps. Firstly, a subtractor with a signed output evaluates $f_G - f_C$. Secondly, the absolute value has to be computed. The circuit performing such a task contains XOR chains which are a known cause of poor performance of the state-of-the-art SAT solvers [41]. The main reasons are that unlike AND/OR gates, the Boolean constraint propagation over XOR gates is limited, and the XOR operations cause the CNF form of the formulae to grow rapidly.

In order to avoid long XOR chains at the output of the miter which slow down the decision process, we propose to employ a different approach. The key idea is to compare the result of the subtractor with both the positive and negative value of the threshold and thus avoid the expensive evaluation of the absolute value. For a given threshold $\mathcal{T}$ on the worst-case absolute error WCAE, it holds that $e > \mathcal{T}$ is satisfied if and only if d is positive and $d > \mathcal{T}$, or d is negative and $-d > \mathcal{T}$. As we typically deal with numbers in the two's complement, the second condition is equal to $\neg d > (\mathcal{T} - 1)$. Hence, we can use the two's complement representation and examine the positive and negative values separately to avoid usage of the absolute difference of the output.

Since the threshold $\mathcal{T}$ is fixed during the design process, we can easily avoid the standard comparator consisting of a long chain of XOR gates. This helps us to further simplify the miter and improve the performance of the decision procedure. In particular, we replace the sequential comparison of the particular bits of the operands implemented as



$$
\begin{aligned}
w_1 &= (A_1 \wedge A_2) \vee A_3 \vee A_4 \vee A_5 \\
w_2 &= (\overline{A_0} \wedge \overline{A_2}) \vee (\overline{A_1} \wedge \overline{A_2}) \vee \overline{A_3} \vee \overline{A_4} \vee \overline{A_5} \\
\text{WCAE}(G,C)>5 &\Leftrightarrow (\overline{sign} \wedge w_1) \vee (sign \wedge w_2)
\end{aligned}
$$

Figure 3.3: The proposed approximation miter for the worst-case absolute error analysis: an example for $\mathcal{T} = 5$, $N = 6$.

$$
A > B \equiv \bigvee_{0 \le i \le N-1} \left(A_i \wedge \neg B_i \bigwedge_{i < j \le N-1} \overline{A_j \oplus B_j} \right),
$$

for B being a constant bit vector representing the threshold $\mathcal{T}$, by a simpler procedure implemented as

$$
A > B \equiv \bigvee_{0 \le i \le N-1 \ \wedge \ B_i = 0} \left(A_i \bigwedge_{i < j \le N-1 \ \wedge \ B_j = 1} A_j \right).
$$

As is evident, the resulting formula does not contain any XOR gate. Note that d is represented as an $m+1$ bit number in the two's complement—hence, A corresponds to the N least significant bits of d where $N = m$. The $(m+1)$-th bit is reserved for the sign and employed for determining whether d encodes a positive or negative number. The miter for $\mathcal{T} = 5$, f_C and f_G with 6-bit outputs is illustrated in Fig. 3.3.

The proposed construction, compared to the construction using the absolute value and full comparators, allows us to obtain smaller and structurally less complex miters. Such miters can be efficiently used in the SAT-based approximate equivalence checking procedures, resulting in a significant acceleration of the candidate circuit evaluation. Our experiments show that, in the case of arithmetic circuits having 64 output bits (e.g. 32-bit multipliers), the proposed construction improves the size of the miters (in terms of the number of And-Inverter Graph (AIG) nodes representing the circuit) by about 25–35% depending on the value of $\mathcal{T}$, where $\mathcal{T}$ ranged from 0.0001% to 0.5% of the maximal value at the output (i.e. 2^{64}) in our experiment.

3.4 WCAE Experimental Evaluation

In order to evaluate the proposed verifiability driven search with the novel miter construction, we primarily focused on complex approximate multipliers as they are the most challenging benchmark problems. Since only 8-bit multipliers with guaranteed error bounds were presented in the literature so far, there are no solutions available for a direct comparison in the case of 16-bit and more complex approximate multipliers. Hence, (1) we compare the 16-bit approximate multipliers that we generated using our method with 16-bit multipliers (available in the literature) whose error was determined using simulation, and then (2) we present Pareto fronts (the error and key circuit parameters) for 20-bit, 24-bit, 28-bit, and 32-bit approximate multipliers and up to 128-bit approximate adders to demonstrate the scalability of the proposed method.

3.4.1 Experimental Setup

We implemented our approach, including the miter construction and verifiability-driven evolution, within the ABC tool [7]. Array multipliers and ripple carry adders composed of 2-input gates were employed as the initial (golden) circuits for CGP. The set of the utilised gate functions consists of the common two-input logic gates and the inverter gate.

The performance of CGP for particular application domains can be tuned by various parameters. The most relevant CGP parameters for our case are the following: the number of offspring (λ), the frequency of mutations, and the CGP grid size and the `L-back` parameter (i.e. connectivity in the chromosome). In the experimental evaluation presented in this chapter, we choose the CGP parameters as discussed in the following paragraphs. For a more in-depth discussion of the CGP parameter values, see Section 4.3.2.

The literature shows that, for a fixed number of generated and evaluated candidate solutions, CGP-based circuit optimisation (i.e. when circuits are not evolved from scratch) with a smaller value of λ usually leads to better fitness values than CGP using larger values of λ [112]. To keep the settings simple and also allow for a long chain of improving solutions, we set the number of offspring $\lambda = 1$.

Regarding the mutation frequency, the literature (see, e.g. [78]) observes that the number of mutations performed between each generation should be small. This way, the evolution gradually alters the candidate solutions in small steps. Otherwise, for a high mutation frequency, the function of a new solution is usually heavily altered and the evolutionary algorithm starts to resemble a random search. Given these observations, we configure the CGP to mutate 1 to 5 genes in each generation.

Finally, we set the dimensions of the chromosome gate matrix as $1 \times W$ where W equals the number of gates in the correct circuit (i.e. all the gates of the original circuit are in one row) and use `L-back` $= W$, i.e. we allow the maximum connectivity of the chromosomes. This setting gives the evolutionary algorithm maximal freedom with respect to the candidate solutions that can be created. This fact is desirable in case of area optimisation we aim at [71, 118].

For each target WCAE, we performed 30 independent runs of CGP to obtain statistically significant results. Each CGP run was executed for 2 hours on an Intel Xeon X5670 2.4 GHz processor using a single core. Note that the individual CGP runs are independent and thus we executed them in parallel using a cluster of these processors to accelerate the design process.

For purposes of the fitness evaluation, the circuit size is estimated as the sum of the relative area of the two-input gates used, where the sizes of each gate are taken from the technology library. The sizes for each gate type are shown in Table 3.1. At the end of the evolution, the 5 most fitting circuits for each WCAE were synthesised using the Synopsys Design Compiler (high-effort compiling for a better quality of the results) for a 45 nm technology library in order to obtain the non-functional parameters like the area and power-delay product (PDP). The accurate implementations were created by the means of Verilog $*$ and $+$ operators and synthesised in the same way as approximate circuits.

Table 3.1: The sizes of the gates used in the numerical trials. The sizes are in μm^2 and correspond to the 45nm technology.

Gate	INV	AND	OR	XOR	NAND	NOR	XNOR
Size	1.40	2.34	2.34	4.69	1.87	2.34	4.69

3.4.2 16-bit Approximate Multipliers

Evaluation of the verifiability-driven search In the first experiment, we approximated the golden 16-bit multiplier for 9 target values of WCAE from the set {0.1, 0.2, 0.5, 1, 2, 5, 10, 15 and 20%} and evaluated the performance of the proposed method with three different settings of the resource limit L controlling the maximal number of conflicts for one AIG node during the SAT solving procedure: (1) no limits, i.e., L=∞, (2) L=160K, and (3) L=20K. Note that the limits L=160K and L=20K roughly correspond to the time limit of 120 sec. and 3 sec., respectively, on 16-bit multipliers.

Fig. 3.4 shows that, for WCAE $\geq$ 2%, the resource limit L has a marginal impact on the PDP and area of the final approximations. However, with a decreasing target WCAE, the limit L=20K provides significantly better results. For example, if WCAE=0.1% and L=20K, 22,050 SAT calls were produced and 11% of them were terminated on average because of the termination condition. In the case of L=160K, 856 SAT calls were produced only (15% terminated). The average number of SAT calls (across all target errors) that were forced to terminate is 6.28% (for L=160K) and 8.84% (for L=20K). If L=∞, 170 SAT calls were evaluated for WCAE=0.1% only. Despite the fact that some potentially good candidate circuits are quickly rejected, the aggressive resource limits allowed us to generate and evaluate significantly more candidate circuits and thus to substantially improve the quality of the achieved results. Box plots in Fig. 3.4 also show that independent runs with L=20K lead to circuits having very similar parameters (low inter-quartile distances). Hence, the resource limit L=20K will be used in the following experiments to approximate adders and multipliers of various bit widths.

Note that the parameters of some approximate multipliers shown in Fig. 3.4 are worse than for the accurate multiplier. The reason is that the relative area is the only non-functional circuit parameter optimized by CGP while the PDP and area are computed at the end of the optimization using the Synopsys Design Compiler. We have never observed this discrepancy for the limit L=20K.

Comparison with other multipliers Next, we generated 16-bit approximate multipliers using the setup described in the previous section and compared them with approximate multipliers available in the literature. In order to perform a fair comparison (the error of the published multipliers was originally estimated using random simulation), we modified

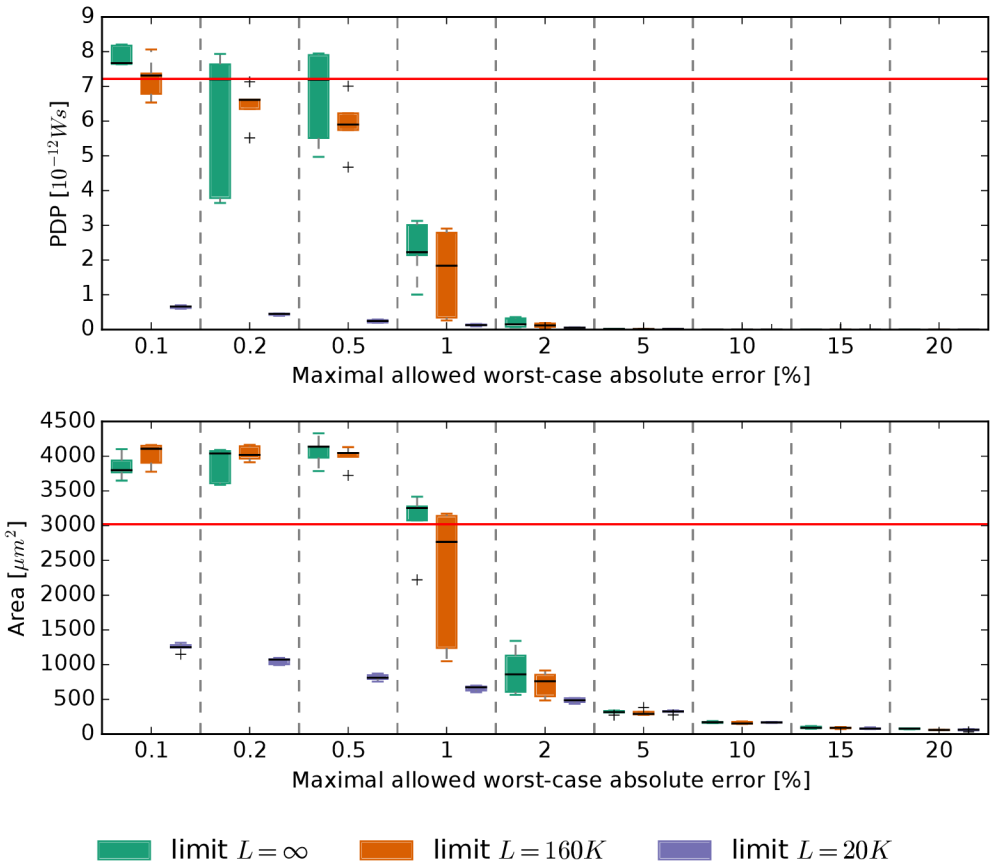


Figure 3.4: PDP and area of approximate 16-bit multipliers for 9 target errors obtained using 3 different resource limits L on the SAT solver. The red line shows the PDP and area of the accurate multiplier.

our method and applied a binary search strategy to determine their WCAE exactly. In addition to WCAE, we also provide MAE obtained using simulation (10^9 vectors).

The following 16-bit approximate multipliers were considered in this study:

M1 Approximate configurable multipliers from the lpACLib library [96]. In this case, the multiplication is recursively simplified using two different variants (denoted as *Lit* and *V1*) of an elementary block representing a 2-bit multiplier. The partial results are summed using accurate adders. We implemented 32 different architectures consisting of four 8-bit multipliers where each of these multipliers is configurable as exact/approximate (2^4 configurations) and can be built using either *Lit* (M1Lit) or *V1* (M1V1) blocks.

M2 The approximate multiplier employing the bit-significance-driven logic compression as introduced in [84].

M3 Approximate multipliers obtained from exact multipliers using the bit-width reduction. The reduction replaces 16-bit multipliers by accurate x-bit multipliers (for $x < 16$). It ignores the LSBs of the operands and leaves the LSBs of the result zero.

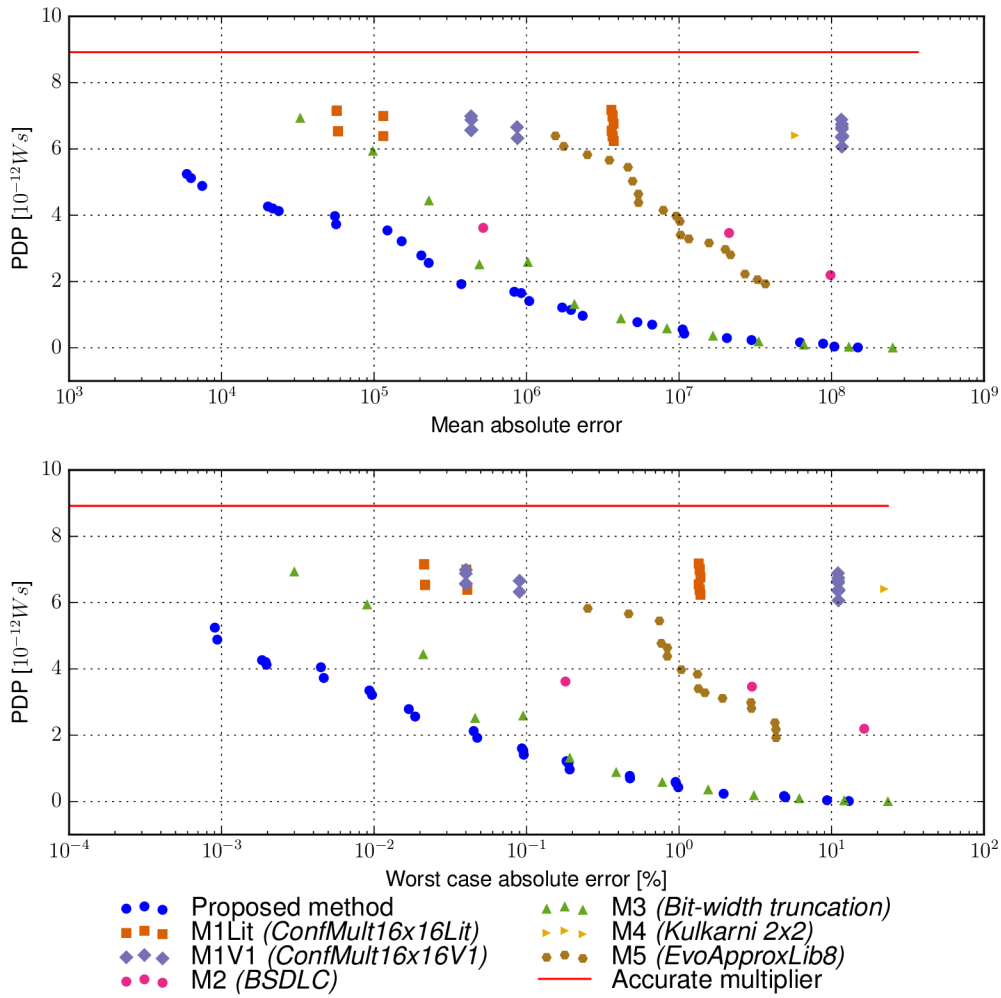


Figure 3.5: Parameters of 16-bit approximate multipliers considered in our study.

M4 The approximate multiplier composed of approximate 2-bit multipliers as proposed in [59].

M5 Approximate multipliers composed of 8-bit multipliers that are available in the EvoApproxLib library [77]. The construction principle is taken from [59].

For all considered multipliers, the value of PDP is plotted against WCAE and MAE in Fig. 3.5 (only Pareto fronts are visualized). While the bit-width reduction provides the same quality of results as our method for large target errors (up to 20% WCAE), it is significantly outperformed by our approach for small target errors. Despite the fact that the existing approximate multipliers typically exhibit good tradeoffs between the error and PDP in specific applications (as demonstrated in the relevant literature), Fig. 3.5 clearly shows that these multipliers are considerably Pareto-dominated by the multipliers obtained using our approach. These results were, in fact, expected as the proposed method is based on a global holistic optimization approach while the other approximate multipliers were composed of smaller ones and the composition procedure always introduces some overhead.

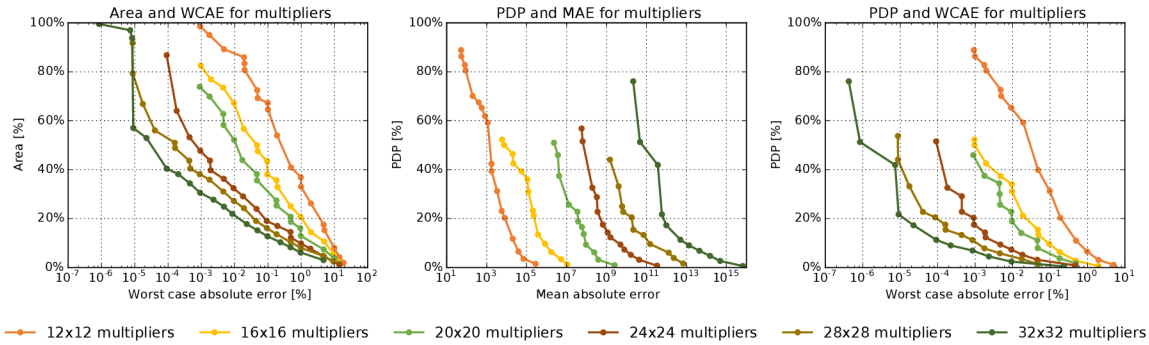


Figure 3.6: Pareto fronts showing parameters of evolved approximate multipliers. 100% refers to parameters of the accurate multiplier for a given bit width.

Finally, it is an interesting observation that MAE follows the trend of WCAE. It seems that WCAE can be used as a good indicator of MAE.

3.4.3 Complex Multipliers

The main aim of our further experiments is to show that the proposed method is scalable and can approximate complex multipliers. We present the results of the approximation process on 12-bit, 16-bit, 20-bit, 24-bit, 28-bit, and 32-bit multipliers. The target WCAEs were adapted accordingly to respect the range of values in the different considered bit widths. We used the same setup as in the previous sections but increased the time of optimization to 4 hours for the 24-bit multiplier and 6 hours for larger multipliers. The reason is that the search space becomes much bigger. Note that while the exact 12-bit multiplier contains 850 two-input gates, the 32-bit exact multiplier requires over 6,300 gates. We obtained (as the result of evolution) over 1190 unique multipliers. Because of this huge number and for the sake of clarity, Fig. 3.6 shows parameters of approximate multipliers occupying the Pareto fronts only.

In the experiments, we observed that, in the case of 12-bit multipliers, 2.4% of SAT calls were terminated on average due to the resource limit L=20K only. However, this number increased to 36.9% in the case of approximate 32-bit multipliers. For all bit widths, the MAE is around 30% of the worst-case error, which again demonstrates that WCAE is a good indicator of MAE. Fig. 3.6 also shows that the obtained approximations cover the whole range (up to 100%) of the Area axis. However, this is not the case for PDP. The reason is that we optimize the relative area and PDP is computed after the synthesis.

All Pareto fronts shown in Fig. 3.6 follow the trend of the presented fronts for the 16-bit multipliers. Since our 16-bit approximate multipliers prove to be highly competitive, we believe that the tradeoffs between the circuit error and size obtained for more complex multipliers are very good and thus the corresponding circuits represent the cutting edge of approximate multipliers and can serve as a new benchmark set for approximate computing.

3.4.4 Approximate Adders

In order to demonstrate that the proposed method is applicable for other complex arithmetic circuits, we constructed Pareto fronts for approximate adders with 20-bit to 128-bit operands. Approximation of adders is much easier than approximation of multipliers since

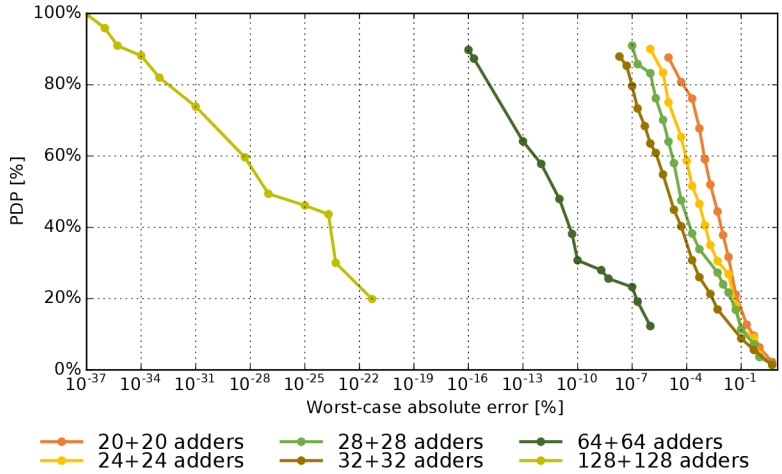


Figure 3.7: Pareto fronts showing the parameters of the evolved approximate adders. 100% refers to parameters of the accurate adder for a given bit width.

adders are structurally less complicated and the number of outputs is lower. For example, the exact 20-bit adder requires 140 two-input gates and the 128-bit adder consists of 1,000 gates.

The approximate adders were constructed using the same setup as in the previous section. A single CGP run took 2 hours for all bit widths. Fig. 3.7 shows parameters of approximate adders occupying the corresponding Pareto fronts. We report 16 to 18 non-dominated implementations of 24-bit, 28-bit, and 32-bit adders in terms of PDP and WCAE. For 64-bit and 128-bit adders, 12 tradeoffs are reported only because we have restricted the number of target error levels.

3.5 SAT-based WCRE Evaluation

In the second part of this chapter, we extend the algorithm towards the *worst-case relative error* (WCRE) that represents another important error metric capturing the worst-case behaviour of approximate circuits. Bounds on WCRE, in contrast to WCAE, require that the approximate circuits provide results that are close to the correct values even for small input values. This is essential for many application domains including, e.g., approximation of neural network hardware architectures [53]. Designing approximate circuits with WCRE bounds, however, represents a more challenging problem (when compared to WCAE) as the approximation has to preserve a larger part of the circuit logic and the circuit evaluation requires a more complicated procedure. To mitigate these challenges, we propose a novel construction of the approximation miter enabling an efficient SAT-based circuit evaluation against WCRE bounds. We integrate this evaluation procedure into the verifiability-driven circuit optimisation and thus significantly extend the existing capabilities of automated techniques for the circuit approximation. Our experiments on circuits with up to 32-bit operands show that, in many cases, the proposed approach offers a superior scalability compared to alternative methods.

For an original golden circuit G computing a function f_G and its approximation C computing a function f_C, both having n-bit inputs, we define WCRE as follows:

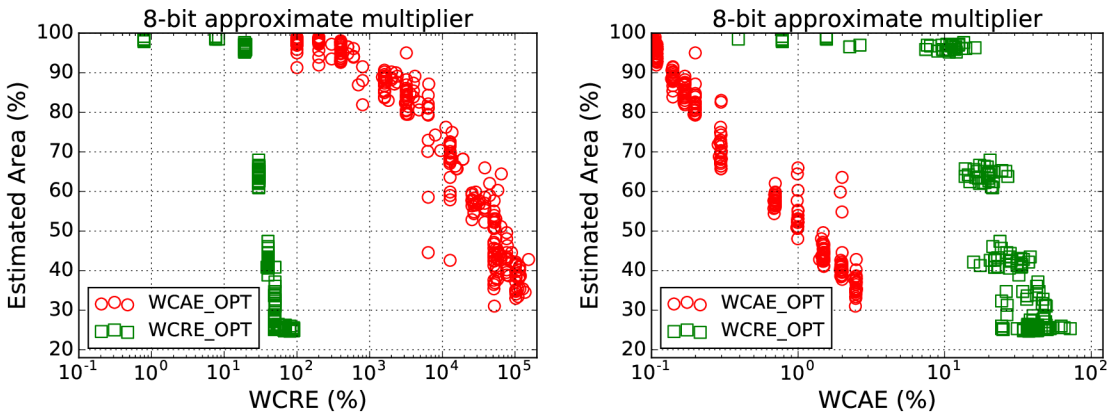


Figure 3.8: A comparison of 8-bit multipliers approximated for WCRE and WCAE. The top plot shows the trade-off between circuit area and WCRE, the bottom plot shows the trade-off between area and WCAE.

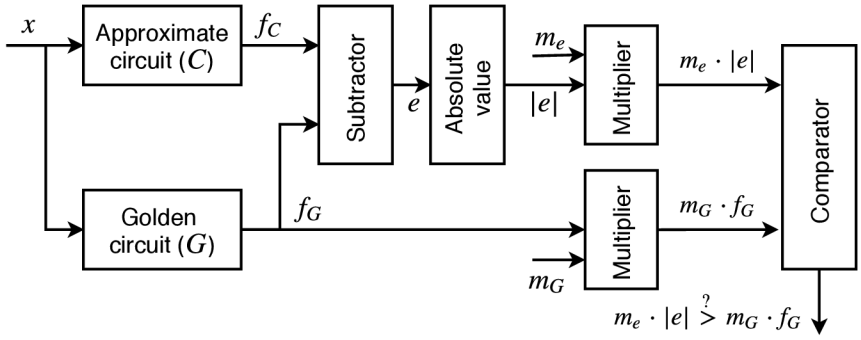


Figure 3.9: The novel approximation miter construction for WCRE evaluation.

$$\mathrm{WCRE}(G, C) = \max_{x \in \{0,1\}^n} \frac{|int(f_G(x)) - int(f_C(x))|}{int(f_G(x))}$$

Fig. 3.8 illustrates the difference between the approximation process targeting at WCAE and WCRE. This difference, in fact, motivates the work presented in this section. The figure shows two sets of circuits approximating 8-bit multipliers optimised for WCRE (green squares) and WCAE (red circles), respectively. The plots show the trade-off between the circuit area (directly effecting the power consumption) and WCRE (top) and WCAE (bottom), respectively. First, we observe that circuits optimised for WCAE have very bad WCRE and vice versa. Second, the plots demonstrate that when optimising 8-bit multipliers circuits for WCAE, we achieve about 50 % area reduction with WCAE = 1 % while we need to set WCRE = 40 % to obtain similar area improvements when optimising for WCRE. This is indeed caused by the fact that a larger part of the circuit logic has to be preserved to obtain approximations with low WCRE.

3.5.1 Checking Worst Case Relative Error

To evaluate whether the given approximate circuit meets the required bound on WCRE, we adapt and extend the miter we designed for WCAE. The miter interconnects the golden

circuit G and the candidate approximate circuit C that both share identical inputs. The subtractor and absolute value blocks allow us to quantify the absolute value of the approximation error between C and G. Here, we need to utilise additional components that compute the worst-case relative error from the absolute approximation error and compare it to the threshold value. Namely, we need to check the satisfiability of the following formula:

$$\max_{x \in \{0,1\}^n} \frac{|int(f_G(x)) - int(f_C(x))|}{int(f_G(x))} > T$$

Note that we do not need to find the maximum of the left-hand side of the formula, but rather determine if there exists a single input combination for which the bound T is violated. Therefore, we can replace the previous formula by the following constraint

$$\exists x \in \{0,1\}^n : |int(f_G(x)) - int(f_C(x))| * m_e > f_G(x) * m_G$$

where $T = m_G/m_e$. Based on this formula, we build a general WCRE miter using two multipliers by a constant and a generic comparator. The structure of the miter is visualised in Fig. 3.9.

Finally, the error is compared to a given threshold value T, and the output of the comparator is set to logical *true* if and only if the threshold T is violated. Thus the miter construction allows us to evaluate whether $WCRE(C, G) > T$ in a single SAT query. Note that, for a given approximation scenario, the threshold T is constant and can therefore be built into the structure of the comparator.

3.5.2 Variants of the WCRE Miter

Observe that the resulting WCRE miter is larger and more complex than the WCAE miter. This indeed slows down its evaluation and thus reduces the overall performance of the approximation process. To improve the performance and scalability with respect to the circuit complexity, we simplify the general WCRE miter and propose three variants of the miter that are smaller but can be used for certain values of the bound T only.

As we work with the binary representation of integers, multiplication by the powers of 2 is identical to a bit shift operation. Thus, each of the constants m_G and m_e can be expressed using two values: mc_x denoting a multiplicative constant and bs_x denoting a number of shifted bits. The subscript $x \in \{e, G\}$ denotes whether the constant in question is used to compute m_G or m_e. The original values of m_G and m_e are then computed as:

$$m_G = mc_G * 2^{bs_G} \qquad\qquad m_e = mc_e * 2^{bs_e}$$

In combinational circuits, a shift by a constant number of bits is represented by a reconnection of wires only and does not contain any logical gates. This setting allows us to remove one or even both of the constant multiplications for a subset of target WCRE error bounds T. If we restrict the values of m_g and m_e to powers of two, we suffice with utilising bit shifts only. This restricts the obtainable values of T to $1/2^{bs_e}$, e.g., 50 %, 25 %, or 12.5 %. Adding one multiplier by a constant significantly broadens the range of supported target values. These can be expressed by one of the formulas: $2^{bs_G}/(mc_e * 2^{bs_e})$ or $(mc_G * 2^{bs_G})/2^{bs_e}$. However, the constants should be kept small. Using higher values leads to larger bit widths representing the compared numbers, and therefore a more complex comparator, thus negating the contribution of this optimisation.

Table 3.2: Numbers of AIG nodes for different miters and WCAE bounds T.

	Bit shifts			One multiplier				Two multipliers			
T [%]	**12.5**	**25.0**	**50.0**	**10.0**	**33.3**	**66.7**	**80.0**	**30.0**	**42.9**	**71.4**	**85.7**
add8	226	228	233	324	327	342	360	447	510	506	519
add16	497	501	502	770	755	773	756	1079	1225	1181	1220
add32	1120	1090	1114	2074	2116	2084	2106	3125	3249	3315	3354
mult4	268	267	273	335	347	356	347	464	499	492	513
mult8	1175	1177	1183	1393	1421	1436	1414	1685	1833	1803	1841
mult12	2617	2621	2622	3032	3057	3060	3051	3512	3748	3726	3756

3.6 WCRE Experimental Evaluation

We have integrated the proposed WCRE miters into the verifiability driven search in our tool ADAC [14] (more information in Chapter 5) and evaluated its performance on a benchmark of circuit approximation problems.

3.6.1 Comparison of the WCRE Miters

In Table 3.2, we compare the size of the proposed WCRE miters. We select three target WCRE bounds T for the bit-shift variant and four target error values for one and two multiplier miter designs. The table shows the average sizes of the different variants of the miter obtained for the three chosen bit widths in adder and multiplier approximation. The size is measured in the number of nodes in the and-inverter graph (AIG) representation of the miter. AIG is a basic representation of circuits in ABC and is directly used as the input for the SAT solving procedure. A larger size of AIGs negatively affects the performance of the solver. We can see that the bit-shift variant is about a factor 2 smaller than the general construction using two multipliers. For approximation of multipliers, the differences in the size between the variants are less significant as the circuits themselves form a bigger part of the miter. Note that the average size of the WCAE miter for a 32-bit adder and 12-bit multiplier is 810 and 2437 AIG nodes, respectively, which is smaller than even the bit-shift variant of the WCRE miters for the corresponding circuits. This clearly indicates that the evaluation against WCRE is considerably harder.

Figure 3.10 illustrates how the size of the miters affects the performance of the candidate circuit evaluation. In particular, it shows the average number of evaluations per second (taken from 20 independent runs) when the approximation of adders (top) and multipliers (bottom) with different bit-widths (the x-axis) is performed. We also compare the miter-based methods with full simulation and WCAE miter evaluation.

We can observe that the simulation is considerably faster for small bit-widths, however, its performance significantly drops for circuits with operands larger than 10-bits. The proposed SAT-based approach scales much better. For the adders, it provides very good performance (around 100 evaluations per second) even for 32-bit operands. For the multipliers (representing structurally more complex circuits), the performance is much lower and drops to 10 evaluations per second for 12-bit operands. As expected, the speed of the miter evaluation slows down with increasing miter complexity—the bit-shift variant is the fastest while the version with two multipliers is the slowest. The difference in the evaluation speed is negligible for smaller circuits but becomes more significant for larger bit-widths.

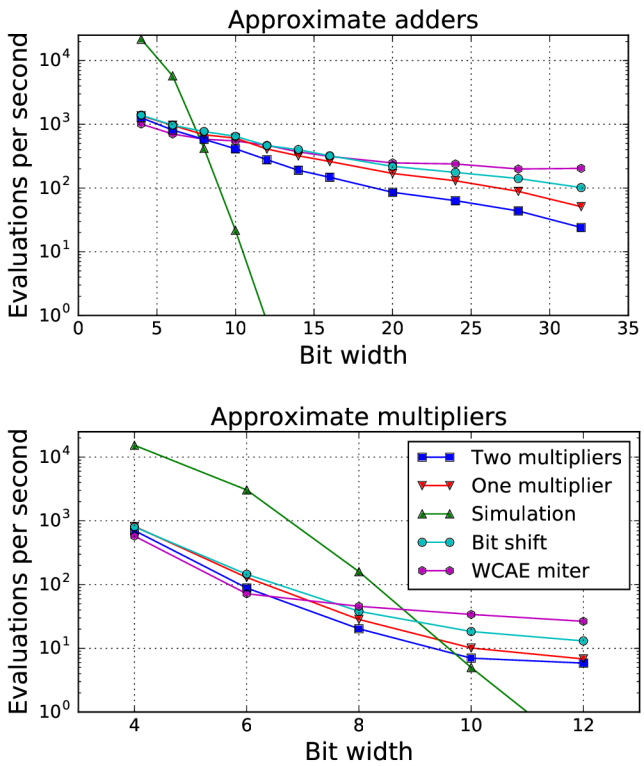


Figure 3.10: The performance of the circuit evaluation using different WCRE miters, the WCAE miter, and simulation. Top: Adders. Bottom: Multipliers.

Note that the evaluation of the WCAE miters is significantly faster due their smaller sizes (e.g. 4-times smaller for the 32-bit adders and 1.5-times smaller for the 12-bit multipliers in comparison to the two multiplier implementation).

For larger miters, the bounds on the SAT solver resources introduced by our verifiability driven search get applied. A small number of circuit evaluation tasks is skipped, e.g., for the WCRE miters, 0.7 % for the 32-bit adders and 6 % for the 12-bit multipliers.

3.6.2 Circuit Approximation

In this section, we study how the proposed SAT-based WCRE evaluation can be leveraged in circuit approximation. The optimisation is formulated as a single-objective optimisation, i.e., for a given threshold on the WCRE bound T, the approximation seeks for a circuit satisfying the bound and having the smallest circuit area. For every value of T, we run a 2-hours-long approximation process. To take into account the randomness of the evolutionary optimisation, we report the median of the circuit area obtained from 20 independent runs. Figures 3.11 and 3.12 illustrate the quality of the obtained approximate circuits for the 16-bit/28-bit adders and the 8-bit multipliers, respectively. The circuits form a Pareto front that captures the trade-offs between the area and the approximation error. The red line shows the area of the golden circuit.

For the *adders*, the proposed approximation method works well and is able to successfully approximate circuits up to 16-bit operands. Fig. 3.11 (top) shows that, for 16-bit adders, the most interesting solutions in the terms of accuracy and area savings are located

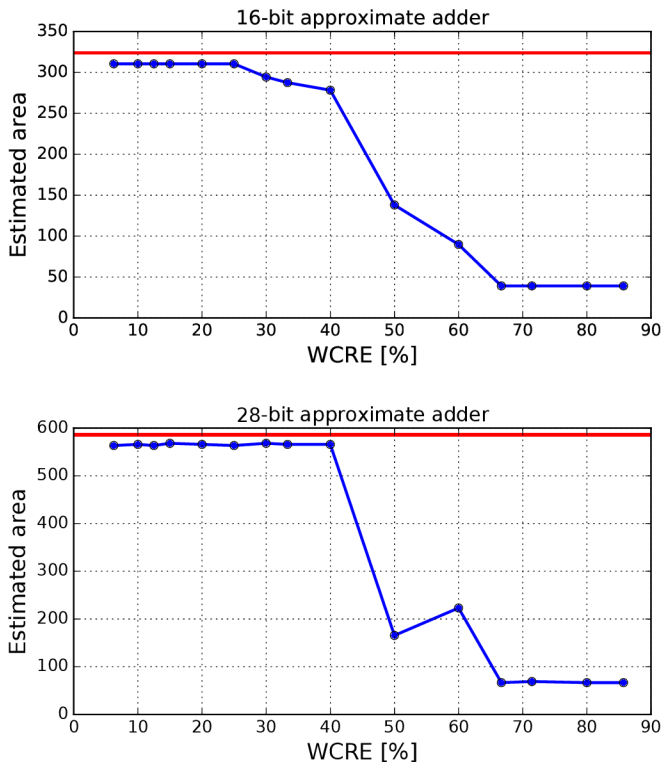


Figure 3.11: The median circuit area of 16-bit (top) and 28-bit (bottom) approximate adders approximated for WCRE. The red line indicates the area of the original circuit.

in the interval between 30 % and 60 % WCRE. For smaller target error values, the reduction of the circuit size is negligible. On the other hand, the solutions with larger approximation errors do not feature further improvements. We can also observe a dramatic area reduction between 40 % and 50 % WCRE. For adders with larger bit widths, the method is still able to find some approximations, however, the final solutions do not form a high quality Pareto front. As seen in Fig. 3.11 (bottom), the median solution for 50 % WCRE is actually smaller than the solution for the 60 % error threshold for 28-bit adders. This is caused by the complexity of the miter for 60 % which needs two multipliers by constant. The algorithm also did not find any significant improvements for the 40 % WCRE solutions.

Approximation of the *multipliers* represents a significantly harder problem. Recall that the size of multipliers (and thus also of the miters) grows quadratically with respect to their bit-widths. Therefore, the design space is larger and candidate evaluation is more complicated as discussed in the previous section. Fig. 3.12 compares the approximate 8-bits multipliers obtained using the simulation-based and SAT-based evaluation procedure. The SAT-based approach slightly lags behind the simulation mainly in the interval between 25 % and 40 % WCRE. This can be explained by the worse performance of the SAT-based evaluation on the 8-bit multipliers (recall Fig. 3.10 (bottom)).

As the performance of the simulation-based evaluation is very low beyond 10-bit multipliers, the approximation process is not able to provide a good approximation of these circuits within a 2-hours-long run. Although the SAT-based approach (namely the bit-shift solution) is able to evaluate around 10 candidates per second (for the 12-bit multipliers), the approximation process also fails to provide good Pareto sets. This is probably caused

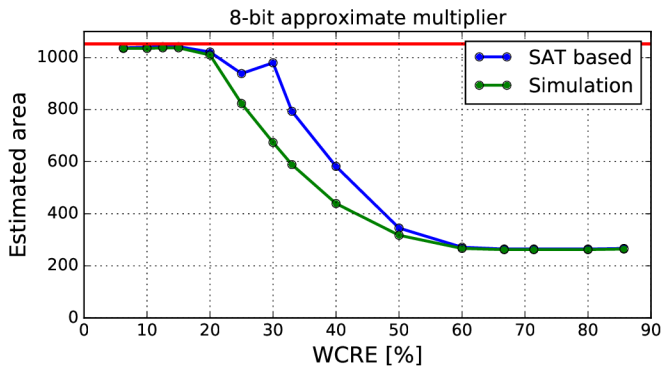


Figure 3.12: The median circuit area of 8-bit approximate multipliers approximated for WCRE. The red line indicates the area of the original circuit.

by the candidates that are skipped during the evaluation due to the resource limits on the underlying SAT solver. Note that this behaviour was not observed for the WCAE approximation that works very well even for 16-bit multipliers despite many skipped solutions. This again indicates that the WCRE approximation is very challenging, and future research is necessary in this area.

3.7 Conclusion

In this chapter, we introduced the verifiability driven search scheme that extends the Cartesian Genetic Programming. This novel search scheme opens new possibilities for SAT-based approximation procedures and allows us to successfully approximate circuits of unprecedented bit widths. We also proposed two novel approximation miter constructions that allow us to efficiently evaluate the WCAE and WCRE of approximate circuits.

The presented techniques represent an important step towards automated design of approximate circuits with formal error guarantees, which is a landmark of provably-correct construction of energy-efficient systems. When approximating circuits with the WCAE metric, our method shows excellent scalability and is able to construct high-quality Pareto sets of 32-bit multipliers and 128-bit adders. In case of WCRE approximation, we reached good results for 16-bit approximate adders, but the SAT-based approach lags behind simulation evaluation for 8-bit multipliers.

Chapter 4

Adaptive Verifiability Driven Search Strategy

The contents of this chapter are based on our work published in [15]. We extend the notion of the *verifiability-driven search* with an *adaptive strategy* that monitors the progress made by the evolution and modifies the resource restrictions accordingly. We also provide a detailed experimental evaluation that features multiple common arithmetic circuits with various bit widths.

4.1 Motivation

In this chapter, we further improve the concept of the verifiability-driven search introduced in Chapter 3. Recall that we *restrict the resources* (running time) available to the SAT solver when evaluating a candidate solution. If no decision is made within the limit, a minimal score is assigned to the candidate circuit. Shortening of the evaluation time allows our strategy to increase the number of candidate designs that can be evaluated within the time given for the entire CGP run. As shown in Section 3.4, compared with existing approximation techniques, our approach is able to discover circuits that have much better trade-offs between the precision and energy savings.

To mitigate the negative effects caused by the shortening of the evaluation time, we propose in this chapter an *adaptive control procedure* that dynamically adapts the limit on resources available to the SAT solver during the evolution. It allows the verification procedure to use more time when needed (typically at the end of the evolution) in order to discover solutions requiring a longer verification time and that would be rejected with a fixed resource limit. On the other hand, the verification time can also be shortened (typically, though not only, at the beginning of the evolution) when many suitable candidate designs are produced.

We have implemented the adaptive strategy in ADAC [14] (for more details see Chapter 5), our tool for automated design of approximate circuits, that is now able to discover complex arithmetic circuits such as 32-bit approximate multipliers, 32-bit approximate multiply-and-accumulate (MAC) circuits, and 24-bit dividers providing high quality trade-offs between the approximation error and energy savings. Such circuits have been approximated by a fully-automated approach with guaranteed error bounds for the first time.

Contributions

We propose and implement the adaptive search scheme that considerably improves the original verifiability-driven strategy. In particular, it improves the overall performance, and, more importantly, it ensures that our approach is versatile, i.e. in contrast to the method described in Chapter 3, it works well for a wide range of arithmetic circuits and approximation scenarios without manual tuning of the parameters of the evolutionary algorithm. The adaptivity is an important methodological improvement as the versatility is indeed essential for applying the approximation process into automated circuit design.

We also significantly extend the numerical evaluation to demonstrate the impact of the features described above. The evaluation newly includes approximate circuits (with different bit-widths) for multiplier–accumulators and dividers representing structurally more complex circuits when compared to the adders and multipliers typically used in the literature. It should be noted that especially MACs play an important role in many energy-aware applications – for example, MACs represent highly energy demanding components in neural network hardware architectures [52].

4.2 Adaptive Verifiability-driven Optimisation

In this section, we propose the adaptive strategy that is integrated into our optimisation scheme. The optimisation scheme now consist of four parts:

1. A *generator* of candidate solutions that builds on Cartesian Genetic Programming (Section 2.2.5).
2. An *evaluator* that evaluates the approximation error of candidate solutions by leveraging the SAT-based verification methods (Section 3.3).
3. A *verifiability-driven search* that integrates the cost of the circuit evaluation into the fitness function (Section 3.2.3).
4. An *adaptive strategy* that adjusts the allowed cost of the evaluation of candidate solutions on-the-fly during the approximation process. We describe the strategy in the rest of this section.

Adaptive resource limit strategy

In the previous Chapter 3, we performed a preliminary evaluation of the verifiability-driven search strategy studying how the limit L on the maximal number of backtracks in the SAT decision procedure affects the performance of the approximation process applied on multipliers and adders of various bit-widths. In particular, we considered 20K, 160K, and unboundedly many backtracks. The results clearly demonstrated that the evolutionary algorithm found the best solutions for the lowest of these three limit settings for a wide range of circuits. However, the question whether a still lower SAT limit would improve the performance even further remained open. Likewise, there remained a question what limits would be appropriate for circuits and bit widths other than those considered in the trials.

Apparently, the lower the limit is the faster the evaluation of each candidate solution will be. This results in the processing of a higher number of generations in a given time interval, hopefully leading to better results. On the other hand, aggressive limit settings reduce the search space of candidate solutions that can be evaluated within the given limit. A too tight

restriction might prevent the candidate solutions from diverting from the original solution and reaching significant improvements (most of the newly generated candidates will likely be skipped due to exceeding the evaluation limit). Also, the type and complexity of the approximated circuit and the approximation error can play a significant role in choosing the ideal limit settings. Thus, to reach the best performance of the method, each new instance of the problem would require an evaluation of different limit values. Moreover, a fixed limit value might not be optimal during the course of the evolutionary process even if it is optimal in some of its phases.

Therefore we propose a new *adaptive strategy* that alters the limit within the evolutionary run and tries to set it to the most suitable value with regards to the recently achieved progress. We designed the strategy scheme based on our previous observations that the limit should be kept low in the early stages of the evolution so that the clearly redundant logic can be quickly eliminated. Later in the evolutionary process, the algorithm converges to a locally optimal solution and improvements in the fitness of the candidates cease to occur. When such a stage is reached, the limit needs to be increased in order to widen the space of feasible candidate solutions at the expense of a slower candidate evaluation. Moreover, once some more significantly changed solution is found, it may again be possible to reduce the resource limit needed for the evaluation, and the process of extending and shrinking the resource limit may repeat (as witnessed also in our numerical trials).

Algorithm 2 Adapting the time limit for evaluating candidates

```
1:  lastGens ← 0
2:  improvementCount ← 0
3:  function UPDATELIMIT(limit, improvement)
4:      lastGens ← lastGens + 1
5:      if improvement then
6:          improvementCount ← improvementCount + 1
7:      if lastGens mod period = 0 then
8:          if improvementCount > τ_dec then
9:              limit ← limit − δ * limit
10:         else if improvementCount < τ_inc then
11:             limit ← limit + δ * limit
12:         improvementCount ← 0
13:         lastGens ← 0
14:     else if improvementCount > τ_res then
15:         lastGens ← 0
16:         improvementCount ← 0
17:         limit ← limit − δ * limit
18:     limit ← max(limit, minLimit)
19:     limit ← min(limit, maxLimit)
20:     return limit
```

Our strategy is described in pseudocode in Algorithm 2. The strategy changes the limit during the evolution process and is driven by four main parameters and two additional limit values with the following semantics:

- *period*: the number of generations after which a *periodic check* whether the evaluation limit should be changed is triggered.

- δ: the *increase/decrease ratio* which says by what fraction of the current limit is the limit increased/decreased when such a change is considered useful.
- τ_{dec}: if the number of improvements that occur in a period is above this threshold, the resource limit for the evaluation will be *decreased*.
- τ_{inc}: if the number of improvements that occur in a period is below this threshold, the resource limit for the evaluation will be *increased*.
- τ_{res}: if this threshold is hit, an *immediate decrease* of the resource limit and a reset of the generation counter is triggered. This threshold applies when the limit becomes clearly too high, which can happen as witnessed by our computations.
- *minLimit*: a *minimum limit bound* that restricts the possible values of the resource limit achievable by the adaptive strategy from below.
- *maxLimit*: a *maximum limit bound* that restricts the possible values of the resource limit achievable by the adaptive strategy from above.

Algorithm 2 allows the strategy to track the current progress of the evolutionary algorithm and adapt the resource limit accordingly. The key purpose of the algorithm is to keep the limit low while the evolutionary process achieves improvements in the candidate solutions, and to increase the available resources once the progress is seemingly stalled by the imposed limit. The algorithm tracks the number of improvements made in the last *lastGen* generations in the global variable *improvementCount*. In every generation, the algorithm calls the function *updateLimit* that accepts two parameters: an integer value *limit* that represents the current resource limit value and a Boolean value *improvement* that records whether an improvement occurred in the current generation. If the number of the current improvements exceeds the value of τ_{res}, the limit is immediately decreased. Otherwise, the algorithm waits until the *period* number of generations is reached (the condition on line 7 is *true*, where *mod* denotes the modulo operation) and then either increases, decreases or keeps the limit value based on the comparison of the *improvementCount* and the thresholds τ_{inc} or τ_{dec}, respectively.

The value of the increment/decrement of the resource limit is relative to the current limit value. This allows the strategy to both delicately alter small limit values and reach high limit values in reasonable time. The limit value is restricted to stay within the interval $\langle minLimit, maxLimit \rangle$. This ensures that we do not get too small limit values that would reject all candidates nor too big limit values that would feature a very long evaluation time, which would effectively stop the approximation process.

4.3 Evaluation of the Proposed Adaptive Search Approach

In this section, we present a detailed numerical evaluation of the proposed method for the evolutionary-driven circuit approximation. We first describe the computational setting and briefly discuss the CGP parameters we used in the evaluation. Afterwards, we present a thorough evaluation of the adaptive feature of our approach as well as a detailed comparison of our approach with other existing approaches. In particular, our computations focus on answering the following questions:

Q1 Can the adaptive strategy reduce the randomness of the evolution-based approximation process?

Q2 Can the adaptive strategy efficiently handle different circuit approximation problems – is it more versatile than the fixed-limit strategies?

Q3 Can the adaptive strategy outperform the best fixed-limit strategy for a given circuit approximation problem?

Q4 Does the proposed method significantly outperform other circuit approximation techniques?

4.3.1 Computational Setup

The proposed adaptive search strategy was implemented in our tool ADAC (Chapter 5). In the computations, we consider the following circuits for evaluating the performance of the proposed method[1]:

- 16-bit multipliers (the input is two 16-bit numbers) having 1525 gates (501 xors and logic depth 34),
- 24-bit multipliers having 3520 gates (1157 xors and logic depth 40),
- 24-bit multiply-and-accumulate (MAC) circuits (the input is two 12-bit numbers and one 24-bit number) having 1023 gates (321 xors and logic depth 39),
- 32-bit MAC circuits having 1788 gates (565 xors and logic depth 44),
- 20-bit squares (the input is one 20-bit number, the result is the second power of the input) with 2213 gates (789 xors and logic depth 38),
- 28-bit squares with 4336 gates (1547 xors and depth 40).
- 23-bit dividers (the input is 23-bit and 12-bit numbers) having 1512 gates (253 xors and logic depth 455),
- 31-bit dividers with 2720 gates (465 xors and depth 799).

It should be noted that the number of gates directly affects the complexity of the approximation process—the more gates a circuit has, the harder the approximation problem is. The number of XOR gates is especially important as they are difficult for the current SAT solvers to deal with. We further report the depth of the circuits as another important structural characteristic. It should also be noted that we use the synthesised gate level implementations (i.e. the golden models) as the seeding circuit (i.e. the initial B-circuit) for the evolutionary approximation.

Recall that we consider the circuit size as the key non-functional characteristic we want to improve by allowing an error in the circuit computation. To estimate the circuit size, we use the gate sizes listed in Table 3.1. These sizes correspond to the 45nm technology which we consider in Section 4.3.7 when comparing the power-delay product (PDP)[2] of our resulting circuits with state-of-the-art solutions.

[1]Gate-level implementations of the considered multipliers, MACs and squares were designed using the Verilog "$*$" and "$+$" operators and subsequently synthesised by the Yosys hardware synthesis tool using the gates listed in Table 3.1. Gate-level representations of the dividers were created according to [89].

[2]Power-delay product is a standard characterisation capturing both the circuit power consumption and performance.

Justification for the selected benchmark set:

Approximation of 16-bit multipliers represents the cutting edge of circuit approximation techniques due to the circuit size (i.e. the number of gates) and structural complexity (i.e. the presence of carry chains), especially when some formal error guarantees are expected from the approximation method [50]. We use such multipliers in Section 4.3.7 to compare our approach with state-of-the-art techniques. The other circuits we consider go beyond this edge: MACs have a more complicated structure and the error of the involved multiplication is further propagated in the consequent accumulation. Square circuits computing the second power of the input represent a specialised version of multipliers. While these circuits feature less inputs than other examined instances, their internal structure is much more complex than the structure of arithmetic circuits with comparable input bit widths. Approximation of dividers represents a true challenge since they are structurally more complicated, much deeper, and significantly less explored (e.g. when compared with multipliers).

For all 8 circuits, we consider various WCAE values, namely, we let WCAE range from $10^{-4}\%$ to 1%. The given bound on the WCAE value determines permissible changes in the circuit structure (i.e. a small error allows only smaller changes in the circuit). Therefore, different WCAE values lead to significantly different approximation problems. We also consider two time limits (1 and 6 hours) for the approximation process. It should be noted that the time limit also considerably affects the approximation strategy as the given time has to be effectively used with respect to the complexity of the approximation problem.

In our evaluation, we explore all three dimensions characterising the circuit approximation problems: i) the circuit type reflecting both the size and the structural complexity, ii) the error bound, and iii) the approximation time. In total, we examine more than 70 instances of the approximation problems that sufficiently cover practically relevant problems in the area of arithmetic circuit approximation. Therefore, the considered benchmark allows us to answer the research questions and, in particular, to robustly evaluate the versatility of the adaptive strategies and their benefits with respect to the fixed-limit strategies.

It should be noted that we exclude adders from the present evaluation as they represent a much simpler approximation problem – comparing to 16-bit multipliers, 128-bit adders have only around two thirds of the gates and are structurally less complex. Therefore, the miter-based error evaluation can handle these circuits without leveraging the resource limits.

4.3.2 CGP Parameters

The performance of CGP for a given application domain can heavily rely on the various CGP parameters. The following parameters are especially relevant in the case of circuit approximation:

- the number of offspring (λ),
- the frequency of mutations, and
- the CGP grid size and the `L-back` parameter (i.e. connectivity in the chromosome).

We now briefly discuss our choice of the values of these parameters that will later be used for the main part of the evaluation of the proposed method.

As shown in [112], for a fixed number of generated and evaluated candidate solutions, CGP-based circuit optimisation with a smaller value of λ usually leads to a better fitness values than CGP using larger values of λ.

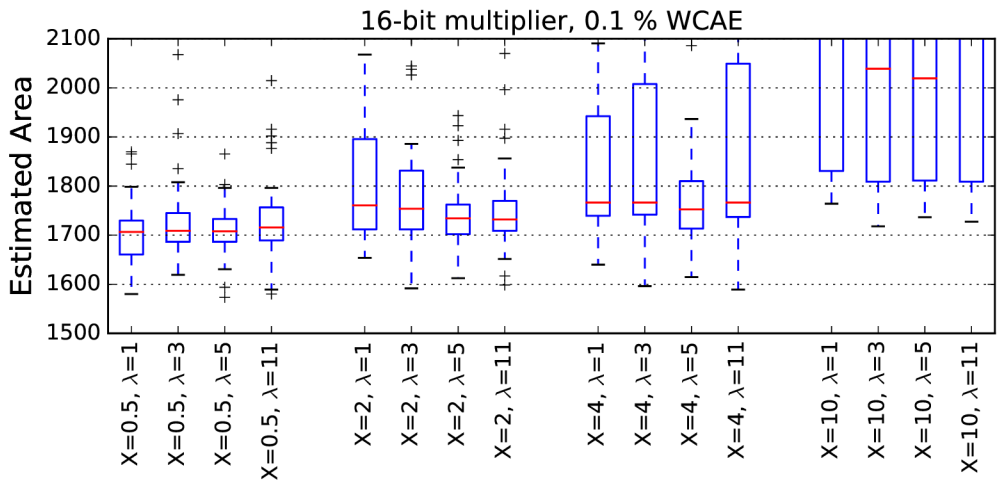


Figure 4.1: The impact of the number of offspring (λ) and mutation frequency (X) on the final circuit area of approximated 16-bit multipliers obtained by CGP with a fixed time limit for each evolutionary run.

Aside from the population size, we also examine the effect of the mutation frequency on the performance of circuit approximation. Each time the mutation operator is applied, it alters a single integer in the chromosome. When we generate a new candidate from a parent, we apply the mutation operator up to M times, $M = 0.01 * X * gates(G)$ where X is the mutation frequency parameter and $gates(G)$ is the number of gates of the golden solution G that is approximated. In the particular trial of 16-bit multiplier approximation, in which $gates(G) = 1525$, the performance was evaluated for $X \in \langle 0.5, 10 \rangle$, i.e. $M \in \langle 8, 153 \rangle$ mutations per chromosome.

Fig. 4.1 provides the results of the approximation of a 16-bit multiplier with 0.1 % WCAE using different combinations of λ and X (the x-axis). The y-axis characterises the estimated size (obtained as the sum of gate sizes) of the best candidate found in every 2-hour run. The SAT resource limit was set to 100. We do not present results for other approximate circuits as they exhibit similar patterns. The boxplots are grouped by mutation frequency. We can see that the performance within each group is very similar and lower mutation frequencies perform better than higher mutation frequencies. We also applied Friedman and Nemenyi statistical tests [26, 33, 83] to evaluate these results. According to the Nemenyi post hoc test, the differences between various λ values within the same mutation frequency are not significant at $\alpha = 0.05$. Mutation frequencies $X = 0.5$ and $X = 2$ are equivalent and perform significantly better than $X = 4$ and $X = 10$, according to the statistical tests.

The numerical trials confirm the general observations known from the literature (see, e.g. [78]): the number of mutations should be small. This way, the mutations perform slight changes between the generations only. Otherwise, for a high mutation frequency, the function of a new solution is usually completely altered. Such a solution is then rejected with a high probability, the search gets close to a random one, and its efficiency deteriorates. Therefore, in the following computations, we choose the mutation frequency $X = 0.5$ %. As population size does not seem to significantly matter, we choose the simplest $\lambda = 1$ scheme.

As for the chromosome dimensions and the `L-back` parameter, we use the same settings as presented in Section 3.4.1. Namely, we arrange all the gates of the chromosome gate matrix in a single row (dimensions $1 \times W$ where W equals the number of gates in the correct circuit) and use `L-back` $= W$, which allows for maximum connectivity.

4.3.3 Comparison of Adaptive Strategies

In the next cycle of simulations, we examine the different versions of the adaptive strategy. Each version corresponds to a different instantiation of the adaptive scheme presented in Section 4.2. The goal of this phase is to select the best adaptive strategies that efficiently work for a wide class of approximation problems. These strategies are further thoroughly evaluated and compared with fixed limit strategies on the selected benchmark.

Based on our experience with the limit values used in Chapter 3, we consider five versions of the adaptive strategy given by the parameter values listed in Table 4.1. These versions have been chosen to adequately cover the space of adaptive strategies and thus they range from strategies that try to promptly react to changes in the evolutionary process ($ada1$, $ada3$) to strategies that evaluate the progress of the evolution over longer periods of time ($ada2$). The remaining strategies ($ada4$, $ada5$) lie in the middle of the range.

The strategies differ mainly in two basic aspects: the length of the period with which the adaptation happens and the thresholds used for the adaption (τ_{dec}, τ_{inc}, τ_{res}). Larger values of the thresholds with respect to the *period* mean that the resource limit will more likely be increased. Strategies with such thresholds ($ada1$, $ada3$ and $ada5$) are faster to magnify the limit once the evolution seemingly gets stuck in a local optimum. Thus, the possible search space is broadened, but each candidate evaluation is likely to take longer time. On the other hand, strategies $ada2$ and $ada4$ try to keep the resource limit as low as possible, and each evaluation is therefore very fast. However, once there are no improvements possible with the current limit value, these strategies are slower to react.

The minimal limit value $minLimit$ and the maximum limit value $maxLimit$ are set to 500 and 15,000, respectively, based on the experience we gained from our previous work (see Chapter 3).

Table 4.1: Adaptive strategy parameters.

Strategy	τ_{dec}	τ_{inc}	τ_{res}	period	minLimit	maxLimit
ada1	4	2	10	1000	500	15000
ada2	2	1	5	15000	500	15000
ada3	4	4	8	3000	500	15000
ada4	1	1	3	5000	500	15000
ada5	5	4	8	5000	500	15000

We evaluate the performance of the described strategies on the approximation scenario of 16-bit multipliers with a total of 8 target WCAE values ranging from 10^{-4}% to 1% with 50 independent 1 hour and 6 hour long evolutionary runs. The quality of the obtained final solutions was evaluated using Friedman and Nemenyi statistical tests with results illustrated in Table 4.2. In the table, we report the pairwise p-values of the Nemenyi statistical test and the average rank values. The rank value for a single run experiment is computed as follows. First, we perform a single evolutionary run for each adaptive strategy. Then, we sort the final solutions from the best (smallest size) to the worst and the best solution is

assigned rank 1, the second best rank 2, etc. In the table, we report the average rank, which is computed as the average of rank values over 50 independent evolutionary runs. A smaller average rank indicates, that the strategy tends to provide better solutions than the strategies with a higher average rank.

For 1h runs, strategy *ada*5 performs the best, but its performance is statistically equivalent to *ada*4 and *ada*3. This group of strategies is significantly better than *ada*1 and *ada*2.

For 6h runs, *ada*2 significantly outperforms the rest of strategies, followed by *ada*4 which also significantly outperforms its successors.

In the overall evaluation, the performance of strategies *ada*2, *ada*4, and *ada*5 is statistically equivalent and significantly better than the performance of strategies *ada*1 and *ada*3. As we aim to acquire the best solutions that our method can provide, we select the strategies *ada*2 and *ada*4 as representatives of adaptive strategies for the following computations.

Table 4.2: Pairwise p-values of Nemenyi statistical test and average rank values for 1h trials, 6h trials and both trials combined.

1h runs	*ada*1	*ada*2	*ada*3	*ada*4	*ada*5
*ada*2	0.74708	-	-	-	-
*ada*3	**6.50E-06**	**0.00155**	-	-	-
*ada*4	**4.30E-07**	**0.00019**	0.98709	-	-
*ada*5	**9.20E-13**	**4.10E-09**	0.09467	0.27627	-
Rank	3.4275	3.2925	2.87125	2.815	2.59375
6h runs	*ada*1	*ada*2	*ada*3	*ada*4	*ada*5
*ada*2	**2.00E-16**	-	-	-	-
*ada*3	0.28188	**4.90E-14**	-	-	-
*ada*4	**5.40E-14**	**0.01082**	**3.10E-12**	-	-
*ada*5	**0.00013**	**6.50E-14**	0.12034	**9.10E-06**	-
Rank	3.6275	2.23125	3.4075	2.5925	3.14125
Overall	*ada*1	*ada*2	*ada*3	*ada*4	*ada*5
*ada*2	**6.10E-14**	-	-	-	-
*ada*3	**9.00E-06**	**1.80E-05**	-	-	-
*ada*4	**5.20E-14**	0.9483	**3.60E-07**	-	-
*ada*5	**5.30E-14**	0.6686	**0.0053**	0.2326	-
Rank	3.5275	2.761875	3.139375	2.70375	2.8675

We further show how the adaptive strategies *ada*2 and *ada*4 change the resource limits during the approximation process. Fig. 4.2 shows how the limits change (increase as well as decrease) over the time during the approximation of the 16-bit multipliers with target 0.1 % WCAE. The approximation ran for 6 hours, and the plot shows the maximum number of SAT backtracks (i.e. the resource limit) that was allowed to be used during the verification of candidate circuits in particular generations of the evolutionary optimisation. The top two plots of the figure illustrate five selected runs for both strategies, and the bottom plot shows the aggregated results for 50 independent runs. It shows the median of the resource limits plotted by the full lines and quartiles Q1 and Q3 plotted by the dashed lines. The information about largest and smallest limit values is omitted for better readability.

The figure confirms our expectations: in the initial stages of the approximation, the limit is kept low because improvements are found frequently. We can also see that the limit increases as well as decreases, and a closer evaluation of our data reveals that both

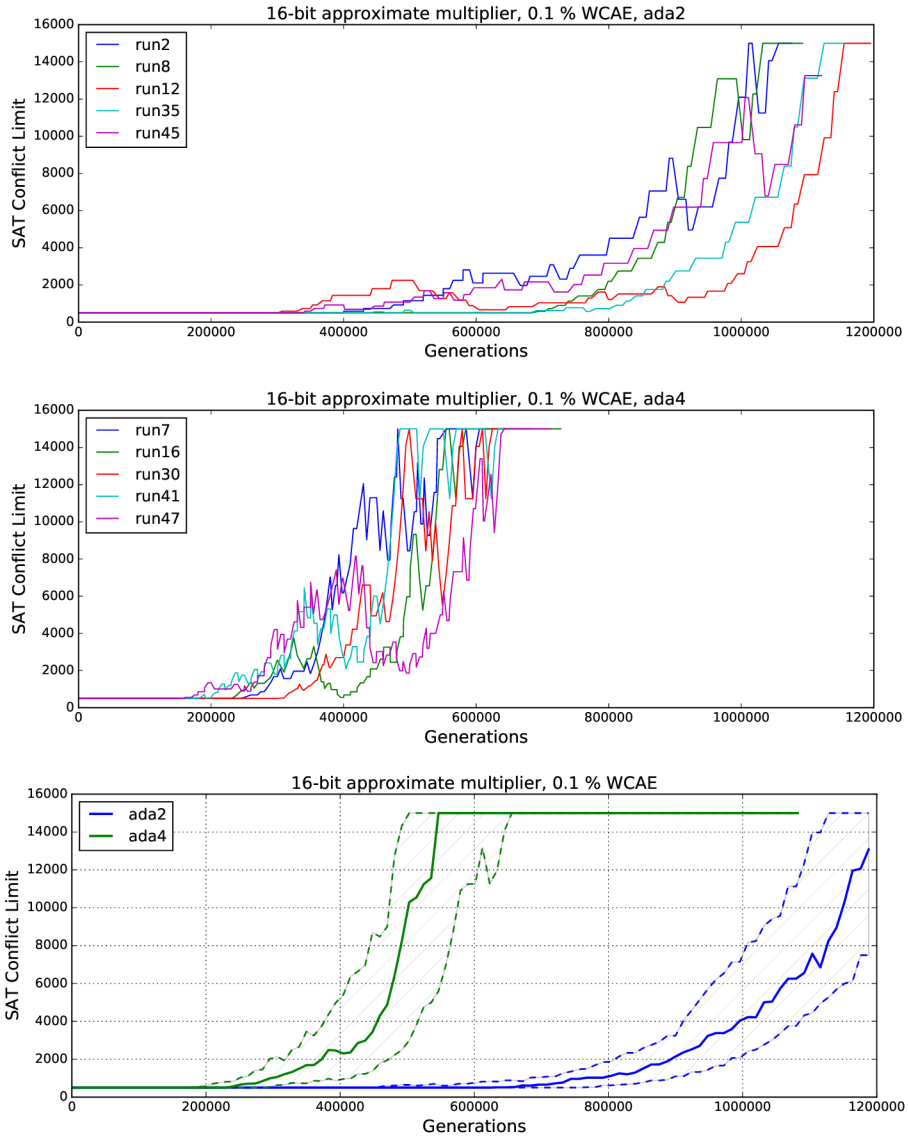


Figure 4.2: The resource limits chosen by the adaptive strategies *ada*2 and *ada*4 during the approximation of the 16-bit multiplier. The top two plots illustrate five selected runs for each strategy. The bottom plot shows the medians (full lines) and the quartiles Q1 and Q3 (dashed lines) over 50 approximation runs.

the periodic and immediate decreases are used. Furthermore, it should be noted that *ada*4 increases the limit much sooner than *ada*2, and the rate of the increase is also much steeper. This fact allows *ada*4 to use more time out of the total time available for the entire evolutionary run for evolving and evaluating solutions that need larger resource limits for their verification. On the other hand, the higher limit slows the evolutionary process down significantly—we see that none of the *ada*4 runs reaches the number of $1.2 * 10^6$ generations in this numerical trial. The particular runs of the strategies also demonstrate that *ada*4 exhibits more changes (including periodic drops of the limits) compared with the more stable strategy *ada*2. The impact of these differences on the quality of the obtained final solutions will be evaluated in the further subsections of this section.

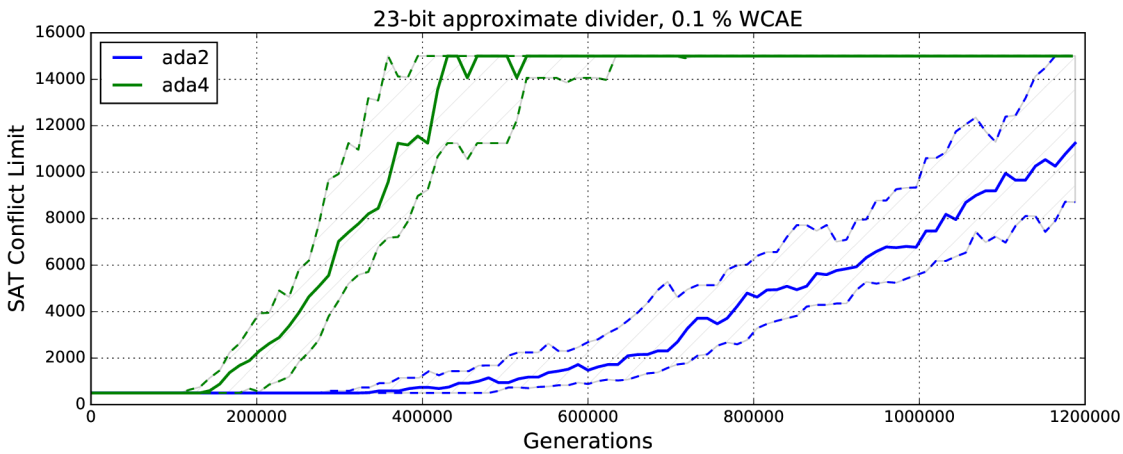


Figure 4.3: The resource limits chosen by the adaptive strategies $ada2$ and $ada4$ during the approximation of a 23-bit divider. The plot shows the median values (full lines) and the quartiles Q1 and Q3 (dotted lines) over 50 runs.

Finally, Fig. 4.3 shows the aggregated results for approximation of 23-bit dividers (with the same WCAE) representing a very different approximation scenario. We observe that the approximation of the dividers requires higher resource limits (i.e. more time for the verification of the candidate solutions) when compared with the multipliers: this is due to the structural complexity of the circuits. For example, in the 400K-th generation, $ada4$ sets the limit to about 2K for the multipliers and to about 12K for the dividers. The difference is, however, less significant in the case of $ada2$.

4.3.4 Reduction of Randomness (Q1)

Evolutionary algorithms involve a significant amount of randomness, and the quality of the final solutions produced by independent runs can considerably vary. One of the goals of the newly designed adaptive strategies is to reduce the amount of the involved randomness and ensure that most of the approximation runs will lead to high-quality solutions. In these numerical trials, we analyse the quality and variability of sets of 50 independent evolutionary runs for the adaptive strategies as well as for various fixed limit resource settings.

For the following simulations, we denote the fixed resource limit strategies as $lim100$, $lim2K$, $lim10K$, $lim20K$, and $lim50K$ for the resource limits of 100, 2000, 10000, 20000, and 50000 backtracks on a single variable, which are used through the whole evolutionary process. We chose these values to represent small, mid range and large values. Recall that in the previous chapter, we used $lim20K$ as the standard resource limit setting.

The plots in Figures 4.4 and 4.5 demonstrate how the size of the candidate solutions decreases during the evolutionary runs. In particular, the dashed red lines show the best and the worst run; and median, first (Q1) and third (Q3) quartile are illustrated by the full blue line and the red lines, respectively.

Fig. 4.4 shows that the adaptive strategies as well as the strategies with lower resource limit values are significantly more stable than the strategies with higher limits. This is caused by the fact that the evolution sometimes has to explore solutions requiring a long verification time. Such solutions are immediately refused by the lower resource limits (100, $2K$) and by the adaptive strategies but more likely accepted by the other strategies ($10K$,

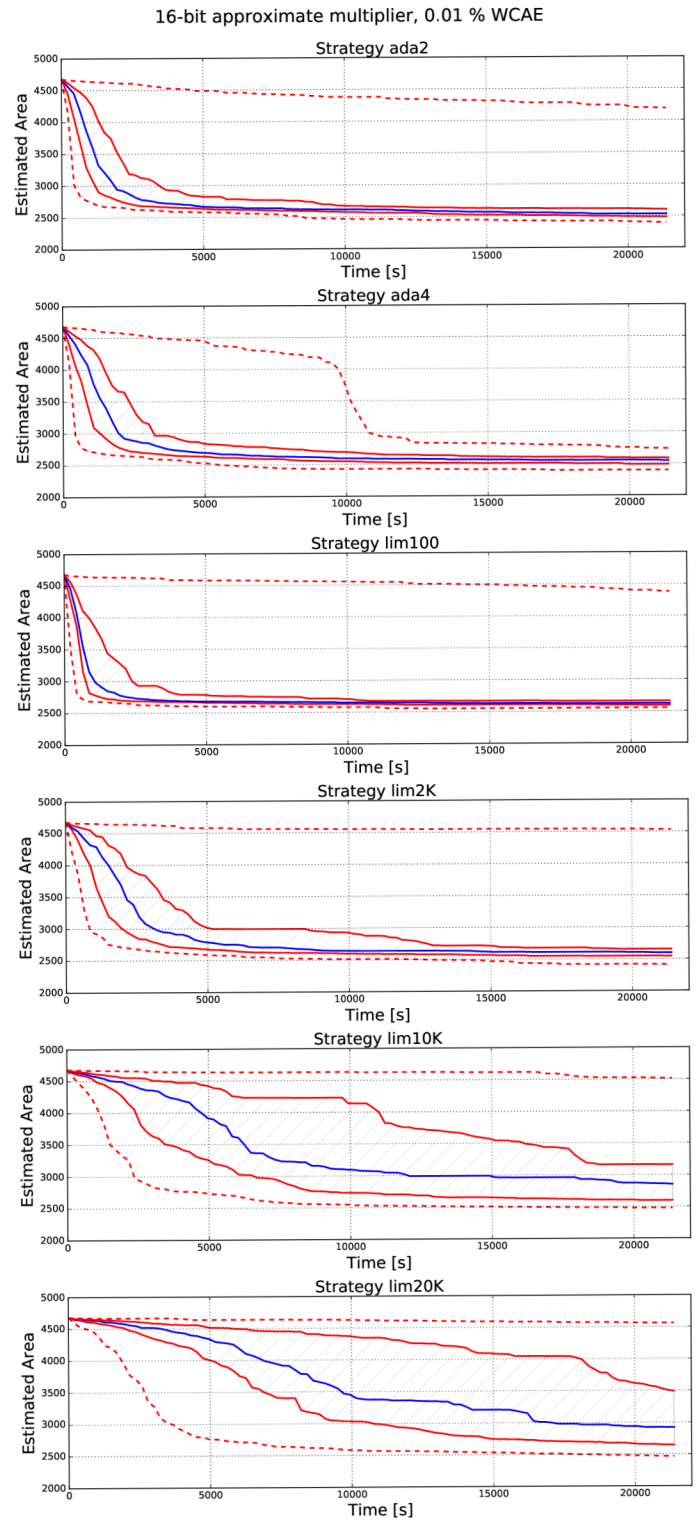


Figure 4.4: Convergence curves for resource limit strategies showing the estimated area for the best, worst, Q1, Q3, and median solutions during 16-bit multiplier approximation.

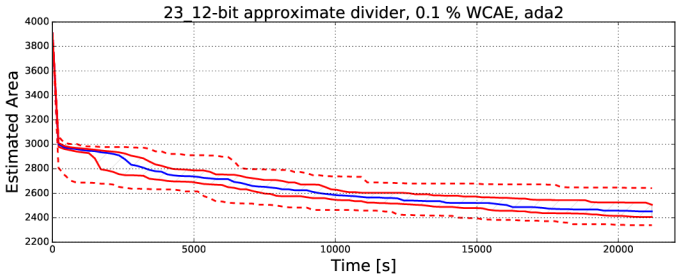


Figure 4.5: Convergence curves for resource limit strategy *ada*2 showing the estimated area for the best, worst, Q1, Q3, and median solutions during 23-bit divider approximation.

$20K$, and $50K$). The long evaluation time is inherited from parents to offspring. The strategies with higher limit settings are therefore much slower to converge to a near optimum solution. The previously described slowdown of the evolution also leads to higher variation in the candidate quality throughout the evolution, which can be observed as the wide interquartile range (IQR) for limits $10K$, $20K$, and $50K$. Other strategies feature a narrow IQR—a desirable attribute of a good resource limit strategy. The convergence of the strategy $lim50K$ is so slow that we exclude this strategy from the rest of the simulations to save computational time.

We obtained similar observations for other WCAE values and bit-width settings for multipliers, MACs and square circuits. The difference between the strategies is even more pronounced for smaller approximation errors which represent a harder optimisation problem. On the other hand, large approximation errors diminish the differences.

The approximation of dividers represents another class of optimisation problems with a different behaviour. The variance of the solutions is very similar for all resource limit settings. As an example, the variance for approximation of a 23-bit divider with the strategy *ada*2 is illustrated in Figure 4.5. Plots for other resource limit strategies are omitted as they feature almost identical variance. What differs between the strategies is the quality of the final solutions that can be obtained. This is described in greater detail in the next section.

Summary for Q1: For a wide class of circuits, the adaptive strategies as well as the low-limit strategies are significantly more stable than other fixed limit strategies (i.e. the effect of the randomness is smaller). All strategies show good stability for the approximation of dividers, however, the low-limit strategies provide considerably smaller reductions of the circuit area.

Note: Since it would be very difficult to present our results while also showing the randomness of the evolutionary runs at the same time, we present only the quality of the median solutions in the rest of this chapter when not stated otherwise.

4.3.5 Versatility of Adaptive Strategies (Q2)

The key feature of circuit approximation strategies is *versatility*, an ability to provide excellent performance for various approximation scenarios including different circuits, WCAE values and time limits. Although the verifiability-driven strategy itself leads to unprecedented performance and scalability of circuit approximation [13], the fixed-limit resource limits do not ensure versatility. This fact is demonstrated in Fig. 4.6 where we fix WCAE to

0.1 % for multipliers, squares, MACs, and dividers, and explore the progress of the approximation process. The right part of each plot illustrates the quality of the final solutions.

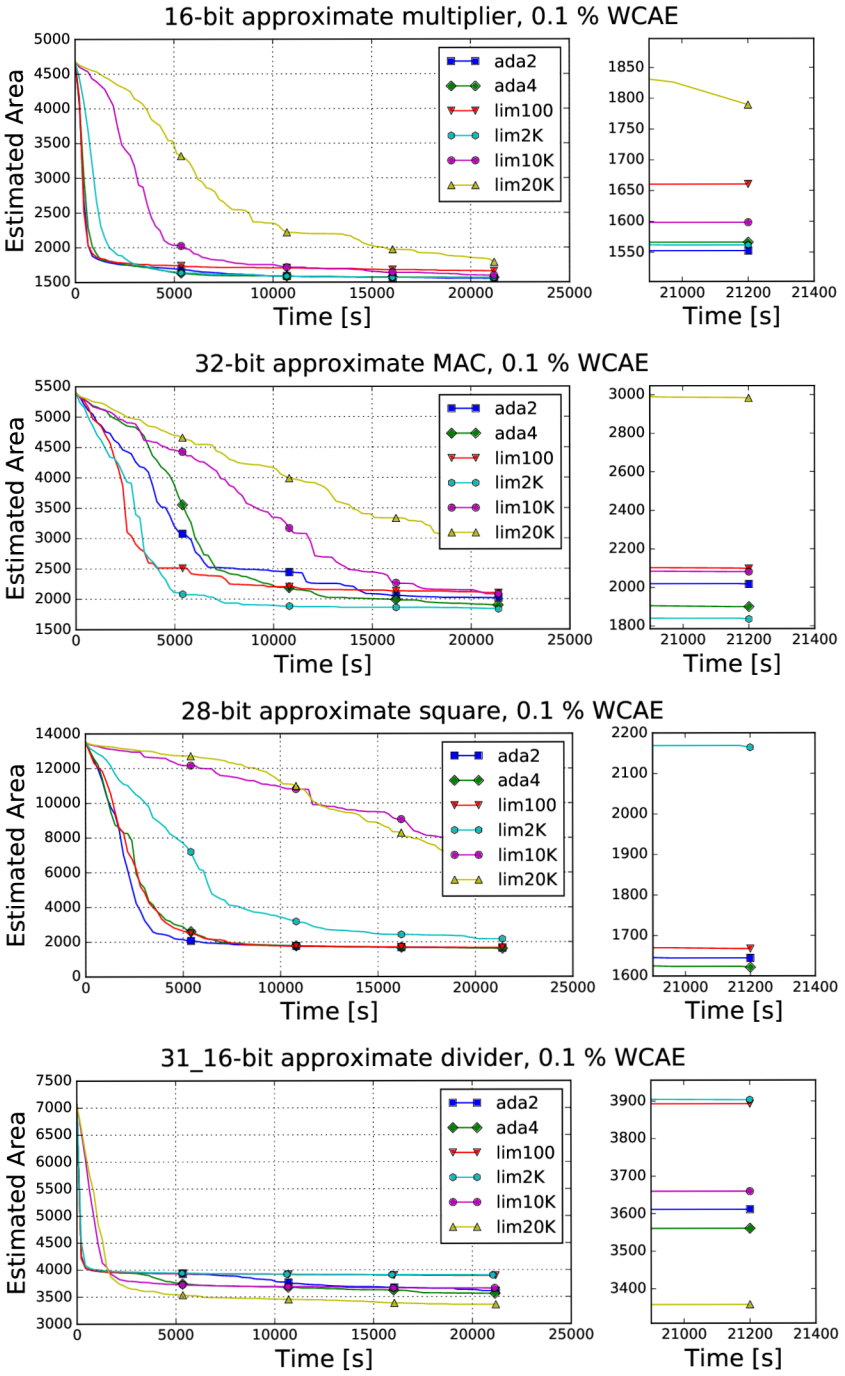


Figure 4.6: Convergence plots of the median solutions calculated from 50 independent evolutionary runs for various combinational circuits.

When comparing the performance of the fixed-limit strategies on the approximation of 16-bit multipliers, we can see that the strategy $lim100$ dominates in the first hour of the approximation process since it provides the fastest convergence. Strategies $ada2$, $ada4$, and $lim2K$ converge slower, but around after the first hour their median solutions outper-

form $lim100$ which cannot achieve further improvements due to the tight resource limit. Strategies $lim10K$ and $lim20K$ provide a significantly slower convergence: $lim10K$ needs around 2.5 hours to provide solutions that are comparable to the aforementioned strategies, $lim20K$ is too slow and its final solution lags behind.

Table 4.3: Relative sizes in % of the median solutions with respect to the size of the golden solution for multiplier approximation. The left column shows the target WCAE threshold.

16-bit multiplier

1h runs	$lim100$	$lim2K$	$lim10K$	$lim20K$	$ada2$	$ada4$
0.001 %	82.2	85.8	95.7	98.1	82.4	**81.1**
0.01 %	61.1	60.4	88.6	96.7	59.3	**57.9**
0.1 %	37.7	37.0	58.9	86.2	36.8	**36.7**
1 %	18.8	17.9	20.2	43.5	18.6	**17.7**
6h runs	$lim100$	$lim2K$	$lim10K$	$lim20K$	$ada2$	$ada4$
0.001 %	74.0	72.8	77.6	82.4	72.5	**71.5**
0.01 %	56.4	55.4	56.8	63.2	55.0	**54.0**
0.1 %	35.5	33.4	34.6	38.5	**33.2**	33.5
1 %	17.4	**15.7**	16.4	17.2	**15.7**	15.9

24-bit multiplier

1h runs	$lim100$	$lim2K$	$lim10K$	$lim20K$	$ada2$	$ada4$
0.001 %	91.2	87.6	96.7	97.7	89.0	**87.2**
0.01 %	32.1	59.7	89.3	94.4	32.8	**31.5**
0.1 %	19.0	19.8	78.7	86.3	**18.4**	18.5
1 %	9.2	**8.6**	21.1	79.9	9.1	8.9
6h runs	$lim100$	$lim2K$	$lim10K$	$lim20K$	$ada2$	$ada4$
0.001 %	43.0	**40.4**	79.3	82.6	41.8	41.0
0.01 %	27.1	27.1	30.2	42.7	26.8	**26.3**
0.1 %	16.3	**15.9**	18.1	23.4	**15.9**	16.0
1 %	8.7	7.6	7.6	8.6	**7.4**	**7.4**

Similar trends among the inspected strategies are observed for the 32-bit MACs (see the second plot in Fig. 4.6). It should be noted that, in general, the convergence is much slower because this circuit is larger and represents a harder optimisation problem compared with the multipliers. Moreover, we observe a larger diversity among the strategies.

The progress tendencies for 28-bit square circuit (see the third plot in Fig. 4.6) significantly differ. The strategies $lim10K$ and $lim20K$ provide an extremely slow convergence and even after 6-hours runs they significantly lag behind the other strategies. The strategy $lim2K$ also converges much slower than the remaining strategies, which show similar performance. After an 1-hour run, $lim2K$ returns circuits that are about two-times larger than the circuits provided by the strategy $lim100$, however, after 5 hours it catches up with the other strategies.

The bottom part of Fig. 4.6 illustrates the results for the dividers. We observe a very different trend in the approximation process. In particular, all strategies converge very quickly to a sub-optimal solution, but the fixed-limit strategies with small resource limits

Table 4.4: Relative sizes in % of the median solutions with respect to the size of the golden solution for MAC approximation. The left column shows the target WCAE threshold.

24-bit MAC

1h runs	*lim*100	*lim*2*K*	*lim*10*K*	*lim*20*K*	*ada*2	*ada*4
0.0001 %	**96.7**	96.9	97.7	97.8	**96.7**	97.1
0.001 %	94.3	**93.7**	95.0	95.3	94.0	93.9
0.01 %	91.3	**82.1**	83.0	95.5	90.0	93.9
0.1 %	73.1	67.8	93.2	92.6	65.8	**64.1**
1 %	38.2	28.9	45.4	67.9	31.1	**28.5**
6h runs	*lim*100	*lim*2*K*	*lim*10*K*	*lim*20*K*	*ada*2	*ada*4
0.0001 %	95.9	96.2	95.8	96.2	**95.5**	95.8
0.001 %	92.3	90.4	89.4	89.4	88.7	**88.5**
0.01 %	84.9	76.0	75.0	78.7	76.6	**75.2**
0.1 %	59.1	56.7	61.2	65.6	53.1	**53.0**
1 %	31.8	27.3	26.1	26.9	**24.7**	24.9

32-bit MAC

1h runs	*lim*100	*lim*2*K*	*lim*10*K*	*lim*20*K*	*ada*2	*ada*4
0.0001 %	**94.6**	95.7	98.7	98.9	95.0	94.8
0.001 %	94.3	94.0	97.5	97.6	93.8	**93.6**
0.01 %	87.1	**81.2**	90.0	95.5	85.5	89.3
0.1 %	87.1	**57.8**	85.7	90.8	77.3	86.2
1 %	24.8	**19.1**	32.0	58.3	19.4	19.7
6h runs	*lim*100	*lim*2*K*	*lim*10*K*	*lim*20*K*	*ada*2	*ada*4
0.0001 %	93.7	88.9	93.7	94.3	91.2	**88.0**
0.001 %	91.9	78.0	83.4	80.8	80.5	**76.9**
0.01 %	61.1	60.4	55.6	62.5	**57.9**	62.7
0.1 %	39.7	**34.0**	39.1	54.8	37.4	35.1
1 %	20.3	17.1	16.4	16.4	15.5	**15.3**

(*lim*100 and *lim*2*K*) are not able to achieve any further improvements, and they significantly lag behind the other strategies in the final solutions. We further observe that the strategy *lim*20*K*, which performs very poorly on the previous circuits, is the best strategy in this case. The proposed adaptive strategies inherit the initial fast convergence using a small limit, but they adapt the limit after the first hour and arrive to results comparable with the strategy *lim*10*K*.

Fig. 4.6 indicates that the performance of the particular fixed-limit strategies fundamentally varies for different circuits under approximation. For example, the strategy *lim*2*K* gives the best results for the MACs, but it behaves very poorly on the dividers, which clearly require a very high resource limit. In Tables 4.3 – 4.6, depicting the results for particular circuits, we show that the selection of the best strategy also depends on the required WCAE and on the bit-width of the particular circuits. The tables list the relative size reductions of the median solutions with respect to the golden circuit obtained using different strategies after 1 and 6 hours for various circuit types, bit-widths and WCAEs. The best solution for each target approximation error is highlighted in bold text. For instance, Table 4.3 shows that the median solution for 16-bit multipliers with 0.01 % WCAE obtained by *ada*4 in 6

Table 4.5: Relative sizes in % of the median solutions with respect to the size of the golden solution for divider approximation. The left column shows the target WCAE threshold.

23-bit divider

1h runs	$lim100$	$lim2K$	$lim10K$	$lim20K$	$ada2$	$ada4$
0.05 %	76.8	76.6	76.4	**74.2**	76.7	76.6
0.1 %	72.7	71.1	66.7	**65.1**	71.3	68.4
0.5 %	51.4	48.2	**43.1**	43.4	48.9	46.0
1 %	42.3	37.2	**33.2**	34.6	39.8	35.7
6h runs	$lim100$	$lim2K$	$lim10K$	$lim20K$	$ada2$	$ada4$
0.05 %	73.7	74.3	72.4	**69.9**	72.6	72.5
0.1 %	66.5	67.8	63.9	**61.4**	62.7	62.8
0.5 %	47.8	44.0	**38.6**	39.9	40.1	39.8
1 %	39.0	32.2	**29.4**	30.2	30.3	31.0

31-bit divider

1h runs	$lim100$	$lim2K$	$lim10K$	$lim20K$	$ada2$	$ada4$
0.05 %	62.4	62.5	63.1	**60.5**	62.3	62.3
0.1 %	55.8	56.0	53.3	**51.1**	55.8	55.2
0.5 %	42.5	38.4	**31.6**	29.8	38.3	36.8
1 %	33.9	28.3	**21.6**	22.6	31.5	29.7
6h runs	$lim100$	$lim2K$	$lim10K$	$lim20K$	$ada2$	$ada4$
0.05 %	61.8	62.0	60.5	**58.2**	59.1	59.0
0.1 %	55.1	55.2	51.8	**47.5**	51.2	50.4
0.5 %	37.9	37.2	**27.8**	26.9	30.6	28.2
1 %	30.4	26.0	19.1	**19.0**	22.0	20.3

hours has the area of 54 % of the original 16-bit multiplier. The quality of this solution dominates the solutions obtained by other strategies for this problem setup.

In order to effectively evaluate the overall performance and versatility of the different strategies, we introduce a *versatility score*. For each selected problem setting, we set the versatility score of the strategy B that found the best solution to 100 %, and other strategies S are assigned the score of $area(S)/area(B) * 100$ %. In other words, this measure shows how many per cent larger the solution obtained by the given strategy is with respect to the best solution for the trial (i.e. a lower score is better). As before, we compute the score from the median solutions produced by 50 independent evolutionary runs.

Table 4.7 shows the versatility scores of the inspected strategies computed for the particular circuits considering 1 and 6 hours runs. These scores aggregate the results presented in Tables 4.3–4.6 and give us a better comparison among the strategies. The right-most column of Table 4.7 contains the versatility scores aggregated over all numerical trials. These scores allow us to answer the research question $Q2$, namely, we can compare the versatility of the fixed limit strategies and the selected adaptive strategies.

The best versatility is achieved by the adaptive strategy $ada4$. The score 106.3 shows that a median solution produced by this strategy is on average by about 6 percentage points worse than a median solution produced by the best strategy for a given scenario. Strategy $ada4$ is closely followed by $ada2$ which is roughly by 2 percentage points worse. The best

Table 4.6: Relative sizes in % of the median solutions with respect to the size of the golden solution for square approximation. The left column shows the target WCAE threshold.

20-bit square

1h runs	*lim*100	*lim*2*K*	*lim*10*K*	*lim*20*K*	*ada*2	*ada*4
0.0001 %	**92.7**	97.6	98.8	99.3	94.6	93.4
0.001 %	81.8	93.4	98.5	98.9	85.7	**80.6**
0.01 %	**40.2**	82.6	95.7	97.8	71.4	51.2
0.1 %	29.4	**25.4**	90.3	89.2	25.7	26.9
1 %	13.4	10.5	9.3	**9.0**	13.1	12.2
6h runs	*lim*100	*lim*2*K*	*lim*10*K*	*lim*20*K*	*ada*2	*ada*4
0.0001 %	70.1	82.9	92.2	97.6	70.5	**68.8**
0.001 %	54.6	61.2	94.5	96.0	55.1	**54.3**
0.01 %	38.6	37.4	51.1	81.4	38.0	**36.3**
0.1 %	22.8	22.6	31.4	21.9	21.9	**21.1**
1 %	12.2	7.9	**7.1**	7.2	7.7	7.3

28-bit square

1h runs	*lim*100	*lim*2*K*	*lim*10*K*	*lim*20*K*	*ada*2	*ada*4
0.0001 %	**95.0**	97.0	98.4	98.6	95.8	96.2
0.001 %	90.4	91.0	96.6	97.9	**90.3**	92.3
0.01 %	**50.6**	81.2	96.4	97.2	60.2	62.6
0.1 %	30.2	67.2	95.1	95.8	**19.0**	30.1
1 %	9.0	16.3	9.1	**6.6**	9.9	8.6
6h runs	*lim*100	*lim*2*K*	*lim*10*K*	*lim*20*K*	*ada*2	*ada*4
0.0001 %	**56.1**	77.1	93.0	96.8	67.5	70.4
0.001 %	**31.4**	40.1	81.0	87.9	32.0	32.3
0.01 %	20.6	22.7	73.9	86.4	21.0	**20.5**
0.1 %	12.3	16.0	57.9	43.3	12.2	**12.0**
1 %	6.5	4.6	4.1	**4.0**	4.8	4.2

performance from fixed-limit strategies is provided by *lim*100 that has the versatility score of 114.1.

However, since the final values are computed as averages, the final ranking is skewed by *lim*2*K*'s poor performance for some problem instances in the square circuit approximation (see Table 4.6: 1h runs for 0.01% WCAE). If we excluded these trials from the final evaluation, *lim*2*K* would perform considerably better than *lim*100.

We further perform the Friedman statistical test with Nemenyi post hoc analysis to assess the significance of the results we obtained. In particular, we analyse the statistical significance of the versatility scores for particular approximation strategies across all conducted trials. Friedman test returns `chi-squared` = 71.39 and `p-value` < 5.2E-14. These values clearly demonstrate that the versatility scores for particular strategies are not statistically equivalent. Therefore, we use Nemenyi post hoc analysis to identify the groups of statistically equivalent strategies. Table 4.8 shows the pair-wise p-values for all strategy pairs. It should be noted that these values take into consideration the evaluation over all strategies and conducted trials. Fig. 4.7 illustrates the average ranks (with respect to the versatility scores) of the examined strategies and also visualises the groups that are not sig-

Table 4.7: Overall versatility scores for the considered strategies aggregated over the considered bit-widths for each of the circuits.

	Multiplier		Divider		MAC		Square		
time limit	1h	6h	1h	6h	1h	6h	1h	6h	**AVG**
$lim100$	104.1	106.9	121.7	124.1	109.8	114.4	116.3	115.4	**114.1**
$lim2K$	113.7	101.3	113.6	117.2	102.2	104.5	160.9	117.7	**116.4**
$lim10K$	201.7	118.8	102.6	103.0	126.3	106.0	196.7	206.6	**145.2**
$lim20K$	322.2	136.0	101.2	100.8	151.1	113.2	193.5	208.2	**165.8**
$ada2$	102.5	101.1	116.6	106.4	108.0	102.6	120.2	106.6	**108.0**
$ada4$	100.6	100.5	111.9	104.2	108.8	101.7	118.7	103.6	**106.3**

Table 4.8: Pair-wise `p-values` obtained using Nemenyi post-hoc test evaluated over all strategies and conducted trials.

	lim100	lim2K	lim10K	lim20K	ada2
lim2K	3.90E-14	-	-	-	-
lim10K	0.278	8.50E-11	-	-	-
lim20K	0.021	2.00E-16	2.10E-06	-	-
ada2	2.00E-16	5.30E-14	2.00E-16	2.00E-16	-
ada4	2.00E-16	2.00E-16	2.00E-16	2.00E-16	4.40E-13

nificantly different at $\alpha = 0.05$ We can conclude that strategy $ada4$ is highly significantly better (`p-value` < 0.01) than all examined fixed limit strategies.

The statistical methods are rank based and thus they do not suffer from excessive sensibility to a few trials with major differences in performance. Interestingly, the final placings in Fig. 4.7 (rank based) and Table 4.7 (average based) are identical with the exception of $lim100$. $Lim100$ provides decent solutions for each problem instance, hence scores well in the average based versatility score. On the other hand, it is slightly outperformed in each case by other strategies and so its rank is even worse than that of $lim10K$. Except for a few trials, $lim2K$ places among the top strategies and comes third in the rank based rankings.

Summary for Q2: The adaptive strategies, in contrast to the fixed-limit strategies, are able to provide very good performance for a wide class of approximation problems. This is demonstrated by the highest versatility score as well as by the statistical significance tests.

4.3.6 A Comparison of Adaptive and Fixed-limit Strategies (Q3)

We saw that the adaptive strategies provide the best versatility score as well as rank score which indicates that they can effectively handle various approximation scenarios. In this section, we look closer at the results presented in Tables 4.3–4.6 and focus on interesting data points revealing the weak and strong properties of the adaptive strategies. In particular, we will discuss if a single adaptive strategy can outperform the best fixed-limit strategy for a given circuit approximation problem.

Table 4.3 shows that the adaptive strategies dominate in almost all approximation scenarios for multipliers. In two scenarios, the strategy $lim2K$ slightly outperforms the adap-

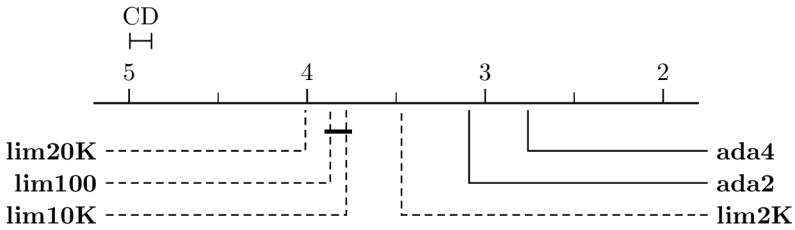


Figure 4.7: The average rank values and pair-wise comparison of all strategies obtained by Nemenyi test. Groups of strategies that are not significantly different (at `p-value` = 0.05) are connected.

tive strategies, however, it significantly lags behind for 1 hour runs and selected WCAEs (i.e. 24-bit version and 0.001 % WCAE).

On the other hand, the adaptive strategies lag behind the best strategies mainly in two sets of trials: MACs in 1 hour evolution and dividers in 1 hour evolution (see Tables 4.4). Their performance is similar to that of $lim100$, and they are outperformed by strategies with higher limit values. Since the adaptive strategies are designed to keep the limit as low as possible while still achieving some improvements in the candidate solutions, they do not increase their limit value during the first hour of the numerical evaluation. Our numerical trials show that even with a low resource limit, the strategies find improvements, but many of the candidate solutions are rejected because they cannot be evaluated within the limit. The difference in performance is reduced as the optimisation process continues and the adaptive strategies increase their resource limit. After 6 hours, the adaptive strategies outperform the other strategies for MACs and come close to the performance of $lim10K$ and $lim20K$ for dividers.

In case of the square circuit approximation, the adaptive strategies always produce a solution that is either the best or close to the best solution found. The exceptions are 1-hour runs for 0.01% WCAE, and 6-hour run for 28-bit version and 0.0001% WCAE, where $lim100$ significantly outperforms the other strategies.

Summary for Q3: The adaptive strategies provide the best performance (or are very close) for a wide class of approximation problems except for MACs with short approximation time where low-limit strategies are slightly better due to faster convergence, and for dividers where high-limit strategies are better, due to the initial phase of the adaptive strategies.

4.3.7 Comparison with State-of-the-art Techniques (Q4)

In this section, we demonstrate that our adaptive approach generates approximate circuits that significantly outperform circuits obtained using the state-of-the-art approximation techniques. In particular, we show that our circuits provide significantly better trade-offs between the precision and energy consumption. We focus on multipliers since their approximation represents a challenging and widely studied problem – see, e.g. the comparative study of [49]. On the other hand, the existing literature does not offer a sufficient number of high-quality approximate MACs or dividers to carry out a fair comparison.

In the comparison, we consider two approximate architectures for multipliers that are known to provide the best results, namely *truncated multipliers* (TMs) that ignore the values of least significant bits and *broken-array multipliers* (BAMs) [32]. TMs and BAMs can be parameterised to produce approximate circuits for the given bit-width and the required

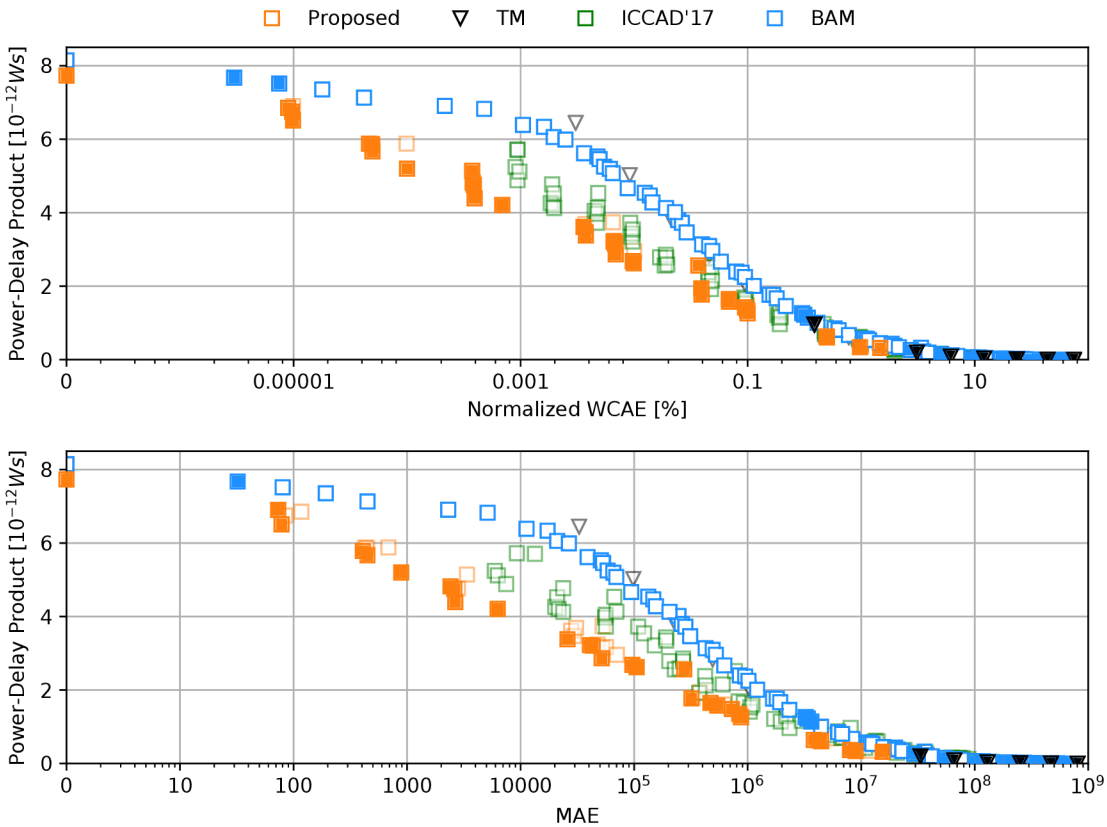


Figure 4.8: A comparison of the 16-bit approximate multipliers obtained using the proposed approach and other state-of-the-art approximation techniques. The plots show the solutions and their trade-offs between the precision and the power-delay-product (PDP) – the top plot depicts WCAE while the bottom plot depicts the mean absolute error (MAE). The filled marks represent solutions providing the best PDP for the given precision.

error. In contrast to our search-based approach, these circuits are constructed using a simple deterministic procedure based on simplifying accurate multipliers. However, the method is applicable for the design of approximate multipliers only. To demonstrate the practical impact of the proposed adaptive strategy, we also consider circuits presented in Chapter 3 obtained using verifiability-driven approximation with a fixed limit strategy – this is a prominent representative of the search-based strategies.

Fig. 4.8 shows the parameters of the resulting circuits belonging to the Pareto front. For each circuit, the figure illustrates the trade-off between the precision and the power-delay-product (PDP) that adequately captures both the circuit's energy consumption and its delay. The top plot of the figure illustrates the WCAE–PDP trade-offs. We also evaluated the mean absolute error (MAE) [20] of the solutions since MAE represents another important circuit error metric. The results are presented in the bottom plot of the figure.

The orange boxes represent circuits obtained using the adaptive strategy $ada4$. The green boxes represent the circuits that were presented in Chapter 3 and obtained using the fixed limit strategy $lim20K$. In both cases, the circuits were generated as follows: we selected 15 target values of WCAE (10 values for the strategy $lim20K$) and for each of these values, we executed 50 independent 2-hour runs using $\lambda = 1$ and the mutation frequency 0.5 %. The 10 best solutions for each WCAE were selected and synthesised to

the target technology. It should be noted that the strategy $lim20K$ provides a much smaller reduction of the chip area when very small values of WCAE are required and thus these small target values were not reported in Chapter 3.

As we have already shown in the previous chapter, the fixed-limit verifiability-driven approach leveraging SAT-based circuit evaluation is able to significantly outperform both TMs and BAMs and represents a state-of-the-art approximation method for arithmetic circuits. Still, Fig. 4.8 shows that the proposed adaptive strategy improves our previously obtained results even further – given the same time limit, it generates circuits having significantly better characteristics.

Summary for Q4: The proposed approach combining the SAT-based candidate evaluation with the adaptive verifiability-driven search strategy provides a fundamental improvement of the performance and versatility over existing circuit approximation techniques.

4.4 Conclusion

In this chapter, we presented the novel adaptive strategy, that extends and improves the idea of the verifiability driven search introduced in Chapter 3. In particular, the adaptive strategy improves the overall performance, and, more importantly, it ensures that our approach is versatile, i.e. in contrast to the method described in the previous chapter, it works well for a wide range of arithmetic circuits and approximation scenarios without a manual tuning of the parameters of the evolutionary algorithm. We also thoroughly examined the impact of various resource limit settings on the progress of the evolutionary algorithm in a wide range of approximation scenarios. The extensive experimental evaluation showed, that the adaptive strategy significantly outperforms the verifiability driven search with fixed resource limit settings.

Chapter 5

ADAC - a Framework for Automatic Approximation

In this chapter, we present ADAC – a framework for automated design of approximate circuits. The framework contains implementations of the methods and algorithms proposed in Chapters 3 and 4 and serves as the tool support for our experimental evaluations. The contents of this chapter are based on the tool paper [14] published at the 30th International Conference on Computer Aided Verification (CAV 2018).

The ADAC framework implements a design loop including (i) a *generator* of candidate solutions employing genetic search algorithms, (ii) an *evaluator* estimating non-functional parameters of a candidate solution, and (iii) a *verifier* checking that the candidate solution does not exceed the permissible error. ADAC is integrated as a new module into the ABC tool – a state-of-the-art and widely used system for circuit synthesis and verification [7]. The framework takes as the inputs:

- a golden combinational circuit in Verilog implementing the correct functionality,
- an error metric (such as worst-case absolute error, mean absolute error, etc.),
- a threshold on the error metric representing the maximal permissible error,
- a time limit on the overall design process, and
- a file specifying sizes of gates available to the design process.

With these inputs, ADAC searches for an approximate circuit satisfying the error threshold and having the minimal estimated chip area. Previous works [13, 78, 113, 114] confirmed that the chip area is a good optimisation objective as it highly correlates with power consumption, which is a crucial target in approximate computing.

The results of [119] clearly demonstrate that search algorithms based on Cartesian Genetic Programming (CGP) [71] are well capable of generating high-quality approximate circuits. For complex circuits, however, a high number of candidate solutions has to be generated and evaluated, which significantly limits the scalability of the design process. Our framework implements several approaches for error evaluation suitable for different error metrics and application domains. They include both SAT and BDD-based techniques for approximate equivalence checking providing formal error guarantees as well as a bit-parallel circuit simulation utilising the computing power of modern processors. The performance

of the approximation with SAT-based error evaluation is further improved by the adaptive verifiability driven search scheme presented in Chapter 4.

As such, the framework offers a unique integration of techniques based on simulation, formal reasoning, and evolutionary circuit optimisation. Our extensive experimental evaluation presented in Chapter 4 demonstrates that ADAC offers an outstanding performance and scalability compared with the existing methods and tools and paves a way towards an automated design process of complex provably-correct circuit approximations.

5.1 Architecture and Implementation

The ADAC framework has a modular architecture illustrated in Fig. 5.1.

Before the approximation process can begin, there is a setup phase which is responsible mainly for converting the golden circuit into a chromosome representation that can be optimised using CGP. The input circuit is given in a high-level Verilog format, which is first translated to a gate-level representation using the tool Yosys [129], and then the chromosome representation is obtained using our built-in Verilog to chromosome converter. The setup phase is also responsible for generating a configuration file controlling the main design loop. It is generated from the user inputs and optional parameters that can further fine-tune CGP and the search strategies.

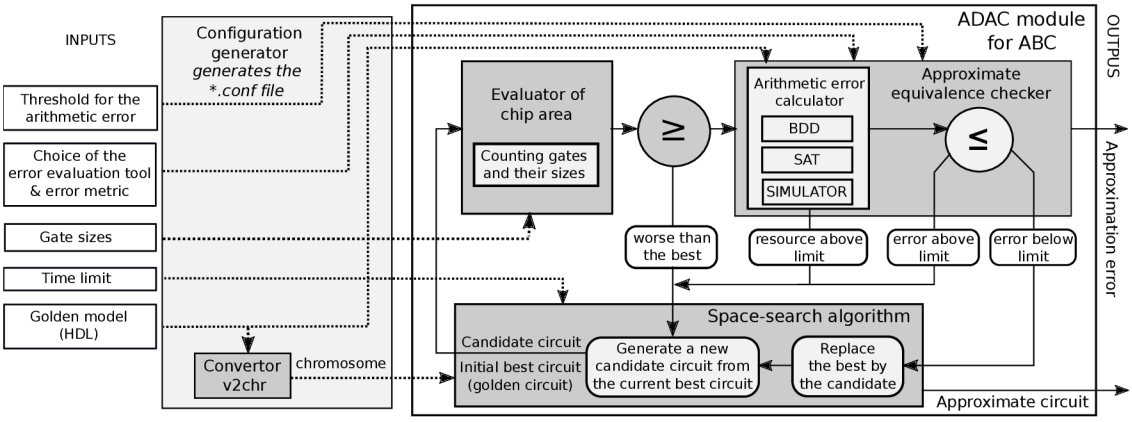


Figure 5.1: A scheme of the ADAC architecture.

The design loop consists of three components: (i) a generator of candidate designs, (ii) an evaluator of non-functional parameters of the candidate circuit (in our case estimating the chip area), and (iii) a verifier evaluating the candidate error. Optionally, the adaptive resource limit strategy can be utilised when using SAT based error verifier.

The chip area and the error form a basis of the *fitness function*, whose value is minimised via our search strategy. In particular, the fitness is infinity if the circuit error exceeds the given threshold, and the chip area otherwise. As an additional feature, ADAC can also utilise the error verifier to quantify the difference (in the given metric) between two given circuits.

The real values of non-functional parameters, such as the chip area or the power-delay product (PDP), depend on the target technology, and the synthesis of an optimal implementation of the given circuit using the target technology is highly time-consuming. Therefore, our design loop uses the chip area as the sole non-functional parameter. The chip area is estimated as the sum of the sizes of the gates of the circuit, which are given as one of the

inputs of ADAC. The output of ADAC (in the gate-level Verilog format) can be passed to industrial circuit design tools to obtain accurate circuit parameters for the target technology. In our experiments, we report PDP for the 45 nm technology synthesised by the Synopsys Design Compiler [109].

5.1.1 Integration to the ABC Tool.

The ABC tool allows us to support an important subset of the Verilog specification and implementation language. We also utilise ABC to translate the circuits among different intermediate representations used for constructing miters. We employ the `iprove` engine in our SAT-based method for evaluating the WCAE and WCRE. Note that `iprove` uses MiniSat [106] as the SAT solver. Despite the fact that ABC supports a BDD-based circuit representation and manipulation, we implemented our own BDD component (based on the BuDDy library [11]) that is tailored for evolutionary circuit approximation.

5.2 Error Evaluation Methods

We now briefly describe the three methods for error evaluation that are supported in ADAC.

5.2.1 Bit-parallel Circuit Simulation

The bit-parallel circuit simulation supports all common error metrics, including the worst-case absolute error (WCAE), the mean absolute error (MAE), the error rate representing the number of inputs leading to an incorrect output, and the Hamming distance. It utilises the power of modern processors by simulating the circuit on multiple input vectors (e.g. 64 inputs for 64-bit processors) in a single pass through the circuit [121]. Even though the parallel processing significantly accelerates the simulation, the error evaluation still becomes slow for circuits with arguments of larger bit-widths (beyond 12 bits). When such circuits are simulated on all possible input combinations, the performance of the approximation process declines, and it is unable to find good solutions in acceptable time bounds. To alleviate this issue, we can utilise the partial simulation, i.e., evaluate the error on a subset of input vectors to speed up the evaluation. However, partial simulation only provides statistical guarantees on the approximation error.

5.2.2 BDD-based Evaluation

The BDD-based evaluation also supports all common error metrics, and, unlike simulation, it is able to provide formal error guarantees for circuits with larger input bit-widths. For the purpose of the evaluation, the original correct circuit and its approximation are interconnected into an auxiliary circuit called a miter such that the error can be deduced from its output (e.g. to compute the error rate, the outputs of the golden and candidate circuits are subtracted, and the result is compared with 0). The miter is encoded as a BDD on which the circuit error is evaluated using efficient BDD operations [114, 120] (e.g. the error rate is evaluated via counting the number of satisfying solutions). On the other hand, this technique does not scale well with the complexity of the circuits in terms of the number of their gates as the resulting BDD representation becomes prohibitively huge. Hence, this approach works well for large adders and similar circuits, but, it fails, e.g., for multipliers beyond 12-bits.

5.2.3 SAT-based Evaluation

The SAT-based evaluation supports only two error metrics – WCAE and WCRE, but provides formal guarantees and a superior performance to the BDD-based technique. ADAC implements a novel miter construction based on subtracting the output of the golden and approximate circuit, followed by a comparison with the error threshold (as described in Chapter 3). The construction is optimised for SAT-based evaluation by avoiding long XOR chains known to cause poor performance of state-of-the-art SAT solvers [20, 41]. This allows us to exploit the ABC engine `iprove`, designed originally for miter-based exact circuit equivalence checking, to quickly evaluate the approximation error.

The SAT-based evaluation is further powered by the verifiability driven search with adaptive resource limits. The strategy uses a limit L on the resources available to the underlying SAT decision procedure. The limit effectively controls the time the SAT solver can use. We require that every improving candidate has to be verifiable using the resource limit L. Therefore the strategy drives the search towards candidates that improve the fitness and can be promptly evaluated. As the result, we can evaluate a much larger set of candidate circuits in the given time. As shown in the previous chapter, approximation of various problem instances can require various resource limits. This is solved by the adaptive resource limit strategy that is able to adjust the limit and provide good performance in a variety of approximation scenarios.

Table 5.1: A comparison of the average times of simulation, BDD (WCAE), BDD (MAE) and SAT (WCAE) error evaluation in approximation of adders with bit widths ranging from 4 to 16 bits. The best time for evaluation of WCAE and MAE for each bit width is highlighted.

Adders	w = 4	w = 6	w = 8	w = 10	w = 12	w = 14	w = 16
Simulation	**44** μs	**210** μs	36 ms	76 ms	1.46 s	31.23 s	—
BDD e_{wcae}	83 μs	350 μs	2.1 ms	12 ms	86 ms	0.38 s	2.09 s
speedup	0.53×	0.59×	1.79×	6.04×	17.02×	80.74×	—
BDD e_{mae}	74 μs	370 μs	**2.9** ms	**13** ms	**0.16** s	**0.79** s	**2.47** s
speedup	0.60×	0.59×	1.27×	5.72×	8.81×	38.94×	—
SAT e_{wcae}	630 μs	920 μs	**1.1** ms	**1.4** ms	**1.6** ms	**1.7** ms	**2.2** ms
speedup	0.07×	0.23×	3.24×	53.7×	941×	18468×	—

5.3 Error Evaluation Performance

In this section, we compare the performance of the different methods of circuit error evaluation supported in ADAC. For that, we use the results from adder approximation obtained from 10 runs, each for 5 minutes. Table 5.1 shows average run times of a single error evaluation using the bit-parallel simulation, the BDD-based miter solution, and the SAT-based miter solution. The reported speedups are computed with respect to the simulation-based evaluation. We can see that the simulation provides the best performance for small bit-widths, but does not scale well for larger bit widths and thus can provide only statisti-

cal guarantees for larger circuits. The SAT-based method offers the best scalability and dominates for larger circuits, but supports the WCAE and WCRE evaluation only. The BDD-based method, like simulation, supports all metrics, including the mean absolute error (MAE), and significantly outperforms the simulation for larger circuits. Note that, for more complex circuits such as multipliers, we would observe similar results with a worse relative performance of the BDD-based approach.

There also exist other known methods for computing approximation errors for arithmetic circuits, including methods based on BDDs [21] or SAT-based miter solutions [20]. Comparing to ADAC, these methods are less scalable, which is demonstrated by the fact that they have been used for approximating multipliers limited to 8-bit operands and adders limited to 16-bit operands only. Apart from that, there are efficient methods for *exact* equivalence checking based on algebraic computations [24, 92]. However, they are so far not known for approximate equivalence checking.

Chapter 6

StS-based Synthesis for Exact and Approximate Circuits

The contents of this chapter are based on our work Satisfiability Solving Meets Evolutionary Optimisation in Designing Approximate Circuits published in [16]. In this part of the thesis, we examine the satisfiability-based exact circuit synthesis for the design of approximate circuits and combine it with evolutionary circuit approximation. The exact optimisation rewrites parts of the approximated circuit and helps the evolutionary algorithm to escape local optimums, thus allowing further improvements of the quality of the final approximate solutions.

6.1 Motivation

Previously, we have defined two main challenges related to the evolutionary-driven circuit approximation: (1) improvements of the analyser (providing a fast and reliable evaluation of candidate solutions) and (2) improvements of the synthesizer (generating approximate circuits that quickly converge to a high-quality solution).

In this chapter, we aim at the second challenge. Inspired by the recent advances in SAT-based exact synthesis [40, 105] (the problem of finding the optimum logic representation of a given Boolean function), we investigate whether a search strategy based on *satisfiability solving* (StS) – i.e. SAT or SMT solving – can improve the state-of-the-art methods for designing complex approximate circuits.

Complex circuits typically have hundreds or even thousands of gates and thus a monolithic approach, i.e. representing the circuit approximation problem as a single StS query, is not tractable. Instead, we build on an iterative approach where sub-circuits are optimised (i.e. the sub-circuit logic is minimised while the original functionality is preserved) [103] or approximated (i.e. the functionality of the sub-circuit is not preserved and the error of the whole circuit is increased).

In the iterative optimisation, the sub-circuit is replaced by a (potentially) smaller sub-circuit implementing the equivalent logic and thus the optimisation preserves the overall circuit functionality. In the iterative approximation, the sub-circuit is replaced by a smaller sub-circuit approximating the logic and thus the error of the whole circuit is typically introduced or increased.

Despite the enormous progress in satisfiability solving, our experiments clearly show that the purely StS-based approximation significantly lags behind the standard evolutionary

approximation. Although the StS-based approximation performs informed (and thus in some sense more useful) changes in the candidate circuits, the overhead caused by calling the solver does not pay off compared to the uninformed but very cheap genetic mutations. Put differently, in the same time interval, the evolution can perform over 100-times more approximation attempts which is enough to overcome the benefit of the informed changes.

In order to leverage the benefits of the informed changes, we propose a *combined approach*. We interleave the evolutionary approximation and the StS-based optimisation. The evolutionary approximation typically quickly converges to a sub-optimal solution. After the progress of the evolution decreases below a certain threshold, we run the StS-based optimisation. It further reduces the circuit size, but, more importantly, it introduces new reconnections in the circuit causing that the subsequent evolution is able to escape a local minimum and further explore the design space. Our experimental results show that the combined approach considerably outperforms the state-of-the-art circuit approximation techniques. In particular, for a given error, it improves the chip area by up to 19 %.

6.2 StS-based Circuit Approximation

In this section, we propose three different approaches for StS-based circuit approximation.

6.2.1 A Monolithic Approach

The monolithic approach builds a single formula encoding the following synthesis problem: *For a given golden circuit GC, the size S of its currently best-known approximation, and an error bound T, synthesise an approximating circuit AC whose size is smaller than S and that satisfies the constraint that $error(GC, AC) \leq T$.*

The formula has to encode the following features: (1) the possible designs of the circuit (i.e. possible interconnections and functionality of the gates), which must be encoded using free variables whose suitable values are to be found by the solver, thus fixing a certain design of the circuit; (2) the way the error of the circuit is to be checked; and (3) the way the circuit size is to be evaluated.

In our approach, we use the circuit representation that also serves as the *chromosome* encoding in Cartesian Genetic Programming. The designed circuit is encoded as a forward-propagating network of two-input gates. Each gate is represented by three integers. The first two represent the inputs of the gate, and they can refer to some of the primary inputs or to the output of one of the gates (which we identify with the gate itself). The third integer then encodes the gate's functionality that is chosen from a predefined set of operations. The gate representation of each operation has a predefined size given by the target chip architecture. To ensure that the size of the synthesised circuit AC is smaller than the size S of the currently best approximation, we add a constraint on the sum of the sizes of the gates forming AC. We investigate and compare (see Section 6.3) the below presented three ways of encoding the structure and functionality of AC.

The first encoding is purely *SAT-based* although we present it using both Boolean and integer variables – those are, however, bit-blasted away. Assume we have k types of (binary) gates, use l gates in the circuit, and have m/n primary input/output bits, respectively. For each gate $g \in G = \{1, ..., l\}$, we use the integer variables $in_{g,1}$ and $in_{g,2}$ to denote the first and second input of g. These variables range over the domain $W = \{1, ..., m + l\}$ of all wires in the circuit where the first m wires carry the primary inputs and the next l wires carry the outputs of the circuit's gates. For $g \in G$, we also use the integer variable f_g to

denote its type with the domain $F = \{1, ..., k\}$. Let $\mathbb{I} = \{0, 1\}^m$ denote the different input combinations. For $u \in W$, we use the Boolean variable b_u^I to hold the value of the wire u for a primary input $I \in \mathbb{I}$. We encode all possible circuits by the conjunction of the formulae $(in_{g,1} = u \wedge in_{g,2} = v \wedge f_g = f) \rightarrow \bigwedge_{I \in \mathbb{I}}(b_{m+g}^I = b_u^I \; op_f \; b_v^I)$ that are generated for all gates $g \in G$, all possible types $f \in F$ of g, and all wires $u, v \in W$ that may be used as the inputs of g. In particular, we require that $u < g$ and $v < g$ to prevent backward connections in the circuit (e.g. the input of g_4 cannot be connected to the output of g_6). In the formula, op_f denotes the Boolean operation implemented by gates of the type $f \in F$.

We also need to link the concrete input combinations with the input wires, which is done by the conjunction $\bigwedge_{I \in \mathbb{I}} \bigwedge_{j \in \{1,...,m\}} b_j^I = I[j]$ where $I[j]$ denotes the j-th bit of I. Finally, for each output $o \in O = \{1, ..., n\}$, we introduce the integer variable out_o, which ranges over the domain of wires W and says from where the output o is taken, and the Boolean variable out_o^I carrying the value of the output o for the primary input $I \in \mathbb{I}$ (this variable will be compared with the appropriate output of the golden circuit). These variables are connected with the rest of the circuit using the conjunction of the formulae $out_o = u \rightarrow \bigwedge_{I \in \mathbb{I}} out_o^I = b_u^I$ generated for every output $o \in O$ and every wire $u \in W$. The solver then chooses a concrete circuit by fixing the values of the variables $in_{g,1}$, $in_{g,2}$, and f_g for every $g \in G$ as well as the values of the variables out_o for every $o \in O$.

Second, using a *theory of arrays*, we simplify the above encoding by using an array $\overline{b}^I : W \rightarrow \{0, 1\}$ for each $I \in \mathbb{I}$ to hold the values of the wires in W for the input I. Then, the conjuncts describing the structure of the circuit may be simplified to $f_g = f \rightarrow \bigwedge_{I \in \mathbb{I}}(\overline{b}^I[m + g] = \overline{b}^I[in_{g,1}] \; op_f \; \overline{b}^I[in_{g,2}])$. The input formula is changed to $\bigwedge_{I \in \mathbb{I}} \bigwedge_{j \in \{1,...,m\}} \overline{b}^I[j] = I[j]$ and similarly for the output. Finally, third, using a *theory of arrays with quantifiers*, one suffices with a single array $\overline{b}$, simplifying the formulae describing the structure of the circuit to $f_g = f \rightarrow \overline{b}[m + g] = \overline{b}[in_{g,1}] \; op_f \; \overline{b}[in_{g,2}])$, adding the universal quantification $\forall i_1, ..., i_m$ over the entire formula, using the input formula $\bigwedge_{j \in \{1,...,m\}} \overline{b}[j] = i_j$, and handling the output accordingly.

Using our encodings of AC, we can easily add a constraint on the required error that compares the WCAE between the result coming from AC (using the out_o variables) and the expected result for all input combinations.

A comparison to existing encoding schemes for exact synthesis. Our encoding of circuits is quite similar to other works such as [105]. The authors of [105] do not consider a predefined set of gates. Instead, they synthesise the internal functionality of the gates (e.g. in the form of look-up-tables) too. The work [105] and other existing approaches use SAT-based encodings only. Further, they consider uniform gate sizes only, and thus the circuit size is given by the number of gates. Our more general formulation using non-uniform gate sizes leads to more complex problems (proving the minimality of a given circuit requires more difficult UNSAT queries). As in [102], we use simplifications and symmetry-pruning techniques to reduce the complexity of the underlying StS queries.

6.2.2 Sub-circuit Approximation

As discussed in [40], the monolithic approach for exact synthesis is feasible only for small circuits with up to 8 input bits (depending on the complexity of the synthesised function). Our experiments confirm similar scalability limits also for circuit approximation (see Section 6.3), and thus we focus on an iterative approach that approximates selected sub-

circuits. We focus on approximation wrt. Hamming Distance as arithmetic metrics are not suitable for sub-circuits. Note that there is no effective method allowing us to determine how the error introduced in the sub-circuit affects the overall circuit error.

In every iteration, we select a single gate (either randomly or by enumeration, depending on the circuit size) and perform a breadth-first search starting from the selected gate to identify a sub-circuit of a suitable size. Note that, in our approach, we consider multi-input and multi-output sub-circuits. The size of the sub-circuits is indeed essential: Considering only very small sub-circuits prevents the approximation from doing more complicated and non-local changes that are crucial for finding high-quality approximate circuits. On the other hand, approximation of larger sub-circuits introduces a significant overhead causing that only a small number of iterations can be done within the given time limit. Regarding the encoding of sub-circuit approximation, we consider the same schemes as in the monolithic approach discussed above. After every sub-circuit approximation, we need to evaluate the error of the whole circuit. If it satisfies the error bound, we accept the circuit as the new candidate solution, otherwise the next iteration continues with the circuit before the approximation.

6.2.3 Evolutionary Approximation with StS-based Optimisation

Evolutionary algorithms, in particular Cartesian Genetic Programming (CGP), have achieved excellent results in approximation of complex circuits [13]. The key idea is similar to sub-circuit approximation, but here CGP performs random changes in the candidate solution instead of utilising satisfiability solving. Unlike finding an optimal sub-circuit approximation, random changes are very fast, and the success of CGP is typically achieved by a large number of small changes. CGP is also able to accumulate a large change in the candidate circuit via the so-called *inactive mutations* [71] – a chain of changes where only the last change directly affects the circuit functionality. Although CGP usually quickly converges to a sub-optimum solution, it can get stuck in this solution for a long time. On the other hand, the StS-based approach is able to systematically search for improvements that are hard to find for CGP.

We hence propose a *combined approach* leveraging the benefits of both techniques. In particular, we interleave the evolutionary search by iterative StS optimisation. In contrast to StS-based approximation, StS-based optimisation minimises the size of the selected sub-circuit by changing the internal structure while preserving its functionality. The rationale behind this is based on the observation that a large portion of approximated sub-circuits are rejected as they cause that the WCAE error of the whole circuit gets above the allowed bound. Compared with CGP, the cost of each approximation operation is too high – in our scenarios, CGP is about 100-times faster. Therefore, the combined approach uses CGP to introduce changes affecting the functionality, and the StS-based optimisation to minimise the logic. The minimisation often leads to an improving solution. Further, we also explore different encoding schemes (based on the ideas discussed in Section 6.2.1) for the optimisation problem.

The interleaving is controlled in the following way. If CGP gets stuck in a local optimum – no improvement of the candidate solution was achieved for a given number of iterations – we switch to the iterative StS optimisation that has a time budget depending on the given overall time for the approximation. Once the budget is spent, we continue with CGP again. Our experiments show that the optimisation helps CGP to escape the local optimum and to further effectively explore the space of candidate circuits.

6.3 Experimental Evaluation

We ran all our experiments on a server with an Intel(R) Xeon(R) CPU at 2.40GHz. Although the search-based approximation can naturally benefit from a simple task parallelisation, we use a single-core computation to simplify the interpretation of the results.

A Comparison of different encoding schemes and satisfiability solvers. We consider a set of formulae relevant for the monolithic as well as for the iterative approach. The set includes both SAT and UNSAT instances including a full adder, 2-bit adder, 2-bit multiplier, 4-1 multiplexor, and some randomly generated 4-input functions, for a total of 48 different formulae. We compare the total time needed to solve all the formulae with a 3 hour time limit. We do not include any additional penalty for timing out. The Z3 solver [75] and the quantified array encoding proved to be the fastest combination of the encoding and the solver. Z3 with separate arrays for different input combinations is about 2 times slower, and the Glucose solver [4] with the purely SAT-based encoding is about 3 times slower. Z3 with the purely SAT-based encoding as well as its SMT variant without bit-blasting were roughly 5 times slower. Other tested solvers – MathSAT [9], Minisat [29], Sadical [45], and Vampire [88]—were all more than 5 times slower. The precise values for each examined solver and method are presented in Table 6.1. Based on these observations, we use Z3 with the quantified array encoding in all further StS-based queries.

Table 6.1: A comparison of the performance of different encoding schemes and satisfiability solvers. We report the total time needed to solve all 48 formulae, along with the number of formulae that timed out (TO) – were not solved within the 3 hour time limit. We do not include any penalty for timing out.

Rank	Method	Number of TOs	Total time [s]	Relative to best
1	Z3-quant	0	4633	—
2	Z3-arr	0	8839	1.91×
3	Glucose	1	15656	3.38×
4	Z3-sat	1	21028	4.54×
5	Z3-smt	2	23041	4.97×
6	MathSAT	1	23235	5.01×
7	Minisat	2	23642	5.10×
8	Vampire	13	141386	30.51×
9	Vampire-arr	19	213713	46.12×
10	Sadical	21	230050	49.65×

The monolithic approach. The monolithic approach was able to find optimal approximations of 2-bit adders and multipliers as well as randomly generated functions with 4 inputs and 2 outputs. Approximation of larger circuits (e.g. 4-bit adders and multipliers) proved to be infeasible, i.e. most instances timed out within the given limit of 3 hours.

We compare the performance of our monolithic approach with Cirkit [101], a state-of-the-art tool for exact synthesis. As expected, Cirkit is able to achieve a better performance and scalability: It is significantly faster on 4-bit functions and it can also synthesize in

minutes optimal solutions for some 6-bit and 8-bit functions. On the other hand, there are also some hard 6-bit instances that are infeasible for Cirkit.

The better performance of Cirkit is mainly caused by the following factors: (1) Our formulation of circuit approximation is more complicated due to the non-uniform gate sizes and the error quantification. (2) Cirkit uses different circuit representations (such as AIGs, MIGs, or n-bit look-up tables) that proved to be more efficient for some exact synthesis problems [102]. (3) Cirkit implements various optimisations and symmetry breaking methods [40]. Some of these methods are problem- and representation-specific and thus not directly applicable to our approximation problem. We are, however, aware that our current prototype implementation could be improved by adapting some of the methods. We emphasise that such improvements would indeed not change the practical limits of the monolithic approach (arithmetic circuits beyond 4-bit operands).

In the following subsections, we will examine and compare three strategies for approximation of complex circuits: (1) CGP: the state-of-the-art evolutionary approximation implemented in our tool ADAC (Chapter 5). (2) StS: the sub-circuit approximation described in Section 6.2.2. (3) COMB: the combined approach described in Section 6.2.3 using the following interleaving strategy: In every iteration, we first run the CGP-based approximation until no improvement is achieved for 100K iterations. Afterwards, we switch to the StS-based sub-circuit optimisation with the time budget of 10 % of the overall approximation time. After that, a new iteration starts.

Based on our preliminary experiments, we use sub-circuits with 5 gates (recall the discussion in Section 6.2.2) in all StS-based sub-circuit approximation and optimisation queries. We also introduce a hard time limit (depending on the size of the circuit) on every such query. It prevents the solver to spend a prohibitively long time in complex queries and thus to significantly slow down the approximation process.

6.3.1 Performance on Small Circuits

We first consider small circuits (a 4-bit multiplier with 67 gates and an 8-bit adder with 49 gates) to understand performance aspects of the search strategies. We report the area savings (as the percentage of the size of the golden circuit) for selected WCAE error bounds and the approximation time limit of 1 hour.

Table 6.2 (top) shows the results obtained from 15 independent approximation runs for each combination of the approximation method, circuit, and target error. For the 8-bit adder, the combined approach wins in all 45 evolutionary runs. On average, the combined approach saves 7.6 % more than the pure CGP and 29 % more than the pure StS-based approach. For the 4-bit multiplier, the combined strategy provides on average 3.27 % better savings than the pure CGP and 19.6 % more than the pure StS-based approach. It also wins 37 out of 45 comparisons.

These experiments show that the pure StS approximation is not competitive, and it is not considered in the following approximation of complex circuits.

6.3.2 Performance on Complex Circuits

In this subsection, we focus on our key research question: *Can the combined strategy improve the performance of the approximation of complex circuits?*

We consider approximation of (1) a 32-bit adder representing a circuit with a simpler logic (the golden model has 235 gates) but a large input domain, and (2) a 16-bit multiplier representing a circuit with a complex logic including carry chains (the golden model

Table 6.2: The average resulting size of the approximate circuits, obtained using the proposed approximation strategies, expressed as the percentage of the size of the golden circuits (top) and of the size of the best known approximations presented in Chapter 3 (bottom).

	8-bit adders			4-bit multipliers		
Err[%]	CGP	StS	COMB	CGP	StS	COMB
1	64.8	83.5	**54.5**	78.4	90.5	**74.6**
2	52.6	78.0	**44.9**	69.3	82.6	**67.1**
5	37.1	57.4	**32.3**	53.4	77.0	**49.7**

32-bit adders			16-bit multipliers		
Err[%]	CGP	COMB	Err[%]	CGP	COMB
10^{-5}	100.0	**81.5**	10^{-3}	97.9	**91.4**
10^{-4}	100.0	**81.3**	0.01	97.6	**91.1**
10^{-3}	100.0	**81.1**	0.1	95.0	**90.1**

has 1,525 gates) but a smaller input domain. To evaluate the potential of the combined strategy, we start with state-of-art approximate circuits we obtained in Chapter 3 by a pure evolutionary search strategy. For each target error, we take two best 32-bit adders (obtained by 2-hour approximation runs) and two best 16-multipliers (obtained by 8-hour approximation runs). From each of these circuits (seeds), we perform 2 pure CGP runs and 2 combined strategy runs. For the 32-bit adders, we consider 10 hour runs; and, for 16-bit multipliers, we consider 75 hour runs. Table 6.2 (bottom) shows the results. The following paragraphs summarise the key observations.

32-bit adders. Each pure CGP run performs around 10 million iterations within the given 10 hours but achieves no improvements at all. This demonstrates that the CGP reached a local optimum and it is not able to further explore the design space. The sub-circuit optimisation, however, introduces changes in the circuit structure, which allow CGP to escape the local optimum and perform further improvements. In total, the combined strategy saves roughly 19 % of the seeding circuit area – 11 % was achieved by the CGP approximation and 8 % by the StS optimisation. This supports our claim that the StS optimisation successfully helps CGP to leave the local optima reached during the approximation process.

16-bit multipliers. As illustrated in Fig. 6.1, which shows the progress of the two approximation strategies for different target errors, the pure CGP approximation improves the candidate slowly and achieves only marginal improvements after 45 hours. The combined strategy is able to significantly improve the candidate solution during the whole 75-hour run—after this time, it saves 4–6 % more than the pure CGP. Compared to 32-bit adders, the approximation of the 16-bit multipliers is significantly more complex. The 8-hour CGP run computing the seed performs around 230K iterations, which is around 13-times less than the 2-hour run for the 32-bit adder. Hence, the pure CGP run requires a much longer time to reach the local optimum. We can see that the combined strategy would most likely keep improving the solution even after the 75 hours.

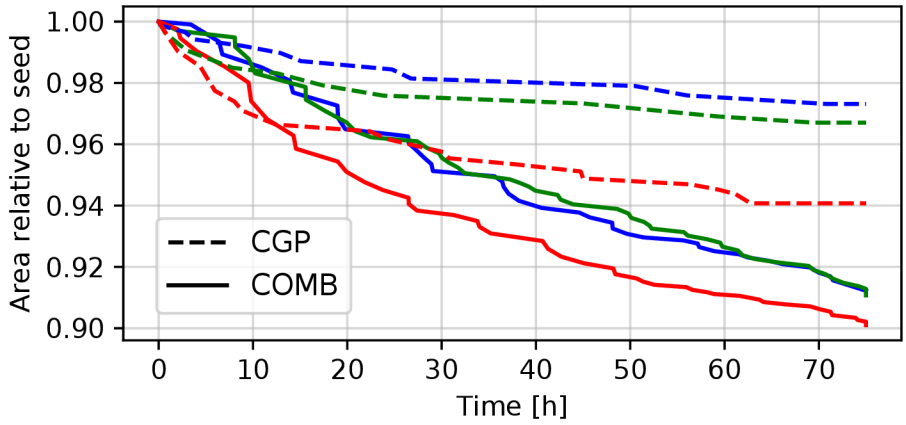


Figure 6.1: Progress of the area reduction for the 16-bit multiplier and target WCAEs: red = 10^{-1}%, green = 10^{-2}%, blue = 10^{-3}%. The y axis shows the ratio of the area of candidate solution to the area of the seeding solution.

6.4 Conclusion

In this chapter, we examined the possible applications of StS-based exact synthesis in the area of approximate circuit design. We proposed three different techniques: 1) the monolithic approach, 2) the StS-based sub-circuit approximation, and 3) the combination of evolutionary approximation and StS-based optimisation.

The monolithic synthesis approach is only applicable to small circuit instances and does not scale for synthesis and approximation of relevant arithmetic circuits. The StS-based sub-circuit approximation is able to introduce error into the candidate solution and gradually reduce its area. However, the speed of the area reduction significantly lags behind the classic evolutionary based approximation. The proposed fusion of StS-based sub-circuit optimisation and evolutionary approximation leads to a new circuit approximation strategy that is able to effectively escape local optima and thus to explore the design space more effectively than pure evolutionary search strategies.

Chapter 7

Novel Mutation Operator for Approximate Circuit Design

The contents of this chapter are based on our work published in [17]. In this chapter, we try to enhance the performance of search based circuit approximation by further improving the *synthesizer* component. First, we examine the various CGP mutation operators available in literature and evaluate their performance in various circuit approximation scenarios. Second, we propose a novel operator tailored to the task of arithmetic circuit approximation. In the experimental evaluation, we focus both on the speed of convergence of the mutations operators as well as on the quality of the final approximate solutions.

7.1 Motivation

There are two main challenges related to the CGP-driven circuit approximation: (1) *Fast and reliable evaluation of candidate solutions* that is essential for CGP to drive the exploration towards high-quality solutions. (2) *Convergence of the search* that directly determines the overall performance of the approximation process and thus the resulting scalability and applicability. While several circuit evaluation techniques, including parallel circuit simulation [121], formal methods [13], or statistical methods [64], have been proposed and successfully applied, dedicated search strategies or mutation operators have not yet been adequately addressed. Instead, state-of-the-art circuit approximation techniques based on CGP typically use a standard CGP mutation operator to generate the offspring from the parent. At the same time, specific search operators and search strategies have been proposed for various other applications of CGP (see, e.g., [70, 55, 37, 38, 48, 54]). However, these operators are not suitable for circuit approximation as they leverage various specific features of the domains for which they were introduced.

In this chapter, we first thoroughly study the impact of the standard mutation operators for CGP on the performance of the circuit approximation process. In the past, these operators were investigated in the context of evolutionary circuit design (from scratch) [122, 117, 107] or evolutionary circuit optimization (aiming at minimizing the number of gates) [115, 112]. Both of these tasks were previously targeted for relatively simple circuits. This chapter addresses a different problem—namely, an *efficient mutation mechanism for CGP-based approximation of complex circuits* where the evolutionary process has to primarily remove some gates and possibly re-connect the remaining ones. In particular, we focus on the

strategy that identifies the genes to be mutated and the new values of the altered genes, the mutation frequency, and the role of explicitly neutral mutations.

Further, we explore in which scenarios the so-called *deactivation operations* that only remove parts of the circuit can effectively drive the evolution towards high-quality circuit approximation. Note that removing a part of the circuit is a typical operation for non-evolutionary circuit approximation techniques like SALSA [125] and thus it seems promising to suitably leverage the deactivation in the mutation operator.

Finally, based on a very detailed experimental evaluation, we select the best mutation and deactivation strategy and propose a novel combined operator, called *SagTree*, that incorporates sub-tree deactivation into the mutation process.

In order to assess the performance and practical impact of the SagTree operator with respect to the existing operators, we consider a broad set of circuit approximation problems including several structurally different arithmetic circuits as well as various bit widths and target precisions. In total, our experiments include 39 approximation problems and over 14,000 approximation runs. We go beyond the standard analysis of the final approximate circuit and also assess the speed of convergence.

Using a rigorous statistical evaluation as well as a detailed analysis of selected results, we demonstrate that the SagTree operator provides a very robust and versatile circuit mutation strategy that significantly outperforms other existing operators used in the CGP-based approximation so far [95]. In particular, there are some hard approximation problems where our newly proposed operator is the only one able to achieve the required reduction within the given limit of 1 million generations. Moreover, for many problems, the SagTree operator improves the performance (due to reducing the number of required generations) by more than one order of magnitude. As such, the proposed mutation operator significantly improves the performance of the state-of-the-art approximation techniques.

7.2 Mutation Operators in CGP

The standard CGP uses either *point mutation* or *probabilistic mutation*. The point mutation randomly modifies up to h genes of a parent genotype to create an offspring. The number of genes can be specified as the percentage of the total number of genes or as an absolute number depending on the particular implementation [71]. In probabilistic mutation, every gene is considered for mutation according to a user-defined probability. Point mutation is easier to implement and more efficient than using a probabilistic mutation as one does not need to linearly walk through the genes to decide which ones to mutate. On the other hand, probabilistic mutation is continuous and allows one to choose very low mutation probabilities. In [48], the authors study the performance of these operators in the context of evolutionary circuit design (i.e. the design from scratch). In the area of circuit optimisation and approximation starting from a known circuit implementation, the existing techniques utilise the point mutation only [119].

Considering the CGP encoding, a single mutated gene causes either re-connection of a node, re-connection of a primary output, or change in the function of a node. Due to the presence of redundant genes, the mutation may occur in the redundant part, which means that the mutated genotype has the same phenotype as its parent. Such a mutation is sometimes denoted as neutral since the fitness value remains unchanged. In such a case, one does not need to carry out the fitness evaluation. It is theorised, however, that while neutral mutations may not actively contribute to an individual's fitness, future non-neutral

mutations might result in beneficial phenotypical traits as a consequence of past neutral mutations [111, 133].

To avoid wasted fitness evaluations, several mutation strategies have been proposed [37, 72]. One possibility is to detect the neutral mutations and skip the time-consuming fitness evaluation procedure. To avoid the mutation parameters whose optimal value is hard to determine in advance, Goldman and Punch introduced a parameter-less mutation strategy denoted as Single Active Mutation (SAM) [37]. The SAM operator mutates the offspring until one active gene is changed. The performance of this operator was close to the best performance of the other strategies [72] when evaluated in the evolutionary design of some small logic circuits. This operator has never been applied in the context of the evolutionary design of approximate circuits.

In [54], two mutation operators explicitly addressing the active and inactive nodes have been introduced. One operator activates the inactive nodes, the second one deactivates the active nodes. The latter operator alters connections to an active node so that the node becomes inactive. More detailed experimental studies are required to confirm benefits of these operators in the context of approximate circuit design.

7.3 Towards Efficient Mutation

In this section, we first discuss candidates for the best mutation and deactivation strategy and then propose a novel combined operator.

7.3.1 Single Active Gene Mutation Operators

To analyze the importance of explicitly neutral mutations in the task of circuit approximation, we propose to evaluate two variants of the SAM operator.

The *Single Active Gene Mutation* (SAGM) performs a single mutation to the active genes and does not alter the inactive genes. This operator suppresses the explicitly neutral mutations, but it still enables the implicit neutrality which refers to mutations affecting active genes and therefore leading to a change in the phenotype but without causing a change in fitness.

Single Active Gene and Explicitly Neutral Mutation (SAGENM) is an extension of the SAGM operator. Except for the one active mutation, it also performs mutations of the inactive part of the chromosome. The frequency of the inactive mutations is controlled by a parameter P_N. Similarly to the probabilistic mutation, SAGENM operator iterates over all inactive genes and mutates each of them with probability P_N. The differentiation between SAGM and SAGENM should allow us to better examine the contributions of the explicitly neutral mutations in the task of circuit approximation.

Due to performing a single mutation, the SAG family of operators has the potential to better solve problems that require small incremental changes—circuit approximation starting from a golden model typically belongs to such a category of problems. However, introducing large changes into the golden model can be sometimes very beneficial. SAGENM operators can achieve this by connecting inactive parts of the chromosome to the active part in a single mutation.

7.3.2 Node Deactivation Operators

Circuit pruning based on the deletion of a part of the circuit represents one of the conventional heuristic techniques used in approximate computing [62]. This technique is based on the assumption that the approximate circuit needs a part of the original circuit structure only to provide the required quality. As the standard mutation operators would typically require multiple iterations to remove a larger part of the circuit, we propose two variants of mutation operators that are able to eliminate a part of the circuit in a single step.

The *Active Node Deactivation* (NodeDeact) operator deactivates a single active node. The deactivation is implemented as follows. Firstly, a single active node N is chosen randomly. Then, the inputs of all nodes that are connected to this node are reconnected (i.e. substituted) to one of the inputs of N. The inputs are selected randomly and individually for each substitution.

The *Active Subtree Deactivation* (TreeDeact) represents a generalization of the previous operator. The operator randomly selects an active node N and identifies a sub-circuit S at the level of the phenotype, i.e. S has the root node N and depth up to d levels. Then, the operator disconnects the whole sub-circuit. The disconnection is implemented as follows. The inputs of all nodes outside of S that are connected to any node belonging to S are reconnected (i.e. substituted) to one of the inputs of S. The substitution works in the same way as in the case of the NodeDeact operator.

Note that S can have more than one output connection to the preserved active nodes. Several changes can happen in the chromosome as a consequence of a single TreeDeact mutation. This is illustrated in Figure 7.1 where inputs of four active nodes need to be reconnected.

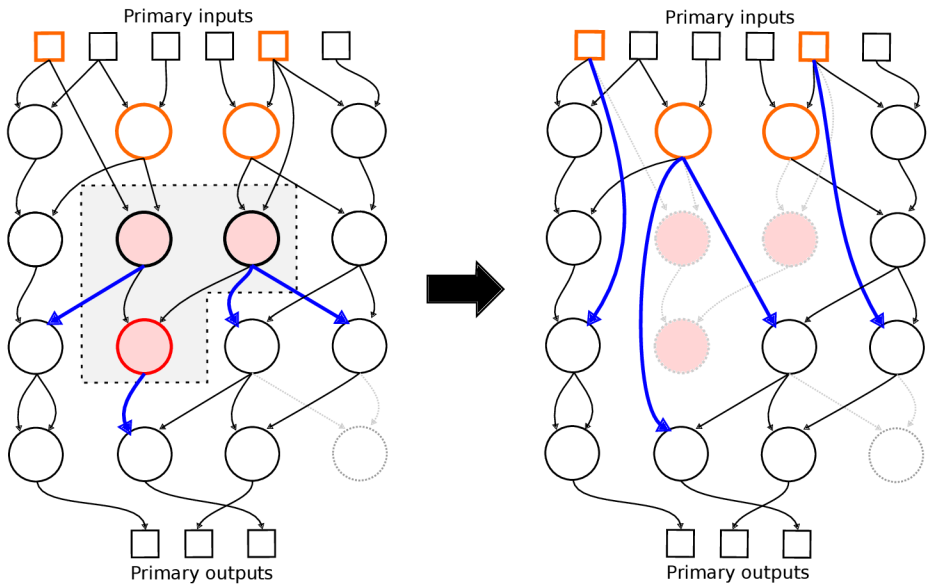


Figure 7.1: An example phenotype of a circuit before (left) and after (right) a TreeDeact mutation. An active gate is chosen (red), and the sub-circuit of the depth of 2 is identified. In the mutation, the outgoing connections of the subtree (blue) are randomly reconnected to the inputs of the subtree (orange). After the application of the mutation operator, the nodes of the subtree are inactive.

7.3.3 Combined Mutation Operators

The idea of combined mutation operators in the circuit approximation is very natural: the deactivation operations are able to quickly identify and eliminate redundant logic while the random mutation operators can find some novel node interconnections that preserve the required circuit functionality and reduce the circuit area. Although the elimination is typically more efficient at the beginning of the approximation process and, on the other hand, the mutation at the end of the process, we emphasize that it is also very beneficial to interleave these two operations. Exploring new interconnections at the beginning is important to decrease the probability that the elimination will get stuck in a local optimum by removing some useful logic. Enabling the elimination in the later phases is useful as the new interconnections can create some redundant logic.

Therefore, we consider combined operators that perform, in every generation, a gene-oriented mutation operation M with a probability P_M and a deactivation-oriented operation R with the probability $P_R = 1 - P_M$. This approach leads to various combined operators denoted as $A - B$, each controlled by the parameter P_M (e.g. $Point - TreeDeact(P_M = 0.50)$, $SAGM - NodeDeact(P_M = 0.75)$). Our strategy for selecting the best combined operator is based on a detailed experimental evaluation of the purely mutating and deactivating operations. As shown in the next section, the experiments unequivocally identify the SAGM and TreeDeact as the most vital operators for building the combined operator.

7.4 Experimental Setup

The presented mutation operators were implemented in our tool ADAC (Automatic Design of Approximate Circuits, Chapter 5). ADAC provides several techniques for evaluating various error metrics of approximate circuits. In particular, it provides an efficient SAT-based evaluation of WCAE (Section 3.3), one of the most commonly used error metrics, which we consider in the following experimental evaluation. CGP is driven by the fitness function defined by Equation 2.1. To estimate the size of the evolved circuits, we employ the heuristic described in Section 3.4.1 (the size of the circuit is estimated as the sum of the circuit's gate sizes). We use the gate sizes presented in Table 3.1.

In order to thoroughly assess the performance and practical impact of the proposed mutation operators, we consider a broad set of circuit approximation problems including several structurally different arithmetic circuits as well as various bit widths and target precisions. In the experimental evaluation, we use circuits characterised in Table 7.1[1].

We consider various WCAE values as the given bound directly determines permissible changes in the circuit structure (i.e. a small error allows one to perform smaller changes in the circuit only). Therefore, different WCAE values lead to significantly different approximation problems.

In our experimental evaluation, we explore all three dimensions characterising the circuit approximation problems: (i) the circuit type reflecting both the size and the structural complexity, (ii) the error bound, and (iii) the approximation time. In total, we examine more than 39 instances of the approximation problems that sufficiently cover practically relevant problems in the area of arithmetic circuit approximation. Therefore, the considered benchmark allows us to robustly evaluate the versatility of the proposed mutation operators and their benefits with respect to the existing operators.

[1] All circuits can be downloaded from https://github.com/ehw-fit/sagtree-seeds. These circuits can be used in the future as a new benchmark suite for the area of evolutionary circuit approximation.

Table 7.1: Arithmetic circuits used as a benchmark: the 2nd column gives the computed function where i_n denotes an n-bit operand, #PI/#PO give the numbers of primary inputs/outputs, and the last column gives the number of gates in the accurate implementation used as the seeding circuit.

Circuit	Function	#PI	#PO	# gates
Adders	$i_8 + i_8$	16	9	54
	$i_{16} + i_{16}$	32	17	115
Multipliers	$i_8 \times i_8$	16	16	346
	$i_{16} \times i_{16}$	32	32	1525
MAC	$(i_8 \times i_8) + i_{16}$	32	17	473
	$(i_{16} \times i_{16}) + i_{32}$	64	33	1788
Square	$(i_8)^2$	8	16	317
	$(i_{16})^2$	16	32	1394
Dividers	i_{15}/i_8	23	8	648
	i_{31}/i_{16}	47	16	2720

7.5 Experimental Results

We divide our experimental results into three parts:

1. Evaluation of the existing mutation operators allowing us to choose the best candidates for the new combined operator:
   - The point operator with the maximum number of mutations set to 0.5 % of the number of gates of the original circuit—the standard operator for circuit approximation further denoted as $Point(P_G = 0.005)$;
   - the SAGM operator;
   - the SAGENM operator with the neutral mutation probability set to $[0.1\ \%, 1\ \%, 3\ \%]$, denoted, e.g., as $SAGENM(P_N = 0.001)$; and
   - the probabilistic operator with the mutation probabilities set to [0.1 %, 0.5 %, 1 %], denoted, e.g., as $Prob(P = 0.001)$.
2. Evaluation of the purely deactivating operators with respect to the performance and convergence of the standard operators:
   - the single-gate NodeDeact operator and
   - the TreeDeact operator with a variable tree depth chosen randomly from the interval $[1, 5]$.
3. Evaluation of the proposed combined operator SAGM-TreeDeact (further denoted as $SagTree$) in three versions that perform the SAGM mutation with the probability P_M set to $[25\ \%, 50\ \%, 75\ \%]$. We denote them, e.g., as $SagTree(P_M = 0.25)$.

In order to provide a clear and succinct presentation of the experimental results including over 14,000 evolutionary runs (29 independent evolutionary runs with 10^6 generations for each circuit type, bit width, and approximation error), we use three types of result visualisation:

1. Complete statistical evaluation over all evolutionary runs using the Friedmann statistical test along with the Nemenyi post-hoc analysis. Based on the analysis, we report the ranks of the operators for all approximation runs (Figure 7.2). The ranks are obtained using the results of the approximation process (i.e. the reduction of the circuit area) taken at 1k, 5k, 10k, 50k, 100k, 250k, 500k, and 1M generations. The results thus take into consideration the performance of the operators during the whole approximation process. Note that the post-hoc analysis defines the Critical Difference (CD) on the ranks guaranteeing a statistically significant difference between two operators with confidence 95 %. Moreover, we provide Table 7.2 which also contains the results of the Friedmann and Nememyi tests, but this time performed at chosen points of the evolutionary process separately. The table demonstrates how the ranks of the particular operators change during the course of the approximation.

2. Figure 7.4 shows, for selected approximation problems, the time course of the approximation runs (i.e. convergence of the area reduction) using a particular mutation operator. The figures show the median of the aggregated runs for each operator.

3. Figure 7.3 shows the median of the number of generations required by the particular mutation operator to achieve the given area reduction for different circuits and WCAE bounds. The figure includes only the more complex approximation problems requiring a considerable number of generations. The white places represent the situation where the required area reduction (bottom x-axis) has not been achieved in at least 50 % of the approximation runs using the respective mutation operator.

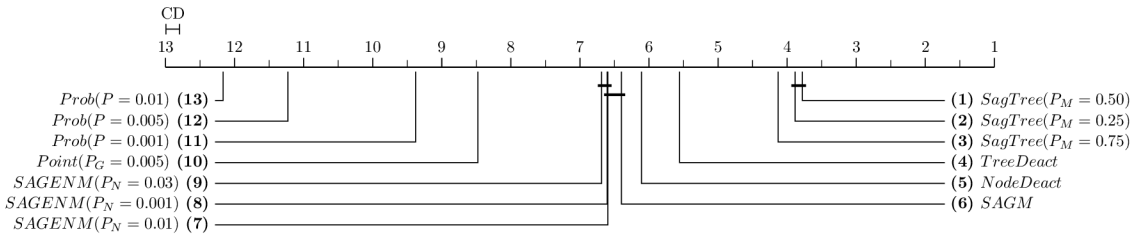


Figure 7.2: An overall rank comparison using the results at 1k, 5k, 10k, 50k, 100k, 250k, 500k, 750k, 1M generations. Groups where the difference in performance is not statistically significant (the difference is smaller than the critical difference) are joined.

7.5.1 Existing Mutation Operators

The statistical results (see Figure 7.2 and Table 7.2) clearly show that the operators based on single gene mutation outperform the other existing mutation operators (namely the Point operator for circuit approximation as well as the different variants of the probabilistic mutation) across the whole set of experiments. The single gene mutation has the best ranks with statistical significance for both the particular generation counts and the overall evaluation. In the following paragraphs, we briefly comment the results.

The probabilistic operators heavily rely on the size of the chromosome: the larger the chromosome is, the more mutations the probabilistic operators perform. It is known that a large number of mutations negatively effects the performance of CGP [37]. This means that, for large circuits, the mutation frequency has to be set very low (i.e. $P = 0.001$) to achieve any improvements. Figure 7.3 demonstrates that, for higher frequencies, the

Table 7.2: A rank comparison of the tested mutation operators using the results obtained solely at various points of the evolutionary process. Values in the table cells show the placing of the operator and its average rank (in the bracket). The value of the critical difference of the ranks is 0.46.

Generation	1 000	10 000	100 000	1 000 000
$Point(P_G = 0.005)$	10 (9.1)	10 (8.9)	10 (8.7)	8 (7.4)
$SAGENM(P_N = 0.001)$	6 (7.7)	7 (7.2)	7 (6.8)	7 (5.6)
$SAGENM(P_N = 0.01)$	7 (7.8)	8 (7.3)	8 (6.8)	6 (5.5)
$SAGENM(P_N = 0.03)$	9 (7.9)	9 (7.5)	9 (6.9)	5 (5.4)
$SAGM$	8 (7.8)	6 (7.1)	6 (6.6)	4 (5.0)
$Prob(P = 0.001)$	11 (9.9)	11 (9.9)	11 (9.6)	9 (8.3)
$Prob(P = 0.005)$	12 (11.7)	12 (11.6)	12 (11.3)	12 (10.6)
$Prob(P = 0.01)$	13 (12.2)	13 (12.3)	13 (12.2)	13 (12.0)
$NodeDeact$	3 (3.2)	4 (3.9)	5 (5.9)	11 (9.4)
$TreeDeact$	**1 (2.3)**	2 (3.4)	4 (5.3)	10 (9.0)
$SagTree(P_M = 0.25)$	2 (3.0)	**1 (3.4)**	2 (3.7)	3 (4.9)
$SagTree(P_M = 0.50)$	4 (3.6)	3 (3.8)	**1 (3.5)**	2 (4.1)
$SagTree(P_M = 0.75)$	5 (4.9)	5 (4.8)	3 (3.7)	**1 (3.9)**

operator is not able to achieve (within 1M generations) the required reductions for most of the approximation problems—it works well for small circuits such as 16-bit adders only (see Figure 7.4A). Even for the low frequency, the operator is significantly worse (in terms of performance and success rate) than the other operators on larger circuits (see, e.g., Figure 7.3, region I). This indicates that the probabilistic mutation paradigm is not very suitable for general circuit approximation.

The Point operator performs slightly better than $Prob(P = 0.001)$. It breaks the purely probabilistic paradigm by considering an interval [1,X] on the number of mutations (X depends on the circuit size). Our results, however, clearly demonstrate that the single gene mutations are more efficient and outperform the Point operator, especially for more complicated approximation problems.

Finally, we observe that the difference between SAGM and various neutral mutation frequencies of SAGENM is in most cases not statistically significant. We can therefore reason that neutral mutations do not bring any positive effect in circuit approximation. The same conclusion can be deduced from Figure 7.3 where these operators feature almost identical numbers of generations to reach the saving thresholds.

7.5.2 Mutation vs. Deactivation Operators

In this section, we aim at the essential question: how can the deactivation operators improve the circuit approximation process and where their limitation is. Recall that the main advantage of the deactivation operators is the quick and effective elimination of the redundant circuit logic. By redundant logic, we mean the parts of the original circuit that are not necessary to implement the approximate functionality. However, after all such logic is removed from the candidate solution, these operators are unable to provide any further improvements. This is crucial for approximation of smaller circuits (see Figures 7.4A

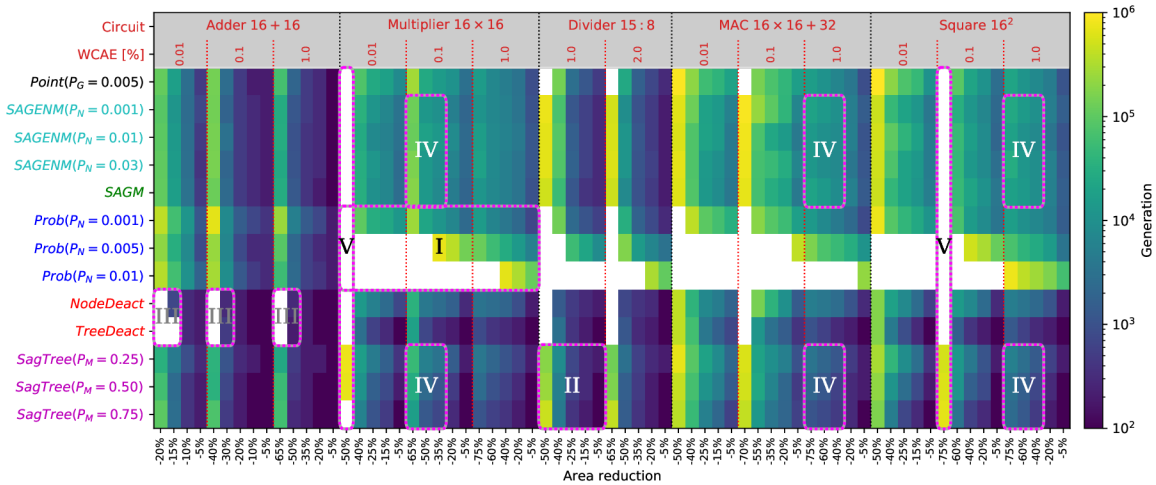


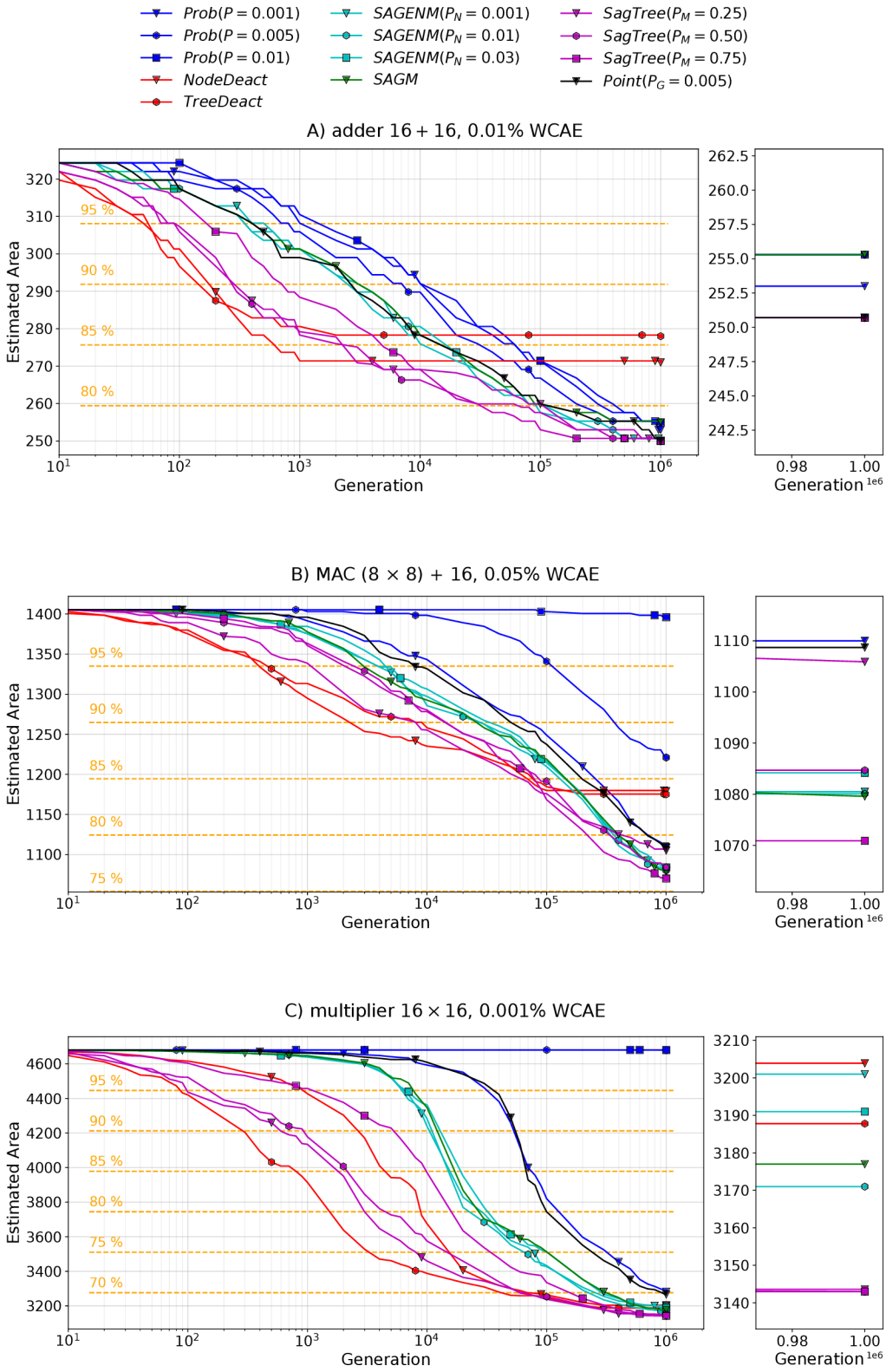
Figure 7.3: Median of the number of generations required to achieve given area reductions (x-axis) for the tested mutation operators and the given circuits and maximal allowed worst-case arithmetic errors. Lower values are better; white places represent the situation where more than a half of the runs did not achieve the target area reduction. The regions marked by the Roman numerals are further discussed in the text.

and 7.4B) where deactivation operators quickly converge to local optima (all redundant logic is removed) and then are significantly outperformed by the standard mutation operators in the later stages of the evolution. For more complex circuits (see the approximation of the 16-bit multiplier in Figure 7.4C), the deactivation operators provide much faster convergence than the standard mutations and are competitive even in the later stages. For very complex approximation problems (see the approximation of 31-bit dividers in Figure 7.4D), the deactivation operators dominate across the whole approximation process.

Similar trends can also be observed in Table 7.2. After 1k generations, the TreeDeact and NodeDeact operators are ranked first and third, respectively, and both still place in the top five after 10k and 100k generations. However, at the end of the evolution, most of the other operators surpass the solution provided by solely deactivating operators—TreeDeact places tenth and NodeDeact eleventh.

We refine these observations and add more quantitative results in the following analysis comparing the best standard mutation operator SAGM with the best deactivation operator TreeDeact on selected circuit approximation problems:

- 16-bit adder (Figure 7.4A): SAGM outperforms TreeDeact after approximately 25k generations. In the end, the solution provided by SAGM is about 11 % better.
- 16-bit MAC (Figure 7.4B): SAGM outperforms TreeDeact after approximately 180k generations. In the end, the solution provided by SAGM is 8.5 % better.
- 16-bit multiplier (Figure 7.4C): SAGM outperforms TreeDeact after roughly 750k generations. In the end, the solution provided by SAGM is only 0.5 % better. However, after 50k generations, TreeDeact is 10.8 % better than SAGM.
- 31-bit divider (Figure 7.4D): TreeDeact outperforms SAGM in all stages of the evolution. The difference is 5.9 % after 50k generations and 3.1 % at the end of the evolution.


Prob(P = 0.001)
Prob(P = 0.005)
Prob(P = 0.01)
NodeDeact
TreeDeact
SAGENM(P_N = 0.001)
SAGENM(P_N = 0.01)
SAGENM(P_N = 0.03)
SAGM
SagTree(P_M = 0.25)
SagTree(P_M = 0.50)
SagTree(P_M = 0.75)
Point(P_G = 0.005)
A) adder 16 + 16, 0.01% WCAE
Estimated Area
Generation
95 %
90 %
85 %
80 %
B) MAC (8 × 8) + 16, 0.05% WCAE
75 %
C) multiplier 16 × 16, 0.001% WCAE
70 %
1e6

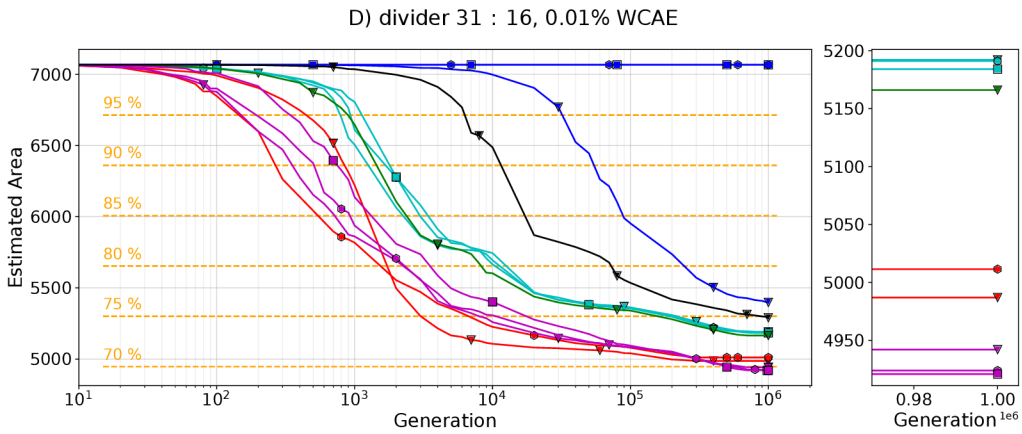


Figure 7.4: Progress of the median solution areas for selected approximate circuits. Orange lines show the proportional area savings with respect to the original solution.

To conclude these experiments, the deactivation operators perform better in the initial stages of the evolution. The standard genetic mutation operators have the potential to provide better improvements if given enough generations. The larger the circuit (and the harder the approximation problem), the longer it takes the standard operators to catch up with the deactivation operators.

7.5.3 The Combined Operator SagTree

The two previous Sections justified our strategy for building the combined operator SagTree that has the merits of the best deactivation operator TreeDeact and the best random mutation operator SAGM. SagTree ensures quick convergence to a near optimal solution using the aggressive sub-circuit deactivation. In the early stages of the evolution, the progress of SagTree is only slightly slower than the progress of the deactivation operators. When the purely deactivating operators get stuck in a local optimum and struggle to provide additional improvements, SagTree quickly takes over and further optimizes the circuit using random mutations.

We first compare different variants of the SagTree operator varying in the frequency of the deactivation operators. As expected from the previous experiments, increasing the probability of a deactivation operation is beneficial in the earlier stage of the approximation process as it improves the convergence. This claim is also supported by Table 7.2. After 1k and 10k generations, the ranks of the SagTree operators are ordered by the frequency P_M in an ascending order. On the other hand, after 1M generations, the order is reversed (the deactivation operations reach their limit, and only the gene-oriented mutations are able to bring improvements in the solution), and $SagTree(P_M = 0.75)$ overcomes the performance of the other operators. We also observe that $SagTree(P_M = 0.25)$ is typically slightly faster for small area reductions but significantly lags behind $SagTree(P_M = 0.75)$ for large area reductions (see, e.g., Figure 7.3, region II).

The following analysis provides a more detailed comparison with existing operators:

- 16-bit adder (Figure 7.4A): SagTree is better than TreeDeact already after 10k generations, and the gap steadily increases. SAGM almost (but not quite) catches up with SagTree mutations in the second half of the evolutionary runs.

- 16-bit MAC (Figure 7.4B): SagTree mutations overcome the deactivation operators after roughly 100k generations. The SAGM mutation equals the performance of $SagTree(P_M = 0.50)$ and $SagTree(P_M = 0.75)$ after about 500k generations but does not manage to overcome the savings provided by $SagTree(P_M = 0.25)$.

- 16-bit multiplier (Figure 7.4C): after 400k generations, all SagTree frequencies are better than the deactivation operators. SagTree is also better than SAGM throughout the whole course of evolution, although at the end the difference between the two operators is about 1 % only.

- 31-bit divider (Figure 7.4D): SagTree mutations are better after about 350k generations. Again, the SagTree mutations outperform SAGM in the whole evolutionary process; at the end, the difference in the final chip area is 4.5 %.

Finally, we summarise the outcomes of the experimental evaluation presented in Figure 7.3. For many instances, the purely deactivating operators fail to achieve large reductions within 1M generations (e.g. Figure 7.3, regions III). For many instances, the combined operators improve the performance (reduce the number of generations) by more than one order of magnitude compared with the existing genetic operators (e.g. Figure 7.3, regions IV). There are some hard approximation problems where only the combined operators are able to achieve the required reduction within the approximation process limited to 1M generations (e.g. Figure 7.3, regions V).

To conclude the experimental results, we recall the overall statistical evaluation in Figure 7.2, where we compare the ranks of the mutation operators across multiple points in the approximation process. The overall best-performing operator is $SagTree(P_M = 0.50)$, closely followed by $SagTree(P_M = 0.25)$. All other operators significantly lag behind.

7.6 Conclusion

In this chapter, we have discussed the existing mutation operators available in the literature in the context of CGP-based circuit approximation. We have demonstrated that the mutation operator for CGP has a fundamental impact on the performance and convergence of the evolution-driven circuit approximation. Using a thorough experimental evaluation, we have showed that the classical Point CGP operators known from the literature can be significantly improved by dedicated deactivation operations eliminating the redundant circuit logic. We have designed a new mutation operator that combines the single active gene mutation with circuit sub-tree deactivation and significantly outperforms existing operators on a wide and practically relevant set of circuit approximation problems.

Chapter 8

Conclusion

Approximate computing is an emerging paradigm with the potential to achieve significant resource savings in many application domains. Approximate computing can be performed on on various system levels ranging from approximate storage and software to dedicated approximate circuits. Approximate arithmetic circuits are especially interesting since such circuits can serve as the basic building blocks of various more complex applications. Since manual design of larger approximate arithmetic circuits is a very complex problem, various automatic approaches have been proposed in the literature.

The main goal of this thesis was to improve the performance and scalability of search-based arithmetic circuit approximation. Specifically, the thesis focuses on approximation of circuits using the Cartesian Genetic Programming. This variant of evolutionary algorithm is well suited for circuit approximation and has been successfully used to evolve approximate circuits in the past. However, CGP also suffers from scalability issues when approximating larger circuit instances. There are two main areas that can be improved in order to enhance the overall performance of based approximation: (i) the analyser, that evaluates the error of candidate solutions, and (ii) the synthesizer, that generates new candidates.

Since CGP usually needs many iterations to find high-quality approximate solutions, it is crucial that error evaluation is as quick as possible. We aimed at the improvements of the analyser component with the goal of evaluation acceleration in Chapters 3 and 4. We utilised satisfiability based WCAE and WCRE error evaluation with an improved miter construction to accelerate the circuit approximation. We further proposed and employed the verifiability driven strategy with adaptive resource limit settings. We evaluated the performance of the proposed approximation framework on a wide range of approximation problems – various arithmetic circuits with different bit widths and multiple error thresholds. When compared to the current state of the art approximate solutions, the proposed method was able to create approximate multipliers of unprecedented complexity and quality. The experiments showed, that Cartesian Genetic Programming currently represents one of the best approaches to search based design of approximate circuits.

In Chapter 5, we briefly introduced our tool called ADAC – a framework for automated design of approximate circuits. The framework is implemented as a module of the ABC hardware design and verification tool. It utilises ABC's optimised circuit representation structures as well as the powerful satisfiability solving engine *iprove*. ADAC contains all the techniques and algorithms presented in Chapters 3 and 4 and was also used for all the experimental evaluations presented in these chapters.

Improvements of the synthesizer component represent an orthogonal approach to the enhancement of the error evaluation algorithms. We aimed at the improvements of the

synthesizer in Chapters 6 and 7. In Chapter 6, we explored the possibilities of approximation using satisfiability-based exact and approximate synthesis. We also combined the exact synthesis optimisation with the evolutionary approximation to improve the quality of the final solutions obtained in long approximation runs. In Chapter 7, we focused on the mutation operators utilised in CGP. We first provided a detailed experimental evaluation of the existing operators, and, based on the presented results, we proposed a novel combined mutation operator called SagTree. The evaluation performed on an extensive approximation benchmark showed how the SagTree operator offers an overall better convergence when compared to the existing operators.

Overall, the results presented in the thesis significantly enhance the capabilities of the state-of-the-art search based circuit approximation. The improvements made to the synthesizer and analyser components allow us to efficiently find approximations of various arithmetic circuits – scaling up to 32-bit multipliers and 128-bit adders – with high-quality trade-offs between the approximation error and the resource savings.

Future Research Directions

As we have shown in this thesis, the approximate computing methodologies can greatly benefit from the utilisation of formal verification techniques. Although we have investigated mainly the improvements brought by SAT and SMT solving, there exist other techniques yet to be more deeply embedded in the approximate circuit design. The newly emerging techniques that could be extended towards approximate equivalence checking include verification techniques based on polynomial representations (Gröbner Basis) [34, 92] or #SAT (counting SAT) [19].

Nevertheless, future research in the area of the automated approximate circuit design (and approximate computing in general) includes many other interesting directions as well. One of them is the detailed examination of the correlation of various error metrics. Such knowledge could lead to designing circuits with a quality trade-off between non-functional parameters and multiple error metrics important for the target application. One work in this research direction was published in [12], a paper co-authored by the author of this thesis. In this work, we examined how various error metrics relate to each other and how the more complex error constraints affect the performance of the evolutionary design algorithm and the quality of the final solutions.

Another direction that needs a deeper research effort is circuit approximation tailored to the given application (e.g. signal processing [131], neural networks [3], etc.). The specialised approximate circuits can take into account the input distribution specific for the target application and can be approximated with regards to the most suitable error metrics. Such circuits have the potential to provide better performance in the target application than the generic approximate circuits. To enable a more widespread usage of approximate circuits in NNs, it is crucial to improve the techniques that simulate the performance of approximate circuits in NNs and also allow to train NNs using approximate circuits.

For some complex application domains (e.g. arithmetic circuits of large bit widths, sequential circuits, etc.), there still remains a need for better scalability and further automation of the approximation algorithms. Higher scalability and more effective synthesising algorithms open the possibilities to approximate larger parts of the computation systems, leading to better resource savings, more possible applications and wider usage.

Bibliography

[1] Almurib, H., Kumar, T.N., Lombardi, F.: Approximate DCT image compression using inexact computing. IEEE Transactions on Computers **67**, 149–159 (2018)

[2] Ansaloni, I.S.G., Pozzi, L.: Circuit carving: A methodology for the design of approximate hardware. In: Proc. of DATE'18. pp. 545–550. IEEE (2018)

[3] Ansari, M.S., Mrazek, V., et al.: Improving the accuracy and hardware efficiency of neural networks using approximate multipliers. IEEE Transactions on Very Large Scale Integration (VLSI) Systems **28**, 317–328 (2020)

[4] Audemard, G., Simon, L.: On the Glucose SAT solver. International Journal on Artificial Intelligence Tools **27**, 1–25 (2018)

[5] Baek, W., Chilimbi, T.M.: Green: A framework for supporting energy-conscious programming using controlled approximation. In: Proc. of PLDI'10. pp. 198–209. ACM (2010)

[6] Bhardwaj, K., Mane, P.S., Henkel, J.: Power- and area-efficient approximate Wallace Tree Multiplier for error-resilient systems. In: Proc. of ISQED'14. pp. 263–269. IEEE (2014)

[7] Brayton, R., Mishchenko, A.: ABC: An academic industrial-strength verification tool. In: Proc. of CAV'10. pp. 24–40. Springer (2010)

[8] Brent, R.P., Kung, H.T.: A regular layout for parallel adders. IEEE Transactions on Computers **C-31**, 260–264 (1982)

[9] Bruttomesso, R., Cimatti, A., et al.: The MathSAT 4 SMT solver. In: Proc. of CAV'08. pp. 299–303. Springer-Verlag (2008)

[10] Bryant, R.: Graph-based algorithms for Boolean function manipulation. IEEE Transactions on Computers **C-35**, 677–691 (1986)

[11] BuDDy: A BDD package. http://buddy.sourceforge.net/manual/main.html, online; March 2023

[12] Češka, M., Matyáš, J., Mrázek, V., Vojnar, T.: Designing approximate arithmetic circuits with combined error constraints. In: Proc. of DSD'22. pp. 785–792. IEEE (2022)

[13] Češka, M., Matyáš, J., Mrazek, V., et al.: Approximating complex arithmetic circuits with formal error guarantees: 32-bit multipliers accomplished. In: Proc. of ICCAD'17. pp. 416–423. IEEE (2017)

[14] Češka, M., Matyáš, J., et al.: ADAC: Automated design of approximate circuits. In: Proc. of CAV'18. pp. 612–620. Springer (2018)

[15] Češka, M., Matyáš, J., et al.: Adaptive verifiability-driven strategy for evolutionary approximation of arithmetic circuits. Applied Soft Computing **95**, 1–17 (2020)

[16] Češka, M., Matyáš, J., et al.: Satisfiability solving meets evolutionary optimisation in designing approximate circuits. In: Proc. of SAT'20. pp. 481–491. Springer International Publishing (2020)

[17] Češka, M., Matyáš, J., et al.: SagTree: Towards efficient mutation in evolutionary circuit approximation. Swarm and Evolutionary Computation **69**, 1–10 (2022)

[18] Češka, M., Češka, M., Matyáš, J., Pankuch, A., Vojnar, T.: Approximating complex arithmetic circuits with guaranteed worst-case relative error. In: Proc. of Eurocast'19. pp. 482–490. Springer Verlag (2020)

[19] Chakraborty, S., Meel, K.S., et al.: Approximate probabilistic inference via word-level counting. In: Proc. of AAAI'16. pp. 3218–3224. AAAI Press (2016)

[20] Chandrasekharan, A., Soeken, M., Große, D., Drechsler, R.: Precise error determination of approximated components in sequential circuits with model checking. In: Proc. of DAC'16. pp. 129:1–129:6. ACM (2016)

[21] Chandrasekharan, A., Soeken, M., et al.: Approximation-aware rewriting of AIGs for error tolerant applications. In: Proc. of ICCAD'16. pp. 1–8. IEEE (2016)

[22] Chippa, V.K., Chakradhar, S.T., Roy, K., Raghunathan, A.: Analysis and characterization of inherent application resilience for approximate computing. In: Proc. of DAC'13. pp. 1–9. IEEE (2013)

[23] Cho, K., Lee, Y., et al.: eDRAM-based tiered-reliability memory with applications to low-power frame buffers. In: Proc. of ISLPED'14. pp. 333–338. IEEE (2014)

[24] Ciesielski, M., Yu, C., et al.: Verification of gate-level arithmetic circuits by function extraction. In: Proc. of DAC '15. pp. 52:1–52:6. ACM (2015)

[25] Dadda, L.: Some schemes for parallel multipliers. Alta Frequenza **34**, 349–356 (1965)

[26] Demšar, J.: Statistical comparisons of classifiers over multiple data sets. Journal of Machine Learning Research **7**, 1–30 (2006)

[27] Drechsler, R., Becker, B.: Binary Decision Diagrams - Theory and Implementation. Springer (1998)

[28] Du, K., Varman, P., Mohanram, K.: High performance reliable variable latency carry select addition. In: Proc. of DATE'12. pp. 1257–1262. EDA Consortium (2012)

[29] Eén, N., Sörensson, N.: An extensible SAT-solver. In: Proc. of SAT'04. pp. 502–518. Springer (2004)

[30] Ercegovac, M.D.: Two-operand addition. In: Digital Arithmetic, pp. 50–135. Morgan Kaufmann (2004)

[31] Esmaeilzadeh, H., Sampson, A., et al.: Neural acceleration for general-purpose approximate programs. In: Proc. of MICRO'12. pp. 449–460. IEEE (2012)

[32] Farshchi, F., Abrishami, M.S., Fakhraie, S.M.: New approximate multiplier for low power digital signal processing. In: Proc. of CADS'13. pp. 25–30. IEEE (2013)

[33] Friedman, M.: A comparison of alternative tests of significance for the problem of m rankings. The Annals of Mathematical Statistics **11**, 86–92 (1940)

[34] Froehlich, S., Große, D., Drechsler, R.: Approximate hardware generation using symbolic computer algebra employing Grobner basis. In: Proc. of DATE'18. pp. 889–892. IEEE (2018)

[35] Froehlich, S., Große, D., Drechsler, R.: Error bounded exact BDD minimization in approximate computing. In: Proc. of ISMVL'17. pp. 254–259. IEEE (2017)

[36] Ghandali, S., Yu, C., et al.: Logic debugging of arithmetic circuits. In: Proc. of ISVLSI'15. pp. 113–118. IEEE (2015)

[37] Goldman, B.W., Punch, W.F.: Reducing wasted evaluations in Cartesian Genetic Programming. In: Proc of EuroGP'13. pp. 61–72. Springer-Verlag (2013)

[38] Goldman, B.W., Punch, W.F.: Analysis of Cartesian Genetic Programming's evolutionary mechanisms. IEEE Transactions on Evolutionary Computation **19**, 359–373 (2015)

[39] Grigorian, B., Reinman, G.: Dynamically adaptive and reliable approximate computing using light-weight error analysis. In: Proc. of AHS'14. pp. 248–255. IEEE (2014)

[40] Haaswijk, W., Mishchenko, A., Soeken, M., Micheli, G.D.: SAT based exact synthesis using DAG topology families. In: Proc. of DAC'18. pp. 1–6. ACM (2018)

[41] Han, C., Jiang, J.: When Boolean satisfiability meets Gaussian elimination in a simplex way. In: Proc. of CAV'12. pp. 410–426. Springer (2012)

[42] Hashemi, S., Tann, H., Reda, S.: BLASYS: Approximate logic synthesis using Boolean matrix factorization. In: Proc. of DAC'18. pp. 1–6. ACM (2018)

[43] He, K., Gerstlauer, A., Orshansky, M.: Controlled timing-error acceptance for low energy IDCT design. In: Proc. of DATE'11. pp. 1–6. IEEE (2011)

[44] Hegde, R., Shanbhag, N.R.: Soft digital signal processing. IEEE Transactions on Very Large Scale Integration (VLSI) Systems **9**, 813–823 (2001)

[45] Heule, M., Kiesl, B., Biere, A.: Encoding redundancy for satisfaction-driven clause learning. In: Proc. of TACAS'19. pp. 41–58. Springer (2019)

[46] Huang, G., Zhu, Q., Siew, C.: Real-time learning capability of neural networks. IEEE Transactions on Neural Networks **17**, 863–878 (2006)

[47] Huang, J., Kumar, T., et al.: Approximate computing using frequency upscaling. IET Circuits, Devices & Systems **13**, 1018–1026 (2019)

[48] Husa, J., Kalkreuth, R.: A comparative study on crossover in Cartesian Genetic Programming. In: Proc. of EuroGP'18. pp. 203–219. Springer (2018)

[49] Jiang, H., Han, J., Lombardi, F.: A comparative review and evaluation of approximate adders. In: Proc. of GLSVLSI'15. pp. 343–348. ACM (2015)

[50] Jiang, H., Liu, L., et al.: Approximate arithmetic circuits: Design and evaluation. In: Approximate Circuits, Methodologies and CAD, pp. 67–98. Springer (2019)

[51] Jiang, H., Santiago, F.J.H., et al.: Approximate arithmetic circuits: A survey, characterization, and recent applications. Proceedings of the IEEE **108**, 2108–2135 (2020)

[52] Jouppi, N., Young, C., et al.: In-datacenter performance analysis of a tensor processing unit. In: Proc. of ISCA'17. pp. 1–12. IEEE (2017)

[53] Judd, P., Albericio, J., et al.: Proteus: Exploiting numerical precision variability in deep neural networks. In: Proc. of ICS'16. pp. 1–12. ACM (2016)

[54] Kalkreuth, R.: Two new mutation techniques for Cartesian Genetic Programming. In: Proc. of ECTA'19. pp. 82–92. SciTePress (2019)

[55] Kaufmann, P., Platzner, M.: Advanced techniques for the creation and propagation of modules in Cartesian Genetic Programming. In: Proc. of GECCO'08. pp. 1219–1226. ACM (2008)

[56] Khudia, D.S., Zamirai, B., et al.: Rumba: An online quality management system for approximate computing. In: Proc. of ISCA'15. pp. 554–566. ACM (2015)

[57] Kim, Y., Zhang, Y., Li, P.: An energy efficient approximate adder with carry skip for error resilient neuromorphic VLSI systems. In: Proc. of ICCAD'13. pp. 130—137. IEEE (2013)

[58] Kogge, P.M., Stone, H.S.: A parallel algorithm for the efficient solution of a general class of recurrence equations. IEEE Transactions on Computers **C-22**, 786–793 (1973)

[59] Kulkarni, P., Gupta, P., Ercegovac, M.D.: Trading accuracy for power in a multiplier architecture. Journal of Low Power Electronics **7**, 490–501 (2011)

[60] Kumar, H.J.N.T., et al.: Simulation-based evaluation of frequency upscaled operation of exact/approximate ripple carry adders. In: Proc. of DFT'17. pp. 1–6. IEEE (2017)

[61] Li, C., Luo, W., et al.: Joint precision optimization and high level synthesis for approximate computing. In: Proc. of DAC'15. pp. 1–6. ACM (2015)

[62] Lingamneni, A., Enz, C., et al.: Energy parsimonious circuit design through probabilistic pruning. In: Proc. of DATE'11. pp. 1–6. IEEE (2011)

[63] Liu, C., Han, J., Lombardi, F.: A low-power, high-performance approximate multiplier with configurable partial error recovery. In: Proc. of DATE'14. pp. 1–4. IEEE (2014)

[64] Liu, G., Zhang, Z.: Statistically certified approximate logic synthesis. In: Proc. of ICCAD'17. pp. 344—351. IEEE (2017)

[65] Liu, S., Pattabiraman, K., et al.: Flikker: Saving DRAM refresh-power through critical data partitioning. In: Proc. of ASPLOS'11. pp. 213–224. ACM (2011)

[66] Mahdiani, H.R., Ahmadi, A., et al.: Bio-inspired imprecise computational blocks for efficient VLSI implementation of soft-computing applications. IEEE Transactions on Circuits and Systems **57**, 850 – 862 (2010)

[67] May, D., Stechele, W.: Voltage over-scaling in sequential circuits for approximate computing. In: Proc. of DTIS'16. pp. 1–6. IEEE (2016)

[68] Mazahir, S., Hasan, O., et al.: Probabilistic error modeling for approximate adders. IEEE Transactions on Computers **66**, 515–530 (2017)

[69] Miguel, J.S., Badr, M., et al.: Load value approximation. In: Proc. of MICRO'14. pp. 127–139. IEEE (2014)

[70] Miller, J.F.: An empirical study of the efficiency of learning Boolean functions using a Cartesian Genetic Programming approach. In: Proc. of GECCO'99. pp. 1135–1142. Morgan Kaufmann Publishers Inc. (1999)

[71] Miller, J.F.: Cartesian Genetic Programming. Springer-Verlag (2011)

[72] Miller, J.F.: Cartesian Genetic Programming: its status and future. Genetic Programming and Evolvable Machines **21**, 129–168 (2020)

[73] Mittal, S.: A survey of techniques for approximate computing. ACM Computational Survey **48**, 62:1–33 (2016)

[74] Mohapatra, D., Chippa, V.K., et al.: Design of voltage-scalable meta-functions for approximate computing. In: Proc. of DATE'11. pp. 1–6. IEEE (2011)

[75] Moura, L.D., Bjørner, N.: Z3: An efficient SMT solver. In: Proc. of TACAS'08. pp. 337–340. Springer-Verlag (2008)

[76] Mrazek, V., Hanif, M.A., et al.: AutoAx: An automatic design space exploration and circuit building methodology utilizing libraries of approximate components. In: Proc. of DAC'19. pp. 1–6. ACM (2019)

[77] Mrazek, V., Hrbacek, R., et al.: EvoApprox8b: Library of approximate adders and multipliers for circuit design and benchmarking of approximation methods. In: Proc. of DATE'17. pp. 258–261. EDAA (2017)

[78] Mrazek, V., Sarwar, S.S., et al.: Design of power-efficient approximate multipliers for approximate artificial neural networks. In: Proc. of ICCAD'16. pp. 81:1–81:7. ACM (2016)

[79] Najm, F.N.: A survey of power estimation techniques in VLSI circuits. IEEE Transactions on Very Large Scale Integration (VLSI) Systems **2**, 446–455 (1994)

[80] Nepal, K., Hashemi, S., et al.: Automated high-level generation of low-power approximate computing circuits. IEEE Transactions on Emerging Topics in Computing **7**, 18–30 (2019)

[81] Nepal, K., Li, Y., Bahar, R.I., Reda, S.: ABACUS: A technique for automated behavioral synthesis of approximate computing circuits. In: Proc. of DATE'14. pp. 1–6. IEEE (2014)

[82] Parhami, B.: Computer Arithmetic: Algorithms and Hardware Designs. Oxford University Press, Inc. (1999)

[83] Pohlert, T.: The Pairwise Multiple Comparison of Mean Ranks Package (PMCMR) (2014), R package

[84] Qiqieh, I., Shafik, R., et al.: Energy-efficient approximate multiplier design using bit significance-driven logic compression. In: Proc. of DATE'17. pp. 7–12. EDAA (2017)

[85] Ramasamy, M., Narmadha, G., Deivasigamani, S.: Carry based approximate full adder for low power approximate computing. In: Proc. of ICSCC'19. pp. 1–4. IEEE (2019)

[86] Ranjan, A., Raha, A., et al.: ASLAN: Synthesis of approximate sequential circuits. In: Proc. of DATE'14. pp. 1–6. IEEE (2014)

[87] Reda, S., Muhammad, M.: Approximate Circuits – Methodologies and CAD. Springer (2019)

[88] Riazanov, A., Voronkov, A.: The design and implementation of VAMPIRE. AI Communications **15**, 91–110 (2002)

[89] Ruiz, A.L., Morales, E.C., et al.: Algebraic Circuits, pp. 159–215. Springer-Verlag (2014)

[90] Samadi, M., Jamshidi, D.A., et al.: Paraprox: Pattern-based approximation for data parallel applications. In: Proc. of ASPLOS'14. pp. 35–50. ACM (2014)

[91] Samadi, M., Lee, J., et al.: SAGE: Self-tuning approximation for graphics engines. In: Proc. of MICRO'13. pp. 13–24. ACM (2013)

[92] Sayed-Ahmed, A., Große, D., et al.: Formal verification of integer multipliers by combining Grobner basis with logic reduction. In: Proc. of DATE'16. pp. 1048–1053. IEEE (2016)

[93] Scarabottolo, I., Ansaloni, G., et al.: Approximate logic synthesis: A survey. Proceedings of the IEEE **108**, 2195–2213 (2020)

[94] Schlachter, J., Camus, V., et al.: Design and applications of approximate circuits by gate-level pruning. IEEE Transactions on Very Large Scale Integration (VLSI) Systems **25**, 1694–1702 (2017)

[95] Sekanina, L., Vasicek, Z., Mrazek, V.: Automated search-based functional approximation for digital circuits. In: Approximate Circuits, pp. 175–203. Springer (2019)

[96] Shafique, M., Ahmad, W., et al.: A low latency generic accuracy configurable adder. In: Proc. of DAC'15. pp. 86:1–86:6. ACM (2015)

[97] Shafique, M., Hafiz, R., et al.: Invited: Cross-layer approximate computing: From logic to architectures. In: Proc. of DAC'16. pp. 1–6. IEEE (2016)

[98] Shim, B., Sridhara, S.R., Shanbhag, N.R.: Reliable low-power digital signal processing via reduced precision redundancy. IEEE Transactions on Very Large Scale Integration (VLSI) Systems **12**, 497–510 (2004)

[99] Shoushtari, M., BanaiyanMofrad, A., Dutt, N.: Exploiting partially-forgetful memories for approximate computing. IEEE Embedded Systems Letters **7**, 19–22 (2015)

[100] Sidiroglou-Douskos, S., S. Misailovic, S., et al.: Managing performance vs. accuracy trade-offs with loop perforation. In: Proc. of ESEC/FSE'11. pp. 124–134. ACM (2011)

[101] Soeken, M.: Cirkit (version 3). https://github.com/msoeken/cirkit (2023), online; March 2023

[102] Soeken, M., Amarù, L.G., et al.: Exact synthesis of majority-inverter graphs and its applications. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **36**, 1842–1855 (2017)

[103] Soeken, M., De Micheli, G., Mishchenko, A.: Busy man's synthesis: Combinational delay optimization with SAT. In: Proc. of DATE'17. pp. 830–835. IEEE (2017)

[104] Soeken, M., Große, D., Chandrasekharan, A., Drechsler, R.: BDD minimization for approximate computing. In: Proc. of ASP-DAC'16. pp. 474–479. IEEE (2016)

[105] Soeken, M., Haaswijk, W., et al.: Practical exact synthesis. In: Proc. of DATE'18. pp. 309–314. IEEE (2018)

[106] Sorensson, N., Een, N.: MiniSat v1.13 – a SAT solver with conflict-clause minimization. In: Proc. of SAT'05. pp. 1–2. Springer (2005)

[107] de Souza, L.A.M., da Silva, J.E.H., et al.: A benchmark suite for designing combinational logic circuits via metaheuristics. Applied Soft Computing **91**, 1–12 (2020)

[108] Sutherland, M., Miguel, J.S., Jerger, N.E.: Texture cache approximation on GPUs. In: Proc. of WAX'15. pp. 1–3 (2015)

[109] Synopsys design compiler. https://www.synopsys.com/, online; March 2023

[110] Thathachar, J.: On the Limitations of Ordered Representations of Functions, pp. 232–243. Springer Berlin Heidelberg (1998)

[111] Turner, A., Miller, J.F.: Neutral genetic drift: an investigation using Cartesian Genetic Programming. Genetic Programming and Evolvable Machines **16**, 531–558 (2015)

[112] Vasicek, Z.: Cartesian GP in optimization of combinational circuits with hundreds of inputs and thousands of gates. In: Proc. of EuroGP'15. pp. 139–150. Springer (2015)

[113] Vasicek, Z., Mrazek, V., Sekanina, L.: Evolutionary functional approximation of circuits implemented into FPGAs. In: Proc. of SSCI'16. pp. 1–8. IEEE (2016)

[114] Vasicek, Z., Mrazek, V., Sekanina, L.: Towards low power approximate DCT architecture for HEVC standard. In: Proc. of DATE'17. pp. 1576–1581. EDAA (2017)

[115] Vasicek, Z., Sekanina, L.: Formal verification of candidate solutions for post-synthesis evolutionary optimization in evolvable hardware. Genetic Programming and Evolvable Machines **12**, 305–327 (2011)

[116] Vasicek, Z., Sekanina, L.: Evolutionary design of approximate multipliers under different error metrics. In: Proc. of DDECS'14. pp. 135–140. IEEE (2014)

[117] Vasicek, Z., Sekanina, L.: How to evolve complex combinational circuits from scratch? In: Proc. of ICES'14. pp. 133–140. IEEE (2014)

[118] Vasicek, Z., Sekanina, L.: Circuit approximation using single- and multi-objective Cartesian GP. In: Proc. of EuroGP'15. pp. 217–229. Springer (2015)

[119] Vasicek, Z., Sekanina, L.: Evolutionary approach to approximate digital circuits design. Transactions on Evolutionary Computation **19**, 432–444 (2015)

[120] Vasicek, Z., Sekanina, L.: Evolutionary design of complex approximate combinational circuits. Genetic Programming and Evolvable Machines **17**, 169–192 (2016)

[121] Vasicek, Z., Slany, K.: Efficient phenotype evaluation in Cartesian Genetic Programming. In: Proc. of EuroGP'12. pp. 266–278. Springer-Verlag (2012)

[122] Vassilev, V.K., Job, D., Miller, J.F.: Towards the automatic design of more efficient digital circuits. In: Proc. of EH'00. pp. 151–160. IEEE (2000)

[123] Vassilev, V.K., Miller, J.F.: The advantages of landscape neutrality in digital circuit evolution. In: Proc. of ICES'00. pp. 252–263. Springer (2000)

[124] Venkataramani, S., Roy, K., Raghunathan, A.: Substitute-and-simplify: a unified design paradigm for approximate and quality configurable circuits. In: Proc. of DATE'13. pp. 1367–1372. EDAA (2013)

[125] Venkataramani, S., Sabne, A., et al.: SALSA: systematic logic synthesis of approximate circuits. In: Proc. of DAC'12. pp. 796–801. ACM (2012)

[126] Venkatesan, R., Agarwal, A., Roy, K., Raghunathan, A.: MACACO: Modeling and analysis of circuits for approximate computing. In: Proc. of ICCAD'11. pp. 667–673. ACM (2011)

[127] Verma, A.K., Brisk, P., Ienne, P.: Variable latency speculative addition: A new paradigm for arithmetic circuit design. In: Proc. of DATE'08. pp. 1250–1255. IEEE (2008)

[128] Wallace, C.S.: A suggestion for a fast multiplier. IEEE Transactions on Electronic Computers **EC-13**, 14–17 (1964)

[129] Wolf, C.: Yosys open synthesis suite. https://yosyshq.net/yosys/, online; March 2023

[130] Xu, Q., Mytkowicz, T., Kim, N.S.: Approximate computing: A survey. IEEE Design and Test **33**, 8–22 (2016)

[131] Yoo-Joo, J., Lim, D., et al.: 6.4 a 56gb/s 7.7mw/gb/s PAM-4 wireline transceiver in 10nm FinFET using MM-CDR-based ADC timing skew control and low-power DSP with approximate multiplier. In: Proc. of ISSCC'20. pp. 122–124. IEEE (2020)

[132] Yu, C., Ciesielski, M.: Analyzing imprecise adders using BDDs – a case study. In: Proc. of ISVLSI'16. pp. 152–157. IEEE (2016)

[133] Yu, T., Miller, J.F.: Neutrality and the evolvability of Boolean function landscape. In: Proc. of EuroGP'01. pp. 204–217. Springer (2001)