UNIVERSITY OF OKLAHOMA

GRADUATE COLLEGE

MULTIPLE REAL-CONSTANT MULTIPLICATION FOR COMPUTATIONALLY EFFICIENT IMPLEMENTATION OF DIGITAL TRANSFORMS

A DISSERTATION

SUBMITTED TO THE GRADUATE FACULTY

in partial fulfillment of the requirements for the

Degree of

DOCTOR OF PHILOSOPHY

By

MATTHEW BRANDON GATELY
Norman, Oklahoma
2012

MULTIPLE REAL-CONSTANT MULTIPLICATION FOR COMPUTATIONALLY EFFICIENT IMPLEMENTATION OF DIGITAL TRANSFORMS

A DISSERTATION APPROVED FOR THE
SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING

BY

Dr. Mark B. Yeary, Chair

Dr. Choon Yik Tang

Dr. Monte P. Tull

Dr. John K. Antonio

Dr. Yan Zhang

Acknowledgements

- First, I'd like to thank God for giving me the strength to persevere this research and produce this dissertation.

- Second, I thank my wife, Trisha, for standing beside me throughout this entire experience.

- Third, I appreciate the advise and encouragement that all of my doctoral committee members provided.

- Of course, I couldn't participate in this research without the funding provided by the National Science Foundation and the College of Engineering's Robert Hughes Centennial Fellowship.

- Last, I sincerely appreciate my family and friends for their unceasing prayers.

Contents

List of Tables

List of Figures

List of Algorithms

Abstract

THE NEED to multiply signals by vectors (or matrices) of *constants* is fundamental and frequently arises in many areas of electrical and computer engineering. In their hardware implementations, performance issues such as circuit area, delay, and power consumption heavily influence the design process. It is well known that multiplication of a signal by a constant can be implemented *multiplierless* as a network of shifts and additions, and that these computational networks, termed *shift-add* networks, can lend to higher performing circuit implementations than when using general multipliers. There is a rich body of work on the optimization of shift-add networks, known as the multiple constant multiplication (MCM) problem. However, the optimization strategies that have been developed for the MCM traditionally assume that the vector multiplications being optimized always stem from integer constants. This assumption breaks down for many real-world applications, where the target constants for MCM optimization are real numbers rather than integers. In these situations there is flexibility in how constants are quantized in digital circuits that can be leveraged. Thus, it is desirable to have a method of jointly optimizing both the constant quantization error and the shift-add network simultaneously.

This dissertation addresses this need by providing a problem framework and algorithms for joint quantization/MCM optimization and, through a series of experiments, shows that there is a potential for tremendous benefit when the optimization of quantization and shift-add networks is executed in one unified problem framework. After reviewing the relevant work, this dissertation rigorously develops the aforementioned joint optimization framework, describing the metrics used for quantization error, and culminates in a formal problem statement. We call this joint optimization problem

the multiple real-constant multiplication (MRCM) problem in order to distinguish it from the traditional MCM problem that operates exclusively with integer constants, which we hereafter refer to as the multiple integer-constant multiplication (MICM) problem. Then, we consider three different cost models used for evaluating shift-add networks and, with each model, we determine the potential advantages of using our MRCM framework over the traditional MICM approach.

First, we consider the traditional *adder-count* cost model. We start by formally defining the MRCM problem in the context of this cost model, and then describe a series of theoretical developments centered around finitizing and pruning the search space, leading to an efficient algorithm for solving the problem. Next, via extensive randomized experiments, we show that our joint framework leads to a reduction on the number of adders by 15%–60% on moderate size problems. In particular, for vectors of arbitrary constants, we show a possibility for 20%–60% reduction with less than 10% vector approximation error for both frameworks, whereas for vectors of low-pass filter coefficients, a 15%–30% reduction is possible without exceeding 10% error in frequency response.

Second, we consider an *adder-bits* cost model, whereby instead of simply counting the number of adders, we compute the combined bitwidth of all the adders.To solve the MRCM problem in this context, we introduce two search algorithms—one greedy and one optimal, each guided by a novel MRCM-aware heuristic. Next, we discuss a randomized experiment, in which we compare both algorithms to an MICM-targeted heuristic. We observe that the greedy search finds solutions with an average cost improvement of 13% over the MICM solution with the trials considered, and the optimal search finds an additional improvement of 6%.

Third, we consider a prominent *gate-level* cost model from the literature that.This gate-level model consider the bitwidths of an adder's inputs and output along with the relative alignment of the inputs/output due to bit shifting, when computing the

adder cost. To solve the MRCM problem in this context, a novel greedy algorithm is developed that uses a *functional programming* approach to solving the MRCM problem. Next, we experimentally show this algorithm to offer an improvement of up to 18%, over a competing MICM algorithm, on small instances having 20 8-bit constants, increasing to up to 59% improvement on larger instances having 80 5-bit constants.

Finally, we conclude the work by offering recommendations for possible future work in the development of efficient MRCM algorithms and novel problem formulations for optimizing MCM circuits.

Chapter 1

Introduction

1.1 Multiple Constant Multiplication

THE NEED to multiply signals by vectors (or matrices) of constants is fundamental and frequently arises in many areas of engineering. For example, the need arises in:

- linear time-invariant digital filters including finite and infinite impulse response (FIR and IIR) filters implemented using transposed structures [1, 2];
- implementation of digital control systems including state-space-based controllers and Kalman filters [3];
- implementation of digital communication systems including multiple-antenna transmitters and receivers [4];
- transformation optics [5];
- video coding [6];
- radar signal processing [7–9];
- and any other application employing constant linear transformations.

It is well known that multiplication of a signal by a constant can be implemented *multiplierless* as a network of shifts, additions, and subtractions, when the signal and constant are represented as fixed-point numbers. Furthermore, when multiplying a

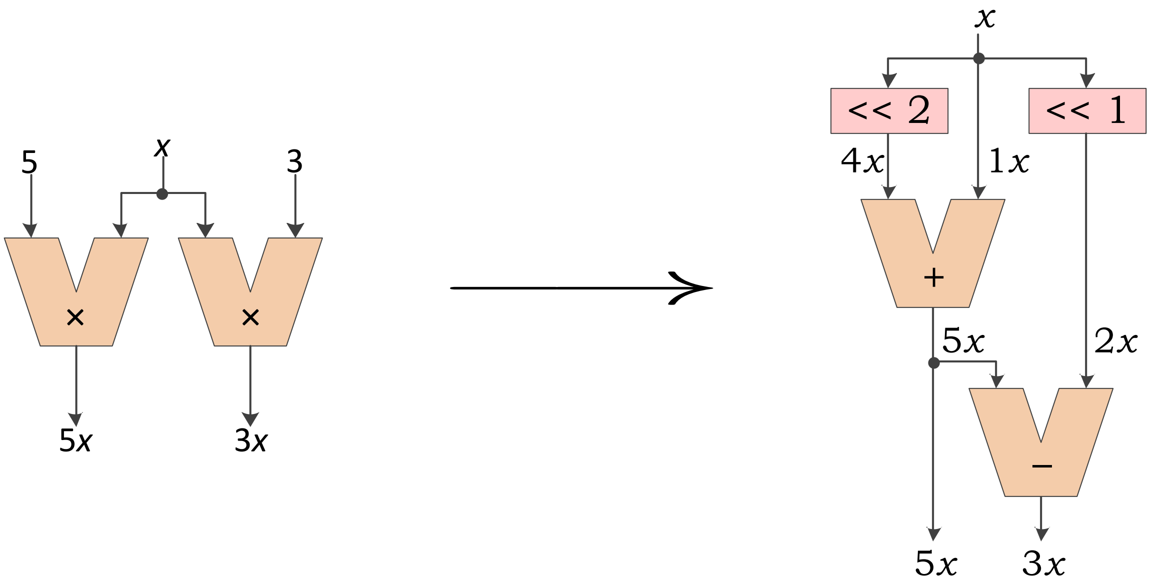


Figure 1.1: Illustration of multiplierless multiplication.

signal by a *vector* of constants, these additions and subtractions may be shared to further reduce computational resources, as shown in Figure 1.1.

These computational networks, termed *shift-add* networks, lend to more efficient computation in both hardware and software systems. In hardware, multiplierless multiplication leads to reduced area, latency, throughput, and power [10–12]. In software, multiplierless multiplication may be used when there is no multiply instruction available and, in general, may lead to reduced cycle counts as compared to using multiply instructions [11, 13].

Because multiplierless multiplication by constants is so prevalent, there is a rich body of work on the optimization of the aforementioned shift-add networks. In fact, the so-named *multiple constant multiplication* (MCM) problem, first introduced by Potkonjak *et al.* [14], has been rigorously studied over the last two decades [10, 13–22]. Although the precise MCM problem statement differs slightly depending on the authors within the relevant literature, the basic idea is to find a shift-add network implementing the multiplication using an optimal arrangement of additions and subtractions.

1.2 Typical Design Work-Flow

Though benefits of multiplierless multiplication exist in both hardware and software, this dissertation limits its discussion to the context of hardware, especially application-specific integrated circuits (ASICs). Consider an application where multiplication of a signal by a vector of *target constants* is required. When employing MCM optimizations, the design process can be thought to have four stages [23, 24]:

1. **Quantization**—the conversion from real infinite-precision constants to acceptable finite-precision approximations

2. **MCM**—the choice of intermediate constants to be computed in the shift-add network performing the multiplication

3. **Interconnect**—the choice and arrangement of shifts and additions/subtractions that connect intermediate and target constants into a realizable shift-add network

4. **Synthesis**—realization of a shift-add network into a gate-level ASIC design

Note that in stages 2 and 3 the term *intermediate constants* refers to those internal signals in a shift-add network that are computed as part of the network but do not correspond to any of the application's target constants.

Furthermore, note that the work of the four stages might be intermixed in a particular design work-flow and need not be treated independently.

1.3 Shortcomings of MCM Research

While the research and development of the MCM problem has been quite extensive, there are a number of shortcomings which stem from the following observation.

Most MCM literature up to this point has made the assumption that the applications involving shift-add networks exclusively include integer target constants. This assumption breaks down for many real-world applications, where the target constants for MCM optimization are real numbers rather than integers. In these cases, it is beneficial to have a method of jointly optimizing both the constant quantization error and the shift-add network simultaneously. Later sections will expand on this major shortcoming and other drawbacks through canonical examples and experimental results.

This dissertation is devoted to addressing this void in the literature. To do so, we define a general class of joint optimization problems, develop a collection of algorithms for solving them, analyze their theoretical properties, compare their efficacy with competing alternatives, and report on the experimental results.

1.4 Original Contributions

More specifically, the original contributions of this dissertation include:

- A formal general definition of the joint optimization problem, which we call the multiple real-constant multiplication (MRCM) problem;

- Three specific problem formulations, using varying cost models and error metrics, along with algorithms for solving them:

    1. The MRCM problem with *adder-count* cost model along with a series of theoretical developments leading to an efficient algorithm called JOINT_SOLVE,

    2. The MRCM problem with *bit-count* cost model along with two algorithms: *greedy* and *optimal*, with both algorithms' searches guided by a novel heuristic, and

    3. The MRCM problem with *gate-level* cost model along with a functional programming implementation called GREEDY_SOLVE.

- Empirical experiments designed to verify the efficacy of each algorithm and compare with the state of the art, with results showing:

    1. *Adder-count cost model*: 20%–60% adder reduction for vectors of arbitrary constants and 15%–30% adder reduction for vectors of low-pass filter coefficients,

    2. *Bit-count cost model*: the greedy search having an average cost improvement of 13% over the multiple integer-constant multiplication (MICM) solutions, and the optimal search having an additional 6% improvement,

    3. *Gate-level cost model*: the algorithm having an improvement of up to 18% over a competing MICM algorithm, on small instances having 20 8-bit constants, increasing to up to 59% improvement on larger instances having 80 5-bit constants.

Also, the work of this dissertation has led to the following publications:

[1] M. B. Gately, M. B. Yeary, and C. Y. Tang, "Reduced-hardware digital filter design via joint quantization and multiple constant multiplication optimization," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process.*, May 2011, pp. 4368–4371.

[2] ——, "Multiple real-constant multiplication with improved cost model and greedy and optimal searches," in *Proc. IEEE Int. Symp. Circuits Syst.*, May 2012.

[3] ——, "An algorithm for jointly optimizing quantization and multiple constant multiplication," *ACM Trans. Design Automation Electron. Syst.*, accepted and to appear, 2012.

Additionally, Chapter 6 discusses a preliminary algorithm and results of ongoing research for which another journal publication undergoing preparation.

1.5 Dissertation Outline

The outline of this dissertation is as follows: First, we discuss the current MCM literature in Chapter 2, devising the term multiple integer-constant multiplication (MICM) to refer to the traditional assumption that target constants are integers. Then, Chapter 3 introduces the MRCM problem as an alternative joint optimization formulation that considers non-integer target constants. Next, Chapter 4 evaluates the MRCM problem with regards to the traditional *adder-count* cost model. Next, Chapter 5 evaluates the problem in conjunction with an alternative *bit-count* cost model. Chapter 6 evaluates the problem using a *gate-level* cost model. Last, Chapter 7 summarizes the findings of this dissertation's research.

1.6 Notation and Terminology

In this dissertation, unless otherwise specified, the following notational conventions are employed:

a) Scalars are represented in lower case in an italic font, e.g., a.

b) Vectors and tuples are represented in lower case in a bold font with their components italicized and identified by subscripts, e.g., $\mathbf{a} = (a_0, \ldots, a_{N-1})$, where N is the length/dimensionality of the vector.

c) Sets are represented in upper case in a bold font, e.g., $a \in \mathbf{A}$. Sets assume no ordering unless it is explicitly stated.

d) Members of ordered sets are identified by subscripts, e.g., a set of scalars $\mathbf{A} = \{a_0, \ldots, a_{M-1}\}$, where M is the cardinality of the set.

Chapter 2

Related Work

In this chapter we discuss the multiple constant multiplication (MCM)-related literature to date. In Section 2.1 the *multiple integer-constant multiplication* problem is defined, which encapsulates most of the current MCM work. Then, Section 2.2 gives some background of the MCM problem as it has been applied to finite impulse response (FIR) digital filters.

2.1 The Multiple Integer-Constant Multiplication Problem

In most of the MCM literature to-date, the authors define the MCM problem under the assumption that all candidate applications for shift-add network optimization deal exclusively with constants that are integers (fixed-point numbers), and that there is no tolerance for error. However, since these assumptions only satisfy a subset of the applications that can be optimized with MCM methods, we give this limited problem definition the name of *multiple integer-constant multiplication* (MICM) in order to distinguish it from *multiple real-constant multiplication* to be defined later.

2.1.1 General Problem Formulation

Each researcher in the field of MICM uses his or her unique formal problem definition of the MICM problem. However, apart from semantical differences, their definitions generally result in the same basic problem formulation. Furthermore, some basic terminology has arisen in the field that is shared by most. Borrowing from the common terminology of papers written by prominent authors in the field [13,15,21,24], we now provide a general definition of the MICM problem:

> Let the term *fundamentals* denote the positive odd integers. Now, consider a candidate application for MCM optimization, that is, a system where an input signal is to be multiplied by a set of fundamentals. Let this set be called the *target fundamentals* $\mathbf{T} = \{f_0, \ldots, f_{M-1}\}$. Let a primitive *shift-add operation* be made up of a two-input adder with possible arithmetic shifts at the inputs and output. Starting with the original input x, find a sequence of shift-add operations that can be successively applied to the input signal in order to compute the circuit outputs $\{f_0 \cdot x, \ldots, f_{M-1} \cdot x\}$. In order to produce all of the circuit outputs, it may be necessary to compute other *intermediate* signals that are not needed for the output, resulting in extra shift-add operations. With this in mind, an optimal solution to the MICM problem is a sequence of shift-add operations that compute all of the outputs while minimizing the total number of operations.

While the previous paragraph is understood as a general definition of the MICM problem, there have been proposed modifications to the problem which are discussed next, along with different approaches to solving the problem.

2.1.2 Variations and Solving Methods

Various methods for solving the MICM problem have been studied, but, in general, the methods can be divided into two categories—common subexpression elimination (CSE)-based algorithms and graph-based algorithms. For example, the authors of [10, 19, 20, 22] define the problem as a form of CSE that the authors of [19, 20, 22] then transform into an integer-linear programming problem, while others [13, 15, 17, 21] address the MICM problem using a graph-based approach. In [15], the authors compare graph-based methods and CSE-based methods and show that graph-based methods allow for more optimal solutions to the MICM problem because the solutions are not affected by the choice of number representation (e.g., binary, canonical-signed-digits).

Variations on the general MICM problem have been proposed also, with some MCM-solving algorithms incorporate constraints on the logic depth of the MCM circuit [13, 16, 19] or the size of the adders [13] as well.

2.1.3 Gate-Level Cost Model

Although, the MCM problem traditionally optimizes the *total number* of shift-add operations, there has been research into lower-level cost models as well. For instance, to more accurately represent the hardware characteristics of an MCM solution, Aksoy et al. [25, 26] and Johansson et al. [24, 27] investigate the hardware cost of shift-add networks at the gate(bit)-level. In doing so, they characterize all of the different adder configurations that can occur in a shift-add network, and for each one compute the number of 1-bit full adders and half-adders that each configuration will produce. To verify the validity of their models, in the results of [24–27], they show that such gate-level cost models do indeed correspond more directly to synthesized hardware cost than the traditional cost model.

Hence, gate-level cost models provide a much more fine-grained cost evaluator for MCM problems than the traditional course-grained approach of simply counting the total number of adders, without respect to bit-level details. Referring back to the four-stage MCM application work-flow of Section 1.2, it can be said that using a gate-level cost model provides a means of more tightly coupling the stages of 2) MCM and 3) Interconnect. The specifics of a gate-level cost model are discussed in more detail in Chapter 6.

2.2 Joint Quantization and MCM in the Limited Context of Finite Impulse Response Filters

Because of their usefulness, finite impulse response (FIR) digital filters have been studied in depth, with papers typically addressing one of two distinct goals: to closely match frequency response characteristics, or to realize the filters optimally in hardware. In FIR filter design, these two goals often compete against each other, so methods improving one will generally adversely affect the other. However, they are often viewed as separate goals and are studied, one without respect to the other.

First, to design realizable FIR filters, one has to consider the finite wordlength effects (i.e., quantization errors). Namely, ideal filters have to be approximated by causal filters having a finite number of coefficients, each with a finite wordlength. Figure 2.1 illustrates such a filter. Much progress has been made in minimizing the effects of these filter approximation limitations (e.g., [28–33]). For instance, McClellan *et al.* [29] developed an algorithm that generates an N-tap filter which approximates a given frequency response with the least error in the minimax sense. However, the generated filter still has infinite-wordlength coefficients and is therefore unrealizable with finite-wordlength coefficients. As another example, Ito *et al.* [33] presented a method for quantizing the coefficients of an N-tap infinite-wordlength filter. Given a total number of quantization bits, this method quantizes the coefficients in such a way as to minimize the maximum error of the resulting frequency response.

Second, the algorithms developed for the MICM problem that are discussed in Section 2.1 are directly applicable to the FIR filter structure, once the coefficients have been quantized into integers. For example, Figure 2.1 shows that the common FIR filter, when implemented in a transposed direct form, includes an MCM operation, since the circuit input is multiplied by all the coefficients simultaneously.

Even though the goals of reducing quantization error and optimizing the hardware

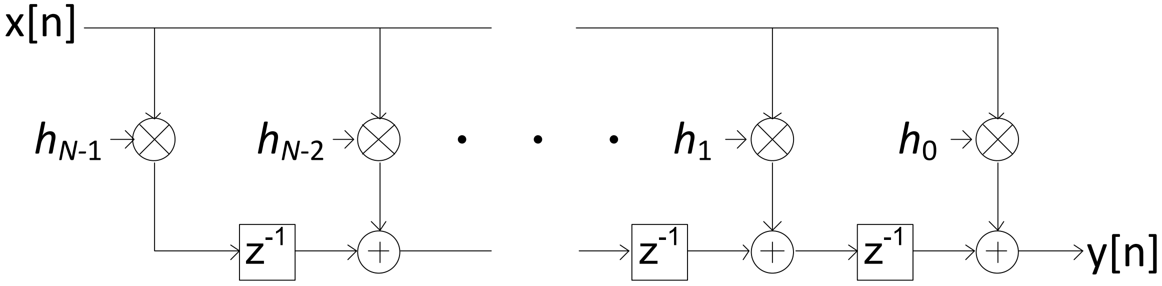


Figure 2.1: An N-tap FIR digital filter with coefficients $h_0, \ldots, h_{N-1}$, implemented in a transposed direct form.

circuit can be handled separately, it is conceivable that solving both of the problems in a *joint* framework would allow for more satisfactory solutions with respect to the competing filter design goals. For this reason, algorithms employing various joint optimization strategies have been developed in the limited context of FIR filters [12, 23, 24, 34–40]. With these algorithms, the filter coefficient vectors considered are those which satisfy a set of frequency response specifications on, for instance, passband and stopband cutoff frequencies and ripple tolerances. Moreover, although they are constructed based on different approaches, these algorithms are capable of exploiting the specific structure of FIR filters, leading to computationally efficient implementations. Such features are important, since FIR filter design remains a central aspect of signal processing applications.

Chapter 3

The Multiple Real-Constant Multiplication Problem

In this chapter the *multiple real-constant multiplication* (MRCM) problem is introduced to address the drawbacks of the traditional MICM problem. In Section 3.1, these drawbacks are discussed to motivate the development of the MRCM problem that is studied in this dissertation. Then, Section 3.2 discusses possible quantization error metrics to be used in MRCM problem formulations. Next, Section 3.3 develops some terminology to be used throughout this dissertation and provides a formal statement of the MICM problem. Finally, Section 3.4 further develops the terminology and provides a formal MRCM problem statement.

3.1 Rationale and Motivation

As discussed in Section 2.1, the traditional literature relating to the MCM problem makes certain assumptions about target applications. Namely, that in a given system designed using MCM optimization, the target constants are all integers (fixed-point numbers). With this assumption in mind, the traditional MCM problem formulations and algorithms are developed independent of quantization considerations. Accordingly, we have devised the name multiple integer-constant multiplication (MICM) to refer to the MCM problem under this assumption.

However, this assumption of integer constants breaks down for a system design that deals with real, non-integer constants that need to be quantized before they are suitable for multiplierless multiplication optimization. In this case, the quantization and MCM stages of the system design must be optimized independently. The drawback to this disjointed approach of system design is that a loss of freedom is incurred that might lead to a less ideal solution than could be obtained by using a joint optimization approach. For instance, there is freedom in choosing how to quantize non-integer constants, and the manner that they are quantized determines the resulting approximation error as well as the opportunity that exists for MCM optimization [41]. Consider an example where the fixed-point bitwidth is given. In this case, quantizing to minimize approximation error may generate fixed-point constants that are not well-suited for MCM optimization. On the other hand, fixed-point constants that lead to an optimal shift-add network may have an approximation error that exceeds application constraints. Furthermore, in [41, 42], we show that substantial hardware cost improvement can be achieved when using an MCM problem formulation that takes real constants and simultaneously solves for the best balanced solution in terms of quantization and adder reduction.

To address this need, we develop an optimization problem for *jointly* quantizing constants and solving and performing MCM network reduction. Hereafter, to distinguish it from the traditional MICM problem, we refer to our joint quantization/adder reduction problem as the *multiple real-constant multiplication* (MRCM) problem.

While various independent joint optimization methods have been developed in the *limited context* of FIR filters (see Section 2.2), in this dissertation we discuss the MRCM problem in a *general setting*, where one is given a vector of ideal, real constants, which can originate from any number of applications, and which is not necessarily associated with an FIR filter.

3.2 Quantization Error Metrics

When defining the MRCM problem with real constants, it becomes necessary to define an error metric to determine the quantized constants that are feasible for a given problem. In this section, we discuss several metrics that can be used for measuring quantization error. We consider metrics that use a scalar error tolerance ε as well as a metric that uses a vector of error tolerances $\boldsymbol{\varepsilon}$ that is the same size as the target constant vector. Both options have particular characteristics that might make them useful for a given MRCM problem. Next, each one is discussed in turn in Sections 3.2.1 and 3.2.2.

3.2.1 Scalar Error Metric and p-norm

It is often useful to be able to define the error tolerance of a quantized constant vector using a single scalar quantity ε. In these cases, p-norms provide natural error functions by accumulating the errors of each individual constant into one measure.

Definition 3.1 (p-norm)**:** Given a positive integer p, the p-norm is computed as $\|\mathbf{x}\|_p = (\sum_{i=0}^{N-1} |x_i|^p)^{\frac{1}{p}}$.

Such metrics may be appropriate for applications where terms are to be summed up, so that the *total error* can be measured. In this work, we use the 1-norm for defining the MRCM problem in Chapters 4 and 6 for reasons to be seen next.

Let $\mathbf{h} = (h_0, h_1, \ldots, h_{N-1}) \in \mathbb{R}^N$ be a vector of ideal constants from a particular application, and let $\hat{\mathbf{h}} = (\hat{h}_0, \hat{h}_1, \ldots, \hat{h}_{N-1}) \in \mathbb{R}^N$ be a fixed-point vector obtained from some quantization of $\mathbf{h}$. To measure the quantization error, we use the 1-norm of $\mathbf{h} - \hat{\mathbf{h}}$ defined as

$$\|\mathbf{h} - \hat{\mathbf{h}}\|_1 = \sum_{n=0}^{N-1} |h_n - \hat{h}_n|. \tag{3.1}$$

Although the quantization error can also be measured using other norms such as the 2-norm or ∞-norm—in fact, they cause little change to our problem description and algorithm development—we will not pursue such alternatives here. We note that the 1-norm has also been used in [11] and [12] as the error metric when searching for computationally efficient fixed-point constants. In addition, it has a nice frequency domain interpretation: The FIR frequency response $H(\omega)$ is calculated from the discrete-time Fourier transform (DTFT) [43] of the filter coefficients $\mathbf{h} = (h_0, \ldots, h_{N-1}) \in \mathbb{R}^N$ as

$$H(\omega) = \sum_{n=0}^{N-1} h_n e^{-j\omega n}, \tag{3.2}$$

where $\omega \in \mathbb{R}$ is the frequency in radians/sample. The vector notation, h_n, is used instead of the signal processing sequence notation, $h[n]$, in order to maintain consistent notation throughout. Then from (3.1) and (3.2), the frequency response error is bounded by the 1-norm of $\mathbf{h} - \hat{\mathbf{h}}$, i.e., for all normalized frequencies ω,

$$\begin{aligned} \Big| \big|H(\omega)\big| - \big|\hat{H}(\omega)\big| \Big| &\leq \big| H(\omega) - \hat{H}(\omega) \big| \\ &= \left| \sum_{n=0}^{N-1} \Big(h_n - \hat{h}_n \Big) e^{-j\omega n} \right| \leq \sum_{n=0}^{N-1} \big| h_n - \hat{h}_n \big| = \| \mathbf{h} - \hat{\mathbf{h}} \|_1. \end{aligned} \tag{3.3}$$

Although conservative, (3.3) allows us to compute an upper bound on the frequency response error without having to compute any DTFT.

Additionally, note that when using a scalar error bound in applications such as filters where terms are summed together at the output of the MCM block, it is probably also appropriate for the quantized constants to have the same alignment of their fractional points. For this reason, in the MRCM problems defined in Chapters 4 and 6, the constants are all required to have the same power-of-two scale factor.

3.2.2 Vector Error Metric

Next, an alternative to having a scalar error metric is to have a vector error metric, appropriate for applications in which each constant has its own rigid error requirements. Given a constant vector $\mathbf{c} = (c_0, \ldots, c_{N-1})$, a quantized vector $\hat{\mathbf{c}} = (\hat{c}_0, \ldots, \hat{c}_{N-1})$, and an error vector $\boldsymbol{\varepsilon} = (\varepsilon_0, \ldots, \varepsilon_{N-1})$, a simple case of this might be the requirement that for all $n \in \{0, \ldots, N-1\}$, $|c_n - \hat{c}_n| \leq \mathcal{E}_n$. Algorithm implementatiosn involving vector error metrics are more simple, in general, because the error from each constant can be treated individually.

In addition, note that when using a vector error metric for applications where the constants can be treated disjointedly after the MCM block, it may be appropriate to allow each constant to have its own power-of-two scale factor. This allows more flexibility in the search space of quantized constant vectors, which is demonstrated in the MRCM problem of Chapter 5.

3.3 General MICM Problem Formulation

In this section, we formally define the MICM problem. To accomplish this task, we first provide several definitions to catalyze the problem definition. Furthermore, these definitions are to be used throughout the rest of this dissertation. While some of the definitions are similar to or the same as those definitions in the related literature (e.g., [10,13–22]), we devise the following definitions in a way that lends to our development of the formal MICM/MRCM problems and associated algorithms that follow.

Definition 3.2 (Fundamentals, $\mathbb{F}$)**:** Let $\mathbb{F}$ be the set of all odd positive integers, referred to as *fundamentals* [13, 16, 24]. Additionally, let $0 \in \mathbb{F}$ as a special case.

Note that we adhere to the traditional approach [18] of limiting integer constants to positive odd integers. This is without loss of generality, since all integer constants can be obtained by shifting positive odd constants and adjusting their signs at some other stage of computation.

Definition 3.3 ($\mathbb{Z}^+$)**:** Let $\mathbb{Z}^+$ be the set of nonnegative integers, i.e., $\{0, 1, 2, \ldots\}$.

To facilitate the MICM problem definition, a function is provided for converting any nonnegative integer into a corresponding fundamental:

Definition 3.4 ($\Phi(x)$)**:** Given a nonnegative integer $x \in \mathbb{Z}^+$, the *fundamentalization* of x, $\Phi(x)$ is the unique fundamental that can be obtained by repeatedly dividing x by 2 until odd. As a special case, let $\Phi(0) = 0$.

For instance, $\Phi(26) = 13$.

Definition 3.5 ($\mathcal{A}$-configuration)**:** Now, let an $\mathcal{A}$-*configuration* [13] be defined as a 3-tuple $(l_1, l_2, s) \in \mathbb{Z}^+ \times \mathbb{Z}^+ \times \{0, 1\}$.

Definition 3.6 ($\mathcal{A}$-operation)**:** Then we say that an $\mathcal{A}$-configuration $\mathbf{g} = (l_1, l_2, s)$ induces a corresponding function, termed an $\mathcal{A}$-*operation* [13] and defined as $\mathcal{A}_\mathbf{g}(u, v) = \Phi(2^{l_1}u + (-1)^s 2^{l_2} v)$, where $u, v \in \mathbb{F}$.

With this definition in mind, an $\mathcal{A}$-operation represents the basic shift and add/subtract operation used to build up shift-add networks. The parameters l_1 and l_2 determine the shifting of the inputs and s determines whether the operation is an addition or subtraction. Note that after adding or subtracting, the output is then fundamentalized as well.

Definition 3.7 ($\mathcal{A}$-relation)**:** Next, in order to encapsulate an instantiation of an $\mathcal{A}$-operation in a shift-add network with specific input and output signals, an $\mathcal{A}$-*relation* is defined as a 4-tuple $(u, v, \mathbf{g}, w)$ where $u, v, w \in \mathbb{F}$, $\mathbf{g}$ is an $\mathcal{A}$-configuration, and $w = \mathcal{A}_\mathbf{g}(u, v)$.

Because it allows later definitions to be written much more concisely and clearly, we devise a special notation here for accessing the $\mathcal{A}$-relation tuple's members by name. As an example of this notation, $\mathbf{r} = (\mathbf{r}.u, \mathbf{r}.v, \mathbf{r}.\mathbf{g}, \mathbf{r}.w)$.

In order to represent an entire shift-add network we now define a collection of $\mathcal{A}$-relations termed an *adder tree*. To facilitate the definition, we first define the *root* $\mathcal{A}$-*relation*, $\mathbf{1} = (1, 1, (0, 0, 0), 1)$. The root $\mathcal{A}$-relation simply serves as a placeholder for the initial input signal of a shift-add network, $1X$. Then, an *adder tree* is a collection of $\mathcal{A}$-relations rooted at $\mathbf{1}$, with the property that each $\mathcal{A}$-relation in the collection has inputs supplied by the outputs of some preceding $\mathcal{A}$-relation.

Definition 3.8 (adder tree)**:** More precisely, we define an *adder tree* to be an ordered set of $\mathcal{A}$-relations $\mathbf{T} = \{\mathbf{r}_0, \ldots, \mathbf{r}_{M-1}\}$, such that $\mathbf{r}_0 = \mathbf{1}$ and for all $k \in \{1, M-1\}$ there exists $i, j \in \{0, k-1\}$ with $\mathbf{r}_k.u = \mathbf{r}_i.w$ and $\mathbf{r}_k.v = \mathbf{r}_j.w$.

Definition 3.9 ($\mathbb{T}$)**:** Let $\mathbb{T}$ denote the set of all adder trees.

Now, given a set of fundamentals, we want to determine the set of adder trees that produce all the fundamentals as outputs.

Definition 3.10 (ψ)**:** Given $F \subset \mathbb{F}$, an adder tree $\mathbf{T} = \{\mathbf{r}_0, \ldots, \mathbf{r}_{M-1}\}$ is a member of $\psi(F)$ if and only if for all $f \in \mathbb{F}$ with $f \neq 0$, there exists an $m \in \{0, \ldots, M-1\}$ such that $\mathbf{r}_m.w = f$.

In Section 2.1, we presented the MICM problem in a traditional way in order to facilitate discussion of the related work presented in that chapter. However, using Definitions 3.2–3.10, the MICM problem can now be more rigorously defined as follows:

Problem 3.1 — General MICM problem: Given a set of fundamentals $F \in \mathbb{F}$, find an adder tree $\mathbf{T} \in \psi(F)$ that minimizes cost($\mathbf{T}$).

Note that Problem 3.1 is a general definition where the particular cost function is not specified. However, when formulating the precise problem statements of Sections 4.2, 5.2, and 6.2, the cost functions are specified accordingly.

The remaining problems introduced in Chapters 4–6 of this dissertation are defined in an analogous manner in order to maintain consistency throughout.

3.4 General MRCM Problem Formulation

As in the previous section, we now define the general MRCM problem, foregoing any particular cost model.

The MRCM problem involves the quantization of the real target constants into finite-bitwidth integers of some maximum bitwidth.

Definition 3.11 ($\mathbb{Z}_b$,$\mathbb{R}_b$)**:** With this in mind, for a given bitwidth b, let $\mathbb{Z}_b$ be the nonnegative b-bit integers, i.e., $\mathbb{Z}_b = \{0, \ldots, 2^b - 1\}$. Also, let $\mathbb{R}_b$ be the real numbers from the same range, specifically the open set $\mathbb{R}_b = (0, 2^b - 1)$.

To facilitate the MRCM problem formulation, a vector form of the fundamentalization function is defined:

Definition 3.12 ($\Phi(\mathbf{x})$)**:** Given a vector $\mathbf{x} \in \mathbb{Z}_b^N$, we define $\Phi(\mathbf{x})$ as the set union of the x_i *fundamentalized* component-wise, i.e., $\Phi(\mathbf{x}) = \cup_{i \in \{0,\ldots,N-1\}} \Phi(x_i)$.

For instance, $\Phi\big((4, 18, 0, 26, 8)\big) = \{1, 9, 0, 13\}$.

Using Definitions 3.11 and 3.12, we can now define the general MRCM problem as follows:

Problem 3.2 — General MRCM problem: Given a vector length $N \in \mathbb{N}$, an ideal constant vector $\mathbf{c} \in \mathbb{R}_b^N$, some error bound, and a number of bits $b \in \mathbb{N}$, find a finite-bitwidth constant vector $\hat{\mathbf{c}} \in \mathbb{Z}_b^N$, and an adder tree $\mathbf{T} \in \psi(\Phi(\hat{\mathbf{c}}))$ that minimize $\mathrm{cost}(\mathbf{T})$ subject to $\hat{\mathbf{c}}$ satisfying the error bound.

Like Problem 3.1, the cost and error functions are not specified in Problem 3.2 since it is a general problem statement. The precise MRCM problem statements are to follow as Problems 4.3, 5.1, and 6.1.

Chapter 4

Multiple Real-Constant Multiplication with Adder-Count Cost Model

4.1 Introduction

In this chapter, we evaluate the MRCM problem's efficacy when using the traditional *adder-count* cost model, that is, simply optimizing the total number of adders. We compare the MRCM problem using this cost model to the de facto disjointed approach using the MICM problem.

In particular, for vectors of arbitrary constants, we show that one could achieve a 20%–60% reduction with less than 10% vector approximation error for both frameworks, whereas for vectors of low-pass filter coefficients, a 15%–30% reduction is possible without exceeding 10% error in frequency response. We stress that the results of our joint framework are not directly comparable with those of the algorithms of [12, 23, 24, 34–40] discussed in Section 2.2, as the joint framework assumes general constants, whereas the latter assume FIR filter coefficients.

The outline of this chapter is as follows: First, in Section 4.2, we formulate the specific MRCM problem using this cost model, leading to a formal problem statement (Problem 4.3). Next, Section 4.3 provides a canonical example that motivates the rest of the chapter. Then, in Section 4.4 we describe a series of theoretical developments centered around finitizing and pruning the search space, leading to an efficient algorithm called JOINT_SOLVE for solving the problem, which represents the main contribution of this chapter. Next, in Section 4.5, we show, through two extensive randomized experiments, that the number of addition/subtraction operations can be reduced substantially using the joint framework and JOINT_SOLVE compared to a disjointed framework, where the quantization and MICM problems are handled separately. Finally, Section 4.6 concludes the chapter.

4.2 Problem Formulation

In this section, the MRCM problem using the adder-count cost model is defined. First, Section 4.2.1 defines the coefficient quantization problem. Next, Section 4.2.2 defines the traditional MICM problem with this cost model. Then, based on this setup, Section 4.2.3 defines the joint optimization problem.

4.2.1 Constant Quantization Problem and Error Metric

For the MCM problem definitions in this chapter, we use the 1-norm error metric as defined in Section 3.2.1. With the error metric (3.1) established, we can formally state the constant quantization problem, which is concerned with quantizing real constants into fixed-point constants with an acceptable amount of error:

Problem 4.1 — Constant quantization problem: Given a vector length $N \in \mathbb{N}$, an ideal constant vector $\mathbf{h} \in \mathbb{R}^N$, an error bound $\varepsilon > 0$, and a number of bits $m \in \{0\} \cup \mathbb{N}$, find a scale factor $s \in \mathbb{Z}$ and a fixed-point constant vector $\hat{\mathbf{h}} \in \mathbb{R}^N$ that minimize $\|\mathbf{h} - \hat{\mathbf{h}}\|_1$ subject to $|\hat{h}_n| \cdot 2^s \in \{0, 1, \ldots, 2^m - 1\}$ for all $n \in \{0, \ldots, N-1\}$ and $\|\mathbf{h} - \hat{\mathbf{h}}\|_1 \leq \varepsilon$.

Note that the 2^s scale factor is simply an indicator of where the fractional point belongs in the fixed-point constant vector and it does not correspond to any actual computation. Also, a solution to Problem 4.1 is an m-bit constant vector that satisfies the error constraint imposed by ε, which may not exist if ε is too restrictive for the given m, leaving no feasible $\hat{\mathbf{h}}$ vectors.

4.2.2 Traditional MICM Problem with Adder-Count Cost Model

Here we formulate the MCM problem in a way that lends itself easily to our joint problem definition in Section 4.2.3. We consider a variant of the problem that incorporates a constraint on the logic depth of the circuit [13, 16]. The traditional MCM problem formulation considers the shift-add operation to be the most primitive operation; shifts and additions can be combined as a single operation because shifts are implemented as simple wires in hardware, and may often be combined with additions in software instructions. Also, subtractions have essentially the same cost as additions, so a shift-add operation refers to a single addition/subtraction with possible shifts before and after. Moreover, we adhere to the traditional approach [18] of limiting integer constants to positive odd integers. This is without loss of generality, since all integer constants can be obtained by shifting positive odd constants and adjusting their signs at some other stage of computation. It is from this understanding that the MCM problem is formed. We borrow terminology (though not exact definitions) from [13, 15] to define the primitive shift-add operation:

Note that much of the problem and algorithm development in this chapter is based on deprecated terminology that has been replaced in more recent work, because the research of this chapter comes from an article that has been accepted for publication [42]. However, to reduce the possibility of introducing errors by rewriting every definition and developing the algorithm in a completely different manner, we maintain the same terminology of the journal version in lieu of the more recent terminology introduced in Chapter 3 and carried on in Chapter 6. As a result, we caution the reader to be aware of certain terms taking on new definitions in this chapter.

Definition 4.1 ($\mathcal{A}$-configuration)**:** An *$\mathcal{A}$-configuration* is a 4-tuple, $c = (l_1, l_2, r, t)$, where l_1, l_2, and r are nonnegative integers and $t \in \{0, 1\}$.

Definition 4.2 (F)**:** Let F be the set of all odd positive integers, referred to as

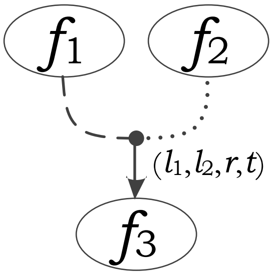


Figure 4.1: Graphical depiction of an $\mathcal{A}$-operation (f_1, f_2, c, f_3) with $c = (l_1, l_2, r, t)$.

fundamentals.

Definition 4.3 ($\mathcal{A}$-operation)**:** An $\mathcal{A}$-*operation* is a 4-tuple, (f_1, f_2, c, f_3), where $c = (l_1, l_2, r, t)$ is an $\mathcal{A}$-configuration and $f_1, f_2, f_3 \in F$, such that $f_3 = (2^{l_1} f_1 + (-1)^t 2^{l_2} f_2)2^{-r}$.

Definition 4.4 (R)**:** Let R be the set of all $\mathcal{A}$-operations.

An $\mathcal{A}$-operation (f_1, f_2, c, f_3) represents an operation induced by an $\mathcal{A}$-configuration $c = (l_1, l_2, r, t)$ with two input fundamentals in order to produce one output fundamental. The parameters l_1 and l_2 determine the left-shifting of the inputs f_1 and f_2, r determines the right-shifting of the output f_3, and t determines whether the $\mathcal{A}$-operation is an addition or a subtraction. Graphically, an $\mathcal{A}$-operation is depicted in Figure 4.1. $\mathcal{A}$-operations are the building blocks of adder trees which are defined next. Let $\mathcal{P}(S)$ denote the power set of a set S.

Definition 4.5 ($G_{R'}$)**:** Given a set of $\mathcal{A}$-operations $R' \in \mathcal{P}(R)$, let $G_{R'} = (V, E)$ be a directed graph induced by R' in the following manner:

(i) $V = \{1\} \bigcup \left(\bigcup_{(f_1,f_2,c,f_3) \in R'} \{f_1, f_2, f_3\} \right)$ and

(ii) $E = \bigcup_{(f_1,f_2,c,f_3) \in R'} \{(f_1, f_3), (f_2, f_3)\}$.

For instance, Figure 4.2 depicts such an induced graph. This definition, along with the one below, allows us to characterize the properties that make an R' useful for the MCM problem.

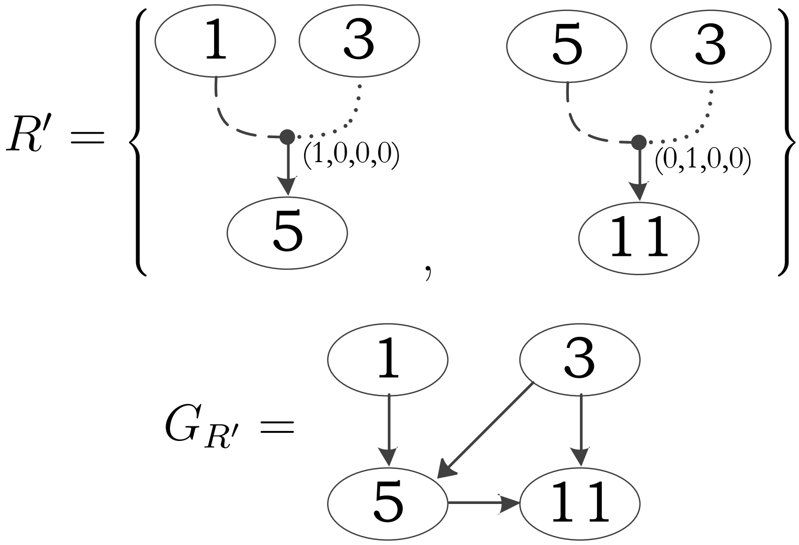


Figure 4.2: Graph $G_{R'}$ induced by R' according to Definition 4.5. Since $R' = \{(1, 3,\ (1, 0, 0, 0),\ 5),\ (5, 3,\ (0, 1, 0, 0),\ 11)\}$, $G_{R'} = (\{1, 3, 5, 11\},\ \{(1, 5),\ (3, 5),\ (5, 11),\ (3, 11)\})$.

Definition 4.6 (Adder tree)**:** A set $R' \in \mathcal{P}(R)$ is an *adder tree* if:

(i) the induced graph $G_{R'} = (V, E)$ has a basis of $\{1\}$ [44] (i.e., for every $v \in V$ where $v \neq 1$, there exists a directed path from 1 to v),

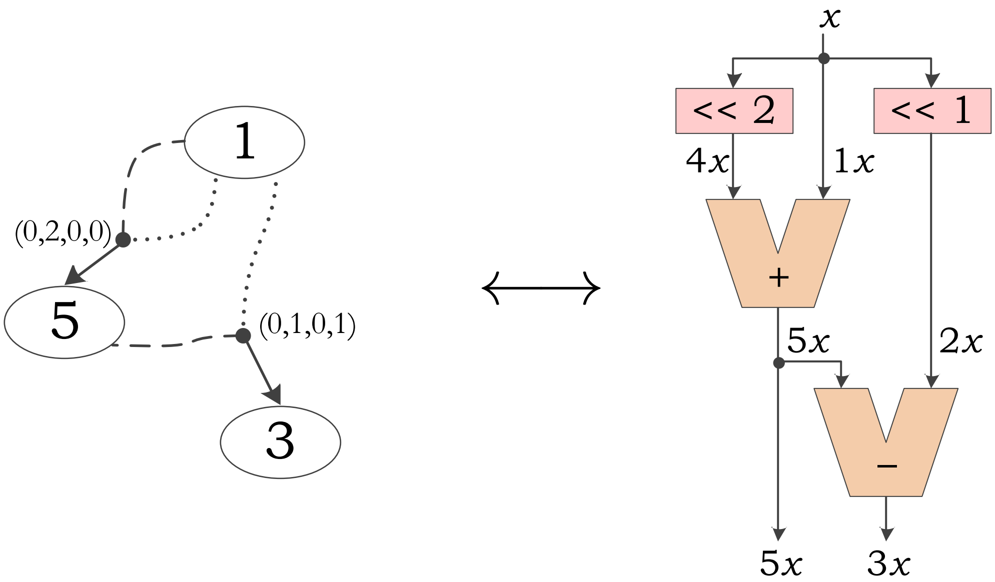
(ii) for all $(f_1, f_2, c, f_3) \in R'$, $f_3 \neq 1$, and

(iii) for any two distinct $(f_1, f_2, c, f_3), (f_1', f_2', c', f_3') \in R'$, $f_3 \neq f_3'$.

For example, in Figure 4.2, R' is not an adder tree because in $G_{R'}$ there is no directed path from node 1 to node 3. In contrast, Figure 4.3(a) depicts a legitimate adder tree. Note that (i)–(iii) in Definition 4.6 ensure that the graph is directed and acyclic, and that it has a one-to-one correspondence with a shift-add network for an MCM operation, as shown in Figure 4.3(b).

Having introduced the notion of adder trees, we next define the characteristics that make an adder tree suitable for the MCM problem.

Definition 4.7 ($\psi(F')$)**:** Given a set of fundamentals $F' \in \mathcal{P}(F)$, let $\psi(F')$ be the set of all adder trees R' such that the induced graph $G_{R'} = (V, E)$ satisfies $F' \subseteq V$.



(a) Depiction of the adder tree R'={(1,1,(0,2,0,0),5), (5,1,(0,1,0,1),3)}, with repeated fundamentals shown as the same node.

(b) Shift-add network representing the MCM computation, $x \cdot (3, 5)$.

Figure 4.3: An adder tree and its corresponding shift-add network. The two structures have a one-to-one correspondence.

For example, Figure 4.3(a) and Figure 4.4 are both members of $\psi(\{3, 5\})$. However, the R' of Figure 4.3(a) corresponds to more efficient computation than the R' of Figure 4.4 because it requires one less shift-add operation. On the other hand, the two adder trees require the same logic depth in their shift-add networks. Therefore, we define the following two metrics that characterize the number of shift-add operations and the logic depth of an adder tree:

Definition 4.8 (Adder tree cost)**:** Given an adder tree R', define its *cost* as $\text{cost}(R') = |R'|$.

Definition 4.9 (Adder tree depth)**:** Given an adder tree R', define its *depth*, $\text{depth}(R')$, as the length of the longest directed path starting from node 1 in the induced graph $G_{R'}$.

Note that it has been shown that if $R' \in \psi(F')$ then $\text{depth}(R') \geq \max_{f \in F'} \lceil \log_2 S(f) \rceil$, where $S(f)$ is the number of nonzero digits in the canonical-signed-digit representation

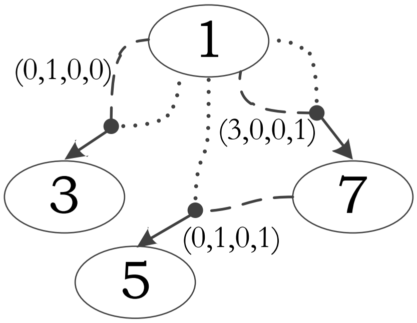


Figure 4.4: A suboptimal adder tree for the target fundamentals, $\{3, 5\}$. Figure 4.3(a) shows that the intermediate node 7 is not required.

of f [18].

With the above, we can now precisely define the MCM problem under a delay constraint:

Problem 4.2 — MICM problem with adder-count cost model: Given a set of fundamentals $F' \in \mathcal{P}(F)$ and a maximum depth $d \in \{0, 1, \ldots, \infty\}, d \geq \max_{f \in F'} \lceil \log_2 S(f) \rceil$, find an adder tree $R' \in \mathcal{P}(R)$ that minimizes $\text{cost}(R')$ subject to $R' \in \psi(F')$ and $\text{depth}(R') \leq d$.

For instance, if $F' = \{3, 5\}$ and d is at least 2, then Figure 4.3(a) shows an optimal adder tree with a cost of 2, while Figure 4.4 shows a suboptimal adder tree with a cost of 3. They both have a depth of 2. Note that a solution to Problem 4.2 always exists, because the given bound on $\text{depth}(R')$ is always achievable for some $R' \in \psi(F')$ [18].

Next, to facilitate the conversion of general integer vectors into fundamentals for use in Problem 4.2, we provide the following definitions from [15]:

Definition 4.10 (GOF(p))**:** Given a nonnegative integer p, let

$$\text{GOF}(p) = \begin{cases} \text{the greatest odd factor of } p, & \text{if } p > 0, \\ 0, & \text{if } p = 0. \end{cases}$$

The function $\mathrm{GOF}(p)$ divides out all powers of 2 from the factorization of p. For example, $\mathrm{GOF}(10) = 5$, $\mathrm{GOF}(16) = 1$, and $\mathrm{GOF}(21) = 21$.

Definition 4.11 ($\mathrm{fund}(\mathbf{p})$)**:** Given a vector of integers $\mathbf{p} = (p_0, \ldots, p_{N-1}) \in \mathbb{Z}^N$ where $N \in \mathbb{N}$, let $\mathrm{fund}(\mathbf{p}) = \Big(\bigcup_{n=0}^{N-1} \{\mathrm{GOF}(|p_n|)\}\Big) \setminus \{0\}$.

The function $\mathrm{fund}(\mathbf{p})$ takes a vector of integers and *fundamentalizes* it to produce a set of corresponding fundamentals. For instance, $\mathrm{fund}\big((4, -18, 0, 26, -8)\big) = \{1, 13, 9\}$.

4.2.3 MRCM Problem with Adder-Count Cost Model

With Problems 4.1 and 4.2 defined, we now state the joint optimization problem, where the goal is to find a quantized constant vector and an optimal adder tree simultaneously.

We formulate the joint problem with *adder tree cost* as the objective function, because this choice enables us to easily compare our joint framework against the de facto disjointed framework based on Problems 4.1 and 4.2.

Problem 4.3 — MRCM problem with adder-count cost model: Given a vector length $N \in \mathbb{N}$, an ideal constant vector $\mathbf{h} \in \mathbb{R}^N$, an error bound $\varepsilon > 0$, a number of bits $m \in \{0\} \cup \mathbb{N}$, and a maximum depth $d \in \{0, 1, \ldots, \infty\}$, find a scale factor $s \in \mathbb{Z}$, a fixed-point constant vector $\hat{\mathbf{h}} \in \mathbb{R}^N$, and an adder tree $R' \in \mathcal{P}(R)$ that minimize $\mathrm{cost}(R')$ subject to $|\hat{h}_n| \cdot 2^s \in \{0, 1, \ldots, 2^m - 1\}$ for all $n \in \{0, \ldots, N-1\}$, $\|\,\mathbf{h} - \hat{\mathbf{h}}\,\|_1 \leq \varepsilon$, $R' \in \psi(\mathrm{fund}(\hat{\mathbf{h}} \cdot 2^s))$, and $\mathrm{depth}(R') \leq d$.

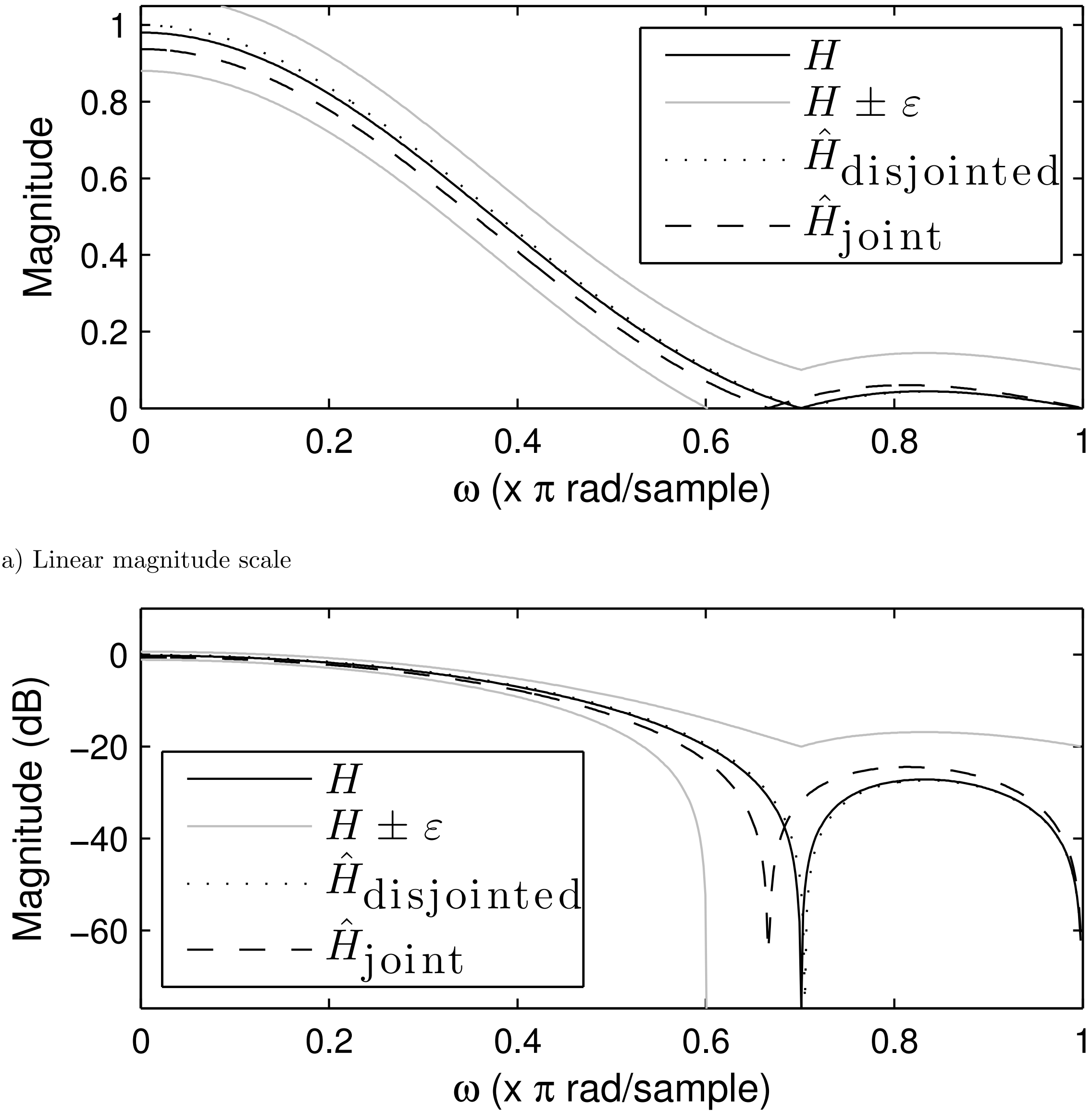
Problem 4.3 says that with error, bitwidth, and depth constraints, the constants $\mathbf{h}$ are quantized to $\hat{\mathbf{h}}$ and an adder tree R' is found for the fundamentals corresponding to $\hat{\mathbf{h}}$ such that the adder tree cost is minimized. Like in Problem 4.1, the 2^s term indicates where the fractional point is located and does not correspond to actual computation. Note that since Problem 4.3 incorporates Problem 4.1 as a subproblem, it inherits the property that a solution may not exist if the error bound is too restrictive for the given number of bits. Additionally, contrary to Problem 4.2, no lower bound can be established for the constraint on depth, and thus d can be chosen from all nonnegative integers. Because of this flexibility, it may be possible that no feasible adder tree can be found that satisfies the depth constraint.

4.3 Canonical Example

To show the benefit of the joint optimization problem, we provide the following example, where the joint optimization solution has a lower adder tree cost than the solution obtained by considering the problems independently. For illustrative purposes, this single trial, drawn from the randomized experiment to be described in Section 4.5, was chosen because it demonstrates the benefit of the joint problem framework, while still being simple enough to be solved ad-hoc.

Let the ideal filter coefficients be given as $\mathbf{h} = (0.1541, 0.3361, 0.3361, 0.1541)$, which correspond to a low-pass filter with a normalized cut-off frequency of approximately 0.4π. In general, four-tap filters such as these are common in wavelet applications. Also let $\varepsilon = 0.1$, so the quantized filter coefficients $\hat{\mathbf{h}} = (\hat{h}_0, \hat{h}_1, \hat{h}_2, \hat{h}_3)$ must be such that $\|\mathbf{h} - \hat{\mathbf{h}}\|_1 \leq 0.1$. As shown in (3.3), this restriction on $\hat{\mathbf{h}}$ guarantees that the quantized filter's frequency response deviates from the ideal one by no more than 0.1 for all frequencies. Both the ideal frequency response H and the ε-boundary lines are shown in Figure 4.5.

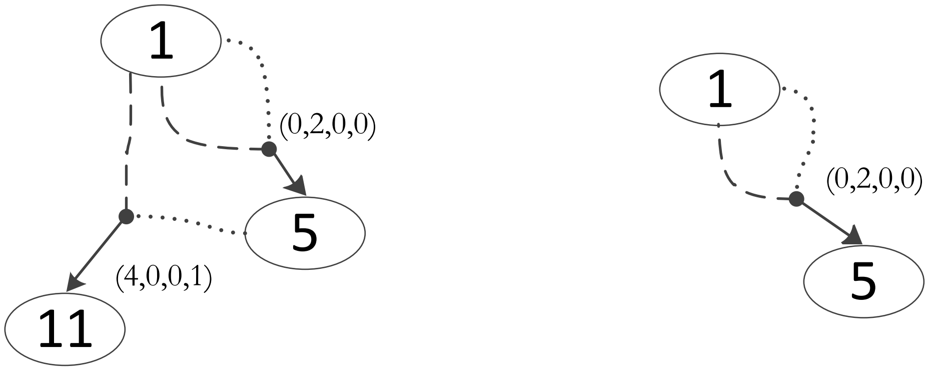
Without using a joint framework, the quantization and the MCM problems would have to be solved separately. For example, one could first find an optimal quantization according to Problem 4.1, and then find an optimal adder tree according to Problem 4.2. Using this approach, we first solve Problem 4.1 with the given $\mathbf{h}$ and ε. It can be shown that, for any number of bits, simple rounding of coefficients leads to the quantization with the least 1-norm error. In particular, this



(a) Linear magnitude scale

(b) Logarithmic magnitude scale

Figure 4.5: Frequency responses of the ideal filter H with the ε-boundary lines and the two quantized filters $\hat{H}_{\mathrm{disjointed}}$ and $\hat{H}_{\mathrm{joint}}$ for the example in Section 4.3.



(a) Optimal adder tree when $d = \infty$ and $F' = \text{fund}((5, 11, 11, 5) \cdot 2^5) = \{5, 11\}$. This disjointed solution requires 2 shift-add operations and has a depth of 2.

(b) Optimal adder tree when $d = \infty$ and $F' = \text{fund}((5, 10, 10, 5) \cdot 2^5) = \{5\}$. This joint solution requires 1 shift-add operation and has a depth of 1.

Figure 4.6: Adder trees for the example in Section 4.3.

property means that the $\hat{\mathbf{h}}$ in this example requires at least four bits in its quantization to satisfy the error constraint of Problem 4.1 because $\|\mathbf{h} - \hat{\mathbf{h}}\|_1 > \varepsilon$ when $\mathbf{h}$ is rounded to one, two, or three bits. Now, rounding $\mathbf{h}$ to four bits, we obtain $m^\star = 4$, $s^\star = 5$, and $\hat{\mathbf{h}}^\star = (5, 11, 11, 5)/2^5$, which is a solution to Problem 4.1 because $\|\mathbf{h} - \hat{\mathbf{h}}^\star\|_1 = 0.02 < \varepsilon$. The frequency response of this disjointed solution $\hat{\mathbf{h}}^\star$ is denoted as $\hat{H}_{\text{disjointed}}$ in Figure 4.5. Next, we take these results of the quantization problem and solve Problem 4.2. To this end, we fundamentalize $\hat{\mathbf{h}}^\star$ to obtain $F' = \text{fund}(\hat{\mathbf{h}}^\star \cdot 2^{s^\star}) = \{5, 11\}$ and, for simplicity, we leave the depth unconstrained with $d = \infty$. With this setup, it can be shown that a solution is the adder tree depicted in Figure 4.6(a), which has a cost of 2 shift-add operations.

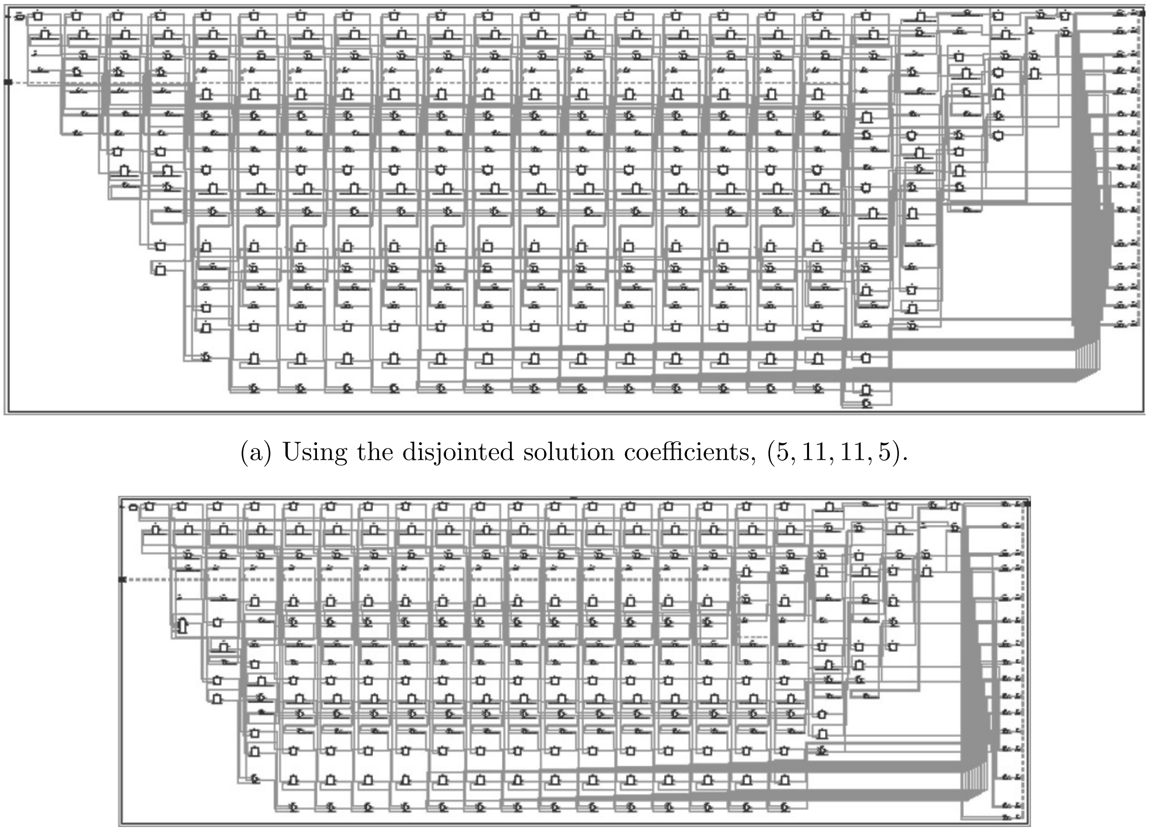
In comparison, one could solve the two problems simultaneously as defined in Problem 4.3. This joint problem requires that the number of bits m be specified, so in this example, we let $m = 4$ to ensure that the joint solution uses no more bits in its coefficients than the disjointed one, thus enabling a fair comparison of the two solutions. We again leave the depth unconstrained with $d = \infty$. Using the JOINT_SOLVE algorithm (to be developed in Section 4.4), a solution to Problem 4.3 is found to be $s^\star = 5$, $\hat{\mathbf{h}}^\star = (5, 10, 10, 5)/2^5$ and the adder tree as depicted in Figure 4.6(b) with a cost of 1 shift-add operation. These quantized coefficients satisfy the error constraint since $\|\mathbf{h} - \hat{\mathbf{h}}^\star\|_1 = 0.05 < \varepsilon$. The frequency response of this joint solution $\hat{\mathbf{h}}^\star$ is denoted as $\hat{H}_{\text{joint}}$ in Figure 4.5.

In the above example, solving the quantization and the MCM problems simultaneously leads to an adder tree that has 50% lower adder tree cost than that of the disjointed solution. This canonical example illustrates the characteristics that make the joint optimization problem beneficial: it allows one to assume extra coefficient error (within tolerances) in exchange for a reduction in adder tree cost. For additional comparison, the two filters obtained above are synthesized into Field-Programmable Gate Array (FPGA) schematics using the Xilinx ISE Design Suite [45] and the screenshots of these schematics are provided in Figure 4.7. Observe that the disjointed solution's filter requires more hardware real-estate than the joint solution's filter, indicating that *adder tree cost* does correlate with *circuit area* and is, therefore, a useful metric in this respect.

It follows from the above canonical example that there could be tremendous benefit to solving the quantization and MCM problems jointly. This provides the motivation to develop an algorithm that could realize those benefits by efficiently solving Problem 4.3, which is described next.

4.4 Proposed Algorithm

In this section, we develop an algorithm, JOINT_SOLVE, which solves the joint optimization problem, Problem 4.3. Section 4.4.1 outlines the basic strategy utilized in the algorithm. Building upon this strategy, Sections 4.4.2 and 4.4.3 detail methods to finitize the search space and find a feasible solution. Section 4.4.4 describes methods for further pruning to make the algorithm more efficient. Last, Section 4.4.5 points out the limitations of the algorithm in practice.



(a) Using the disjointed solution coefficients, $(5, 11, 11, 5)$.

(b) Using the joint solution coefficients, $(5, 10, 10, 5)$.

Figure 4.7: Screenshots of FPGA technology schematics generated using the filter architecture of Figure 2.1 and the coefficients obtained from the two solutions in Section 4.3.

4.4.1 Basic Strategy

Let $N \in \mathbb{N}$, $\mathbf{h} \in \mathbb{R}^N$, $\varepsilon > 0$, $m \in \{0\} \cup \mathbb{N}$, and $d \in \{0, 1, \ldots, \infty\}$ be given according to Problem 4.3. For convenience, let the optimization variables $s \in \mathbb{Z}$, $\hat{\mathbf{h}} \in \mathbb{R}^N$, and $R' \in \mathcal{P}(R)$ be written as $(s, \hat{\mathbf{h}}, R')$ and referred to as a *triplet.* A triplet $(s, \hat{\mathbf{h}}, R')$ is said to be *feasible* if it satisfies all of the constraints in Problem 4.3, which are:

$$|\hat{h}_n| \cdot 2^s \in \{0, 1, \ldots, 2^m - 1\} \quad \forall n \in \{0, \ldots, N-1\}, \tag{4.1a}$$

$$\|\mathbf{h} - \hat{\mathbf{h}}\|_1 \leq \varepsilon, \tag{4.1b}$$

$$R' \in \psi(\mathrm{fund}(\hat{\mathbf{h}} \cdot 2^s)), \quad \text{and} \tag{4.1c}$$

$$\mathrm{depth}(R') \leq d. \tag{4.1d}$$

Furthermore, a feasible triplet $(s, \hat{\mathbf{h}}, R')$ is said to be a *solution* of Problem 4.3 if it minimizes the cost function, $\mathrm{cost}(R')$. A basic algorithmic strategy to solve Problem 4.3 is to exhaustively search every triplet, checking each for feasibility, and then return one that minimizes cost. Such an algorithm is included as Algorithm 4.1. However, since the triplets live in the infinite set $\mathbb{Z} \times \mathbb{R}^N \times \mathcal{P}(R)$, it is impossible to search every triplet in finite time. To overcome this issue, we finitize the search space next.

4.4.2 Finitizing the $(s, \hat{\mathbf{h}})$ Search Space

Out of the four constraints (4.1a)–(4.1d), constraints (4.1a) and (4.1b) involve only the pair $(s, \hat{\mathbf{h}})$. In this subsection, we show that these two constraints alone force the feasible pairs to lie within a finite, albeit large, set, so that the search space for $(s, \hat{\mathbf{h}})$ can be made finite without loss of optimality.

Before deriving this finite search space, we address two special cases that are best handled separately. First, consider the case where $\varepsilon \geq \|\mathbf{h}\|_1$. In this situation,

Algorithm 4.1 INFINITE_SOLVE

```
procedure INFINITE_SOLVE $(N, \mathbf{h}, \varepsilon, m, d)$
    $(s^\star, \hat{\mathbf{h}}^\star, R'^\star) \leftarrow$ null
    $cost^\star \leftarrow \infty$
    for all $s \in \mathbb{Z}$ do
        for all $\hat{\mathbf{h}} \in \mathbb{R}^N$ do
            for all $R' \in \mathcal{P}(R)$ do
                if $\left\{ \begin{array}{l} |\hat{h}_n| \cdot 2^s \in \{0, 1, \ldots, 2^m - 1\} \\ \qquad \forall n \in \{0, \ldots, N-1\}, \\ \|\mathbf{h} - \hat{\mathbf{h}}\|_1 \le \varepsilon, \\ R' \in \psi(\mathrm{fund}(\hat{\mathbf{h}} \cdot 2^s)), \quad \text{and} \\ \mathrm{depth}(R') \le d \end{array} \right\}$ then
                    if $\mathrm{cost}(R') < cost^\star$ then
                        $(s^\star, \hat{\mathbf{h}}^\star, R'^\star) \leftarrow (s, \hat{\mathbf{h}}, R')$
                        $cost^\star \leftarrow \mathrm{cost}(R')$
                    end if
                end if
            end for
        end for
    end for
    return $(s^\star, \hat{\mathbf{h}}^\star, R'^\star)$
end procedure
```

$(s, \mathbf{0}, \varnothing)$ for any $s \in \mathbb{Z}$ is a (trivial) solution, where $\mathbf{0} = (0, \ldots, 0) \in \mathbb{R}^N$ and $\varnothing$ denotes the empty set. This is because if the error bound is too large, then even the trivial all-zero constant vector satisfies the error constraint (4.1b). Moreover, the rest of the constraints are satisfied and the cost is optimal, because $R' = \varnothing$ has zero cost (it requires no shift-add network at all). Next, consider the case where $m = 0$. In this situation, constraint (4.1a) necessitates that $\hat{\mathbf{h}} = \mathbf{0}$. It follows that $(s, \mathbf{0}, \varnothing)$ for any $s \in \mathbb{Z}$ is a solution if $\varepsilon \ge \|\mathbf{h}\|_1$, and no solution exists otherwise. In the sequel, we exclude these two special cases by assuming, from here onward, that

$$\varepsilon < \|\mathbf{h}\|_1 \quad \text{and} \tag{4.2}$$

$$m > 0. \tag{4.3}$$

Notice that for each fixed $s \in \mathbb{Z}$, there is only a *finite* number of $\hat{\mathbf{h}}$ vectors satisfying

(4.1a), and that this number may decrease if $\hat{\mathbf{h}}$ is also required to satisfy (4.1b). Thus, if we can show that for sufficiently small s and for sufficiently large s, the set of $\hat{\mathbf{h}}$ satisfying (4.1a) and (4.1b) is empty, then we would have established that the set of $(s, \hat{\mathbf{h}})$ pairs satisfying (4.1a) and (4.1b) is finite. The proposition below does just that and provides closed-form expressions for a sufficiently small and a sufficiently large s. In the proposition and its proof, $\|\mathbf{x}\|_1 = \sum_{i=0}^{N-1} |x_i|$ and $\|\mathbf{x}\|_\infty = \max_{i\in\{0,\ldots,N-1\}} |x_i|$ denote, respectively, the 1-norm and ∞-norm of $\mathbf{x} = (x_0, \ldots, x_{N-1}) \in \mathbb{R}^N$.

Proposition 4.1. *Let $N \in \mathbb{N}$, $\mathbf{h} \in \mathbb{R}^N$, $\varepsilon > 0$, and $m \in \{0\} \cup \mathbb{N}$ be given and suppose (4.2) and (4.3) hold. Let*

$$\underline{s} = \left\lceil -\log_2\bigl(\|\mathbf{h}\|_\infty + \varepsilon\bigr)\right\rceil \quad \textit{and} \tag{4.4}$$

$$\overline{s} = \left\lfloor \log_2 \frac{N(2^m - 1)}{\max\bigl\{\|\mathbf{h}\|_1 - \varepsilon, N\bigl(\|\mathbf{h}\|_\infty - \varepsilon\bigr)\bigr\}} \right\rfloor. \tag{4.5}$$

Then, for any $s \in \mathbb{Z}$, if $s < \underline{s}$ or $s > \overline{s}$, the set of $\hat{\mathbf{h}}$ satisfying (4.1a) and (4.1b) is empty.

Proof. First, suppose $s < \underline{s}$. Because s must be an integer and because of (4.4), $s \leq \underline{s} - 1 < -\log_2\bigl(\|\mathbf{h}\|_\infty + \varepsilon\bigr)$. Simplifying, we obtain

$$\frac{1}{2^s} > \|\mathbf{h}\|_\infty + \varepsilon. \tag{4.6}$$

Next, assume there exists an $\hat{\mathbf{h}} \in \mathbb{R}^N$ such that (4.1a) and (4.1b) hold. Due to (4.1b) and (4.2), $\hat{\mathbf{h}} \neq \mathbf{0}$. This fact can be combined with (4.1a) to show that

$$\|\hat{\mathbf{h}}\|_\infty = \max_{i\in\{0,\ldots,N-1\}} |\hat{h}_i| \geq \frac{1}{2^s}. \tag{4.7}$$

Combining (4.6) and (4.7), we obtain

$$\|\hat{\mathbf{h}}\|_\infty - \|\mathbf{h}\|_\infty > \varepsilon. \tag{4.8}$$

The reverse triangle inequality says that

$$\|\hat{\mathbf{h}} - \mathbf{h}\|_\infty \geq \|\hat{\mathbf{h}}\|_\infty - \|\mathbf{h}\|_\infty. \tag{4.9}$$

Furthermore, it is known that the 1-norm and ∞-norm satisfy

$$\|\hat{\mathbf{h}} - \mathbf{h}\|_1 \geq \|\hat{\mathbf{h}} - \mathbf{h}\|_\infty. \tag{4.10}$$

Combining (4.8)–(4.10), we obtain $\|\hat{\mathbf{h}} - \mathbf{h}\|_1 > \varepsilon$, contradicting (4.1b). Thus, if $s < \underline{s}$, the set of $\hat{\mathbf{h}}$ satisfying (4.1a) and (4.1b) is empty.

Second, suppose $s > \overline{s}$ where $\overline{s}$ is given in (4.5), which is well-defined due to (4.2) and (4.3). Since s must be an integer,

$$s \geq \overline{s} + 1 > \log_2 \frac{N(2^m - 1)}{\max\bigl\{\|\mathbf{h}\|_1 - \varepsilon, N\bigl(\|\mathbf{h}\|_\infty - \varepsilon\bigr)\bigr\}}. \tag{4.11}$$

Now, suppose there exists an $\hat{\mathbf{h}} \in \mathbb{R}^N$ such that (4.1a) and (4.1b) hold. Then, (4.1a) implies that

$$\|\hat{\mathbf{h}}\|_\infty = \max_{i \in \{0,\ldots,N-1\}} |\hat{h}_i| \leq \frac{2^m - 1}{2^s}. \tag{4.12}$$

Consider the following two cases:

Case 1: $\|\mathbf{h}\|_1 - \varepsilon \geq N(\|\mathbf{h}\|_\infty - \varepsilon)$. In this case, (4.11) becomes $s > \log_2 \frac{N(2^m-1)}{\|\mathbf{h}\|_1 - \varepsilon}$. It follows that

$$\frac{N(2^m - 1)}{2^s} < \|\mathbf{h}\|_1 - \varepsilon. \tag{4.13}$$

Furthermore, the 1-norm and ∞-norm are known to satisfy

$$\|\hat{\mathbf{h}}\|_1 \leq N\|\hat{\mathbf{h}}\|_\infty. \tag{4.14}$$

Combining (4.12)–(4.14), we obtain $\|\hat{\mathbf{h}}\|_1 \leq \frac{N(2^m-1)}{2^s} < \|\mathbf{h}\|_1 - \mathcal{E}$, and thus $\mathcal{E} < \|\mathbf{h}\|_1 - \|\hat{\mathbf{h}}\|_1$. Using the reverse triangle inequality, we have $\mathcal{E} < \|\mathbf{h}\|_1 - \|\hat{\mathbf{h}}\|_1 \leq \|\mathbf{h} - \hat{\mathbf{h}}\|_1$, contradicting (4.1b).

Case 2: $\|\mathbf{h}\|_1 - \mathcal{E} < N(\|\mathbf{h}\|_\infty - \mathcal{E})$. In this case, (4.11) becomes $s > \log_2 \frac{N(2^m-1)}{N\left(\|\mathbf{h}\|_\infty - \mathcal{E}\right)}$. It follows that

$$\mathcal{E} + \frac{2^m-1}{2^s} < \|\mathbf{h}\|_\infty. \tag{4.15}$$

Combining (4.12) and (4.15), we obtain $\mathcal{E} + \frac{2^m-1}{2^s} < \|\mathbf{h}\|_\infty \leq \frac{2^m-1}{2^s}$, implying that $\mathcal{E} < 0$, contradicting the fact that $\mathcal{E} > 0$.

Hence, if $s > \overline{s}$, the set of $\hat{\mathbf{h}}$ satisfying (4.1a) and (4.1b) is empty. □

Proposition 4.1 has two implications:

(i) If $\underline{s} > \overline{s}$, then there are no $(s, \hat{\mathbf{h}})$ pairs satisfying (4.1a) and (4.1b), so that Problem 4.3 is infeasible; and

(ii) if $\underline{s} \leq \overline{s}$, then any $(s, \hat{\mathbf{h}})$ pair satisfying (4.1a) and (4.1b) must also satisfy

$$s \in \{\underline{s}, \ldots, \overline{s}\}. \tag{4.16}$$

Since case (i) is of limited interest, in the sequel we focus on case (ii). Note that for this case, given a particular s from (4.16), the set of $\hat{\mathbf{h}}$ satisfying (4.1a) and (4.1b) can be expressed as

$$\hat{\mathbf{h}} \in \left\{\tfrac{-(2^m-1)}{2^s}, \ldots, \tfrac{2^m-1}{2^s}\right\}^N \bigcap \left\{\mathbf{x} \in \mathbb{R}^N : \|\mathbf{h} - \mathbf{x}\|_1 \leq \mathcal{E}\right\}. \tag{4.17}$$

Together, the s from (4.16) paired with the $\hat{\mathbf{h}}$ from (4.17) form the finite $(s, \hat{\mathbf{h}})$ search space.

Although (4.16) and (4.17) completely characterize the finite $(s, \hat{\mathbf{h}})$ search space, they do not prescribe an efficient means of enumerating the space. To see this, note that a simple way to loop through the space is

```
for $s = \underline{s}$ to $\overline{s}$ do
    for $\hat{h}_0 = \frac{-(2^m - 1)}{2^s}$ to $\frac{2^m - 1}{2^s}$ do
        ⋱
            for $\hat{h}_{N-1} = \frac{-(2^m - 1)}{2^s}$ to $\frac{2^m - 1}{2^s}$ do                (4.18)
                if $\|\mathbf{h} - \hat{\mathbf{h}}\|_1 \leq \varepsilon$ then
                    ⋮
```

where the constraints (4.1a) and (4.1b) are handled independently. While possible, this looping scheme may be very inefficient because it does not leverage the fact that (4.1a) and (4.1b) can be handled simultaneously at each level of the nested loops, i.e.,

```
for $s = \underline{s}$ to $\overline{s}$ do
    for $\hat{h}_0 = \frac{\alpha_0}{2^s}$ to $\frac{\beta_0}{2^s}$ do
        ⋱                                                                                   (4.19)
            for $\hat{h}_{N-1} = \frac{\alpha_{N-1}}{2^s}$ to $\frac{\beta_{N-1}}{2^s}$ do
                ⋮
```

where for all $n \in \{0, \ldots, N-1\}$, α_n and β_n are integers defining the limits of the loops. If all of the α_n's and β_n's are defined properly (in a manner dependent on $\hat{h}_0, \ldots, \hat{h}_{n-1}$), then (4.1a) and (4.1b) will be implicitly satisfied at each level of the nested loops, thus removing the inefficiencies associated with the simple loop structure of Line 6. The α_n's and β_n's that achieve this goal are now derived.

Suppose s has been assigned in the outermost loop of Line 5. Observe from (3.1) that the total 1-norm error $\|\mathbf{h} - \hat{\mathbf{h}}\|_1$ is obtained by summing all of the individual constant errors. Also observe that for each $n \in \{0, \ldots, N-1\}$, the minimum possible error of $\hat{h}_n$ satisfying (4.1a), denoted by ξ_n, is

$$\xi_n = \min_{x \in \{-(2^m-1), \ldots, 2^m-1\}} \left| h_n - \frac{x}{2^s} \right|. \tag{4.20}$$

Note that the ξ_n's need only be calculated once for each s.

Using these ξ_n's, we may recursively derive the *individual error tolerances* of $\hat{h}_0, \ldots, \hat{h}_{N-1}$, denoted as $\mathcal{E}_0, \ldots, \mathcal{E}_{N-1}$. The quantity $\mathcal{E}_0$ is given by the total error tolerance $\mathcal{E}$ minus the minimum possible error incurred by $\hat{h}_1, \ldots, \hat{h}_{N-1}$, i.e.,

$$\mathcal{E}_0 = \mathcal{E} - \sum_{i=1}^{N-1} \xi_i. \tag{4.21}$$

To compute $\mathcal{E}_1, \ldots, \mathcal{E}_{N-1}$, suppose we are at the loop level where $\hat{h}_n$ is to be assigned. For this $\hat{h}_n$, the quantity $\mathcal{E}_n$ is given by the total error tolerance $\mathcal{E}$ minus the error incurred by the *assigned* constants $\hat{h}_0, \ldots, \hat{h}_{n-1}$ and the minimum possible error incurred by the *unassigned* constants $\hat{h}_{n+1}, \ldots, \hat{h}_{N-1}$. For example, $\mathcal{E}_1 = \mathcal{E} - |h_0 - \hat{h}_0| - \sum_{i=2}^{N-1} \xi_i$ which, due to (4.21), can be rewritten as $\mathcal{E}_1 = \mathcal{E}_0 - |h_0 - \hat{h}_0| + \xi_1$. Likewise, $\mathcal{E}_2 = \mathcal{E} - \sum_{i=0}^{1} |h_i - \hat{h}_i| - \sum_{i=3}^{N-1} \xi_i = \mathcal{E}_1 - |h_1 - \hat{h}_1| + \xi_2$, and so on. Therefore, for each $n \in \{1, \ldots, N-1\}$, the quantity $\mathcal{E}_n$ may be computed recursively as

$$\mathcal{E}_{n+1} = \mathcal{E}_n - |h_n - \hat{h}_n| + \xi_{n+1}. \tag{4.22}$$

With (4.21) and (4.22), we can establish the exact α_n and β_n limits that guarantee the satisfaction of (4.17). Since each $\hat{h}_n$ must satisfy (4.1a) and lie in the interval $[h_n - \mathcal{E}_n, h_n + \mathcal{E}_n]$, we arrive at the limits

$$\alpha_n = \lceil \max\{-(2^m - 1), (h_n - \mathcal{E}_n)2^s\} \rceil \quad \text{and} \tag{4.23}$$

$$\beta_n = \lfloor \min\{(h_n + \varepsilon_n)2^s, 2^m - 1\} \rfloor. \tag{4.24}$$

Note that if $\alpha_n > \beta_n$ for some $n \in \{0, \ldots, N-1\}$, then there is no $\hat{h}_n$ and, thus, no $\hat{\mathbf{h}}$ satisfying (4.1a) and (4.1b) with the given s, causing the particular loop level to terminate.

In summary, executing Line 5 with the α_n's and β_n's given by (4.23) and (4.24) allows us to enumerate the established $(s, \hat{\mathbf{h}})$ search space efficiently without ever visiting any pairs outside the space.

4.4.3 Applying $\mathbf{H_{cub}}$ to the $\boldsymbol{R'}$ Search Space

In Section 4.4.2, we focused on constraints (4.1a) and (4.1b) and derived the set of $(s, \hat{\mathbf{h}})$ pairs satisfying them. In this subsection, we assume that a pair $(s, \hat{\mathbf{h}})$ satisfying (4.1a) and (4.1b) is given, and focus on the remaining optimization variable $R' \in \mathcal{P}(R)$, which must satisfy constraints (4.1c) and (4.1d). Unlike Section 4.4.2, however, the R' search space cannot be made finite by these two constraints in general, because if there is one adder tree R' satisfying (4.1c) and (4.1d) with nonzero depth, then there are infinitely many other adder trees that can be built from R' that also satisfy (4.1c) and (4.1d). Fortunately, we may alleviate this issue in the following manner. Note that with s and $\hat{\mathbf{h}}$ fixed, Problem 4.3 reduces to the MCM problem, i.e., Problem 4.2 with $F' = \text{fund}(\hat{\mathbf{h}} \cdot 2^s)$, which can be solved using currently available algorithms (e.g., [10,13–17,19–22]). Although multiple choices exist, here we adopt the $\mathrm{H_{cub}}$ algorithm of [13] since it is well-established as a viable MCM solver—see [17,21]—and since the full source code is publicly available [46].

In its simplest form, $\mathrm{H_{cub}}$ takes a set of input fundamentals and a maximum depth and returns an approximate solution to Problem 4.2. The returned adder tree is an *approximate* solution because $\mathrm{H_{cub}}$ is a heuristic. For the purpose of solving the MCM part of Problem 4.3, we invoke $\mathrm{H_{cub}}$ as follows:

$$R' \leftarrow \mathrm{H_{cub}}(\mathrm{fund}(\hat{\mathbf{h}} \cdot 2^s), d).$$

With $\mathrm{H_{cub}}$, the returned adder tree R' is guaranteed to satisfy (4.1c), but it does not always satisfy the depth constraint (4.1d). Indeed, $\mathrm{H_{cub}}$ is implemented so as to find an adder tree with depth not exceeding d, but when it cannot, $\mathrm{H_{cub}}$ still returns an adder tree with nearly minimal depth (see [46] for more details). As a result, the satisfaction of (4.1d) must be verified for the returned adder tree. This behavior also means that calling $\mathrm{H_{cub}}$ with $d = 0$ leads to a near-minimal-depth solution.

Combining the $(s, \hat{\mathbf{h}})$ search space of Section 4.4.2 with the use of $\mathrm{H_{cub}}$ in this subsection, we have established a finite $(s, \hat{\mathbf{h}}, R')$ search space along with an efficient looping scheme, given in Line 5. Together, these can be used to transform the infinite-length algorithm of Algorithm 4.1 with the finite one in Algorithm 4.2.

Algorithm 4.2 FINITE_SOLVE

```
procedure FINITE_SOLVE $(N, \mathbf{h}, \mathcal{E}, m, d)$
    if $\mathcal{E} \geq \|\mathbf{h}\|_1$ then                                        ▷ first special case
        return $(0, \mathbf{0}, \varnothing)$
    end if
    if $m = 0$ then                                                        ▷ second special case, no solution
        return null
    end if
    $(s^\star, \hat{\mathbf{h}}^\star, R'^\star) \leftarrow$ null
    $cost^\star \leftarrow \infty$
    for $s = \lceil -\log_2(\|\mathbf{h}\|_\infty + \mathcal{E}) \rceil$ to $\left\lfloor \log_2 \frac{N(2^m-1)}{\max\{\|\mathbf{h}\|_1 - \mathcal{E}, N(\|\mathbf{h}\|_\infty - \mathcal{E})\}} \right\rfloor$ do      ▷ from (4.4) and (4.5)
        for $n = 0$ to $N - 1$ do
            $\xi_n \leftarrow \min_{x \in \{-(2^m-1),\ldots,2^m-1\}} |h_n - \frac{x}{2^s}|$      ▷ from (4.20)
        end for
        $\mathcal{E}_0 \leftarrow \mathcal{E} - \sum_{i=1}^{N-1} \xi_i$                ▷ from (4.21)
        for $\hat{h}_0 = \frac{\lceil \max\{-(2^m-1), (h_0 - \mathcal{E}_0)2^s\} \rceil}{2^s}$ to $\frac{\lfloor \min\{(h_0 + \mathcal{E}_0)2^s, 2^m - 1\} \rfloor}{2^s}$ do      ▷ from (4.23) and (4.24)
nested loops:      ⋱
$n \in \{1,\ldots,N-1\}$   $\mathcal{E}_n \leftarrow \mathcal{E}_{n-1} - |h_{n-1} - \hat{h}_{n-1}| + \xi_n$      ▷ from (4.22)
            for $\hat{h}_n = \frac{\lceil \max\{-(2^m-1), (h_n - \mathcal{E}_n)2^s\} \rceil}{2^s}$ to $\frac{\lfloor \min\{(h_n + \mathcal{E}_n)2^s, 2^m - 1\} \rfloor}{2^s}$ do      ▷ from (4.23) and (4.24)
            ⋮
                $R' \leftarrow \mathrm{H_{cub}}(\mathrm{fund}(\hat{\mathbf{h}} \cdot 2^s), d)$
                if $\mathrm{depth}(R') \leq d$ then                ▷ must check $\mathrm{H_{cub}}$ output's depth
                    if $\mathrm{cost}(R') < cost^\star$ then
                        $(s^\star, \hat{\mathbf{h}}^\star, R'^\star) \leftarrow (s, \hat{\mathbf{h}}, R')$
                        $cost^\star \leftarrow \mathrm{cost}(R')$
                    end if
                end if
            end for
        end for
    end for
    return $(s^\star, \hat{\mathbf{h}}^\star, R'^\star)$
end procedure
```

4.4.4 Further Pruning the $(s, \hat{\mathbf{h}}, R')$ Search Space

As it turns out, further pruning of the $(s, \hat{\mathbf{h}}, R')$ search space is possible. In this subsection, we discuss five pruning strategies that further reduce this search space for efficient implementation of our JOINT_SOLVE algorithm. These strategies are developed to leverage cases where it is deduced that a triplet will be either infeasible, non-optimal, or redundant. To facilitate their development, suppose that $(s^\star, \hat{\mathbf{h}}^\star, R'^\star)$ is a solution to Problem 4.3. With this in mind, the pruning strategies can be described as follows:

(a) *Removing smaller approximations*: Observe that if a solution exists where all the constants are represented using strictly less than m bits, then another solution exists that uses all m bits in its representation, so we can disregard the former. The following proposition formalizes this claim:

Proposition 4.2. *If* $\|\hat{\mathbf{h}}^\star\|_\infty \cdot 2^{s^\star} < 2^{m-1}$*, then there exists another solution* $(s^{\star\star}, \hat{\mathbf{h}}^{\star\star}, R'^{\star\star})$ *to Problem 4.3, where* $\|\hat{\mathbf{h}}^{\star\star}\|_\infty \cdot 2^{s^{\star\star}} \geq 2^{m-1}$*.*

Proof. Let $s^{\star\star} = m - 1 - \lfloor \log_2 \|\hat{\mathbf{h}}^\star\|_\infty \rfloor$, $\hat{\mathbf{h}}^{\star\star} = \hat{\mathbf{h}}^\star$, and $R'^{\star\star} = R'^\star$. Then,

$$\begin{aligned}
\|\hat{\mathbf{h}}^{\star\star}\|_\infty \cdot 2^{s^{\star\star}} &= \|\hat{\mathbf{h}}^{\star\star}\|_\infty \cdot 2^{m-1-\lfloor \log_2 \|\hat{\mathbf{h}}^{\star\star}\|_\infty \rfloor} \\
&= 2^{m-1-\lfloor \log_2 \|\hat{\mathbf{h}}^{\star\star}\|_\infty \rfloor + \log_2 \|\hat{\mathbf{h}}^{\star\star}\|_\infty} && (4.25) \\
&\geq 2^{m-1}. && (4.26)
\end{aligned}$$

Next, we show that $(s^{\star\star}, \hat{\mathbf{h}}^{\star\star}, R'^{\star\star})$, derived from the solution $(s^\star, \hat{\mathbf{h}}^\star, R'^\star)$, is also a solution. First, since $\hat{\mathbf{h}}^{\star\star} = \hat{\mathbf{h}}^\star$ and $R'^{\star\star} = R'^\star$, (4.1b) and (4.1d) hold. Second, (4.26) shows that $s^{\star\star} > s^\star$, since $\|\hat{\mathbf{h}}^{\star\star}\|_\infty \cdot 2^{s^{\star\star}} \geq 2^{m-1} > \|\hat{\mathbf{h}}^\star\|_\infty \cdot 2^{s^\star}$. Therefore, the quantity to be fundamentalized in (4.1c), $\hat{\mathbf{h}}^{\star\star} \cdot 2^{s^{\star\star}}$, is simply $\hat{\mathbf{h}}^\star \cdot 2^{s^\star}$ multiplied by a power of 2, so the corresponding fundamentals remain unchanged and (4.1c) is satisfied. Last, (4.25) implies that $\|\hat{\mathbf{h}}^{\star\star}\|_\infty \cdot 2^{s^{\star\star}} < 2^m$, suggesting that (4.1a) holds. Since (4.1a)–(4.1d) hold and $\mathrm{cost}(R'^{\star\star}) = \mathrm{cost}(R'^{\star})$, $(s^{\star\star}, \hat{\mathbf{h}}^{\star\star}, R'^{\star\star})$ is another solution to Problem 4.3. □

Since we are interested in *any* solution to Problem 4.3, Proposition 4.2 implies that we need not consider any pair $(s, \hat{\mathbf{h}})$ for which $\|\hat{\mathbf{h}}\|_\infty \cdot 2^s < 2^{m-1}$, leading to a pruning of the $(s, \hat{\mathbf{h}})$ search space. To implement the pruning, we introduce the additional constraint $\|\hat{\mathbf{h}}\|_\infty \cdot 2^s \geq 2^{m-1}$, which translates to $s \geq \log_2 \frac{2^{m-1}}{\|\hat{\mathbf{h}}\|_\infty}$. Due to the triangle inequality, (4.10), and (4.1b), we have $\|\hat{\mathbf{h}}\|_\infty \leq \|\mathbf{h} - \hat{\mathbf{h}}\|_\infty + \|\mathbf{h}\|_\infty \leq \|\mathbf{h}\|_\infty + \varepsilon$. Consequently, $s \geq \log_2 \frac{2^{m-1}}{\|\mathbf{h}\|_\infty + \varepsilon}$, so we can prune the s search space in (4.16) by replacing (4.4) with the tighter limit

$$\underline{s} = m - 1 + \left\lceil -\log_2\left(\|\mathbf{h}\|_\infty + \varepsilon\right)\right\rceil.$$

(b) *Matching constant signs*: Each approximating constant $\hat{h}_n$ can be restricted to having the same sign as the corresponding ideal constant h_n without loss of optimality. The following proposition formalizes the claim, where $\mathrm{sgn}(x) = 1$ if $x > 0$, -1 if $x < 0$, and 0 if $x = 0$:

Proposition 4.3. *If there exists an* $n \in \{0, \ldots, N-1\}$ *such that* $h_n \neq 0$, $\hat{h}_n^\star \neq 0$, *and* $\mathrm{sgn}(\hat{h}_n^\star) \neq \mathrm{sgn}(h_n)$, *then there exists another solution* $(s^{\star\star}, \hat{\mathbf{h}}^{\star\star}, R'^{\star\star})$ *to Problem 4.3, where* $\mathrm{sgn}(\hat{h}_n^{\star\star}) = \mathrm{sgn}(h_n)$.

Proof. Let $\hat{\mathbf{h}}^{\star\star} = \hat{\mathbf{h}}^\star$, but with $\hat{h}_n^{\star\star} = -\hat{h}_n^\star$. Then, $\mathrm{sgn}(\hat{h}_n^{\star\star}) = \mathrm{sgn}(h_n)$. Also, let $s^{\star\star} = s^\star$ and $R'^{\star\star} = R'^\star$. We now show that $(s^{\star\star}, \hat{\mathbf{h}}^{\star\star}, R'^{\star\star})$, derived from the solution $(s^\star, \hat{\mathbf{h}}^\star, R'^\star)$, is also a solution. First, (4.1d) holds and $\mathrm{cost}(R'^{\star\star}) = \mathrm{cost}(R'^\star)$, because $R'^{\star\star} = R'^\star$. Next, (4.1a) and (4.1c) hold because they only involve the magnitudes of the constants, all of which remain unchanged by the definition of $\hat{\mathbf{h}}^{\star\star}$. Last, $\|\mathbf{h} - \hat{\mathbf{h}}^{\star\star}\|_1 < \|\mathbf{h} - \hat{\mathbf{h}}^\star\|_1$ since the sign of $\hat{h}_n^{\star\star}$ was made to match that of h_n, so (4.1b) is also satisfied. It follows that $(s^{\star\star}, \hat{\mathbf{h}}^{\star\star}, R'^{\star\star})$ is another solution to Problem 4.3. $\square$

Proposition 4.3 says that all the signs of $\hat{\mathbf{h}}$ can be forced to match those of $\mathbf{h}$ without loss of optimality. Therefore, (4.1a) can be replaced by the more restrictive constraint

$$\hat{h}_n \cdot 2^s \in \{0 \cdot \mathrm{sgn}(h_n), 1 \cdot \mathrm{sgn}(h_n), \ldots, (2^m - 1) \cdot \mathrm{sgn}(h_n)\} \quad \forall n \in \{0, \ldots, N-1\}, \tag{4.27}$$

thereby pruning the $(s, \hat{\mathbf{h}})$ search space. This new constraint (4.27) can be taken into account by replacing the looping scheme Line 5 with

```
for $s = \underline{s}$ to $\overline{s}$ do
    for $\hat{h}_0 = \frac{\alpha_0 \cdot \mathrm{sgn}(h_0)}{2^s}$ to $\frac{\beta_0 \cdot \mathrm{sgn}(h_0)}{2^s}$ do
        $\ddots$
            for $\hat{h}_{N-1} = \frac{\alpha_{N-1} \cdot \mathrm{sgn}(h_{N-1})}{2^s}$ to $\frac{\beta_{N-1} \cdot \mathrm{sgn}(h_{N-1})}{2^s}$ do        (4.28)
                $\vdots$
```

and replacing the calculations of ξ_n, α_n, and β_n in (4.20), (4.23), and (4.24) with

$$\begin{aligned}
\xi_n &= \min_{x \in \{0, \ldots, 2^m - 1\}} \left| |h_n| - \frac{x}{2^s} \right|, \\
\alpha_n &= \lceil \max\{0, (|h_n| - \mathcal{E}_n) 2^s\} \rceil, \quad \text{and} \\
\beta_n &= \lfloor \min\{(|h_n| + \mathcal{E}_n) 2^s, 2^m - 1\} \rfloor,
\end{aligned}$$

respectively. Note that the recursive calculations of $\mathcal{E}_n$ in (4.21) and (4.22) are unaffected. With the above changes in place, the algorithm treats all h_n's and $\hat{h}_n$'s as nonnegative for the individual error tolerance calculations and then imposes the proper signs in Line 4.

(c) *Mirroring duplicate constants*: When an ideal constant vector $\mathbf{h}$ has duplicate components (in magnitude), the approximation $\hat{\mathbf{h}}$ should have the same components duplicated. The following proposition formalizes the claim:

Proposition 4.4. *If there exist* $n_0, n_1 \in \{0, \ldots, N-1\}$ *such that* $|h_{n_0}| = |h_{n_1}|$ *but* $|\hat{h}^\star_{n_0}| \neq |\hat{h}^\star_{n_1}|$*, then there exists another solution* $(s^{\star\star}, \hat{\mathbf{h}}^{\star\star}, R'^{\star\star})$ *where* $|\hat{h}^{\star\star}_{n_0}| = |\hat{h}^{\star\star}_{n_1}|$.

Proof. Without loss of generality, suppose n_0 and n_1 are such that $|h_{n_0} - \hat{h}^\star_{n_0}| \leq |h_{n_1} - \hat{h}^\star_{n_1}|$. Let $\hat{\mathbf{h}}^{\star\star} = \hat{\mathbf{h}}^\star$, but with $\hat{h}^{\star\star}_{n_1} = \hat{h}^\star_{n_0}$ if $h_{n_0} = h_{n_1}$, and $\hat{h}^{\star\star}_{n_1} = -\hat{h}^\star_{n_0}$ otherwise. Then, $|\hat{h}^{\star\star}_{n_0}| = |\hat{h}^{\star\star}_{n_1}|$. Also, let $s^{\star\star} = s^\star$ and $R'^{\star\star} = R'^\star$. We now show that $(s^{\star\star}, \hat{\mathbf{h}}^{\star\star}, R'^{\star\star})$ is a solution as well. Clearly, (4.1d) holds and $\text{cost}(R'^{\star\star}) = \text{cost}(R'^\star)$. To establish (4.1b), note that if $h_{n_0} = h_{n_1}$, we have $|\hat{h}^{\star\star}_{n_1} - h_{n_1}| = |\hat{h}^\star_{n_0} - h_{n_0}| \leq |h_{n_1} - \hat{h}^\star_{n_1}|$. If $h_{n_0} \neq h_{n_1}$ so that $h_{n_0} = -h_{n_1}$, we have the exact same inequality. Since $\hat{\mathbf{h}}^{\star\star} = \hat{\mathbf{h}}^\star$ except for the n_1-th component, (4.1b) is satisfied. Last, (4.1a) and (4.1c) still hold because no additional constants (magnitudes) are introduced by the definition of $\hat{\mathbf{h}}^{\star\star}$. Consequently, $(s^{\star\star}, \hat{\mathbf{h}}^{\star\star}, R'^{\star\star})$ is another solution to Problem 4.3. □

Proposition 4.4 says that we need not consider any $\hat{\mathbf{h}}$ where the duplicate components from $\mathbf{h}$ are not matched properly. Accordingly, we prune the $\hat{\mathbf{h}}$ search space by bypassing the search loop for a particular $\hat{h}_n$ when it is forced to match and replacing the loop with a simple assignment instead (see lines 2–13 of Algorithm 4.3).

(d) *Utilizing lower bounds from [18]*: It is conceivable that, before reaching the innermost loop of Line 5, the number of unique fundamentals corresponding to the assigned $\hat{h}_n$'s might grow so large as to eliminate the possibility of having a feasible or cost-optimal adder tree, thereby eliminating the need to execute the inner nested loops. Indeed, in [18] Gustafsson establishes lower bounds on the cost and depth of adder trees associated with a given set of fundamentals. The results can be summarized as follows:

Given a fundamental $f \in F$, let $S(f)$ denote the number of nonzero digits in the canonical-signed-digit representation of f. Also, given two positive integers x_1, x_2 with $x_1 \leq x_2$, let

$$E(x_1, x_2) = \begin{cases} 1, & x_1 = x_2, \\ \left\lceil \log_2 \frac{x_2}{x_1} \right\rceil, & x_1 < x_2. \end{cases}$$

Using these two functions along with (6) and (18) of [18], such lower bounds can be computed. Let $F' = \{f_1, f_2, \ldots, f_M\} \subset F$ be a set of fundamentals that are ordered by increasing $S(f_i)$. Then, the lower bounds from [18] on the cost and depth of any adder tree that produces F' are given by, respectively,

$$\text{cost}_{\text{LB}}(F') = \lceil \log_2 S(f_1) \rceil + \sum_{i=1}^{M-1} E(S(f_i), S(f_{i+1}))$$

and

$$\text{depth}_{\text{LB}}(F') = \max_i \lceil \log_2 S(f_i) \rceil$$

These lower bounds lead to the following pruning opportunity: Suppose $cost^\star$ denotes the minimum adder tree cost of all feasible $(s, \hat{\mathbf{h}}, R')$ triplets encountered thus far. Also, suppose we are at the loop level where $\hat{h}_n$ has just been assigned for some $n \in \{0, \ldots, N-1\}$. Then, if the assigned constants $\hat{h}_0, \ldots, \hat{h}_n$ having fundamentals $F' = \text{fund}((\hat{h}_0, \ldots, \hat{h}_n) \cdot 2^s)$ are such that $\text{cost}_{\text{LB}}(F') \geq cost^\star$ or $\text{depth}_{\text{LB}}(F') > d$, we can prune the search space by terminating the current loop level.

(e) *Pruning H_{cub}*: The algorithm H_{cub}, obtained from [46] and described in Section 4.4.3, always returns an adder tree R' satisfying (4.1c), but it does not guarantee that (4.1d) holds. Furthermore, it runs to completion, even when there is no way to beat the minimum adder tree cost found so far. Therefore, we have modified $\mathrm{H_{cub}}$ to take an additional parameter and to operate more efficiently. In particular, given a set of input fundamentals, a maximum depth, and a maximum cost, this modified algorithm returns an adder tree in the same manner as $\mathrm{H_{cub}}$, but is pruned so as to terminate immediately if either the maximum depth or cost is surpassed. We call this modified algorithm Hcub_Pruned. Supposing $cost^\star$ is the minimum adder tree cost of all feasible $(s, \hat{\mathbf{h}}, R')$ triplets encountered thus far, we invoke the algorithm for a particular $(s, \hat{\mathbf{h}})$ pair as:

$$(R', valid) \leftarrow \textsc{Hcub\_Pruned}(\mathrm{fund}(\hat{\mathbf{h}} \cdot 2^s), d, cost^\star),$$

where $valid = true$ if the returned R' satisfies (4.1d) and $\mathrm{cost}(R') < cost^\star$. Otherwise, $valid =$ false, implying that Hcub_Pruned is terminated prematurely due to exceeding the depth or cost limits.

Altogether, we incorporate the five pruning strategies (a)–(e) described above into the search loop enumerating the finite $(s, \hat{\mathbf{h}}, R')$ search space derived in Sections 4.4.2 and 4.4.3. In doing so, we obtain the proposed Joint_Solve algorithm of Algorithm 4.3 (along with the sub-procedures of Algorithm 4.4 that it calls) for solving Problem 4.3. Note that Joint_Solve accepts all of the parameters of Problem 4.3 and returns a near-optimal triplet $(s, \hat{\mathbf{h}}, R')$, or `null` if no feasible triplet is found. Also, although Algorithm 4.3 contains pseudocode to illustrate the structural flow of Joint_Solve, an actual implementation should include proper data structures for optimal array access and should reuse computational results when possible.

Algorithm 4.3 JOINT_SOLVE

```
procedure $(s^\star, \hat{\mathbf{h}}^\star, R'^\star) = $ JOINT_SOLVE $(N, \mathbf{h}, \varepsilon, m, d)$
    if $\varepsilon \geq \|\mathbf{h}\|_1$ then                                ▷ first special case
        return $(0, \mathbf{0}, \varnothing)$
    end if
    if $m = 0$ then                                       ▷ second special case, no solution
        return null
    end if
    $(s^\star, \hat{\mathbf{h}}^\star, R'^\star) \leftarrow$ null
    $cost^\star \leftarrow \infty$
    for $\left\{ \begin{array}{l} s = m - 1 + \left\lceil -\log_2\left(\|\mathbf{h}\|_\infty + \varepsilon\right)\right\rceil \\ \quad \textbf{to} \quad \left\lfloor \log_2 \frac{N(2^m-1)}{\max\left\{\|\mathbf{h}\|_1 - \varepsilon, N\left(\|\mathbf{h}\|_\infty - \varepsilon\right)\right\}} \right\rfloor \end{array} \right\}$ do    ▷ from (4.4) and (4.5), pruned by 4.4.4(a)
        for $n = 1$ to $N - 1$ do
            $\xi_n \leftarrow \min_{x \in \{0,\ldots,2^m-1\}} \left| |h_n| - \frac{x}{2^s} \right|$          ▷ from (4.20), pruned by 4.4.4(b)
        end for
        $\varepsilon_0 \leftarrow \varepsilon - \sum_{i=1}^{N-1} \xi_i$                              ▷ from (4.21)
        $funds \leftarrow \varnothing$
        RECURSE$(0, 0, 0)$                              ▷ recursively execute nested loops
        if $cost^\star = 0$ then
            break
        end if
    end for
    return $(s^\star, \hat{\mathbf{h}}^\star, R'^\star)$
end procedure
```

4.4.5 Limitations

With the JOINT_SOLVE algorithm completely defined, we now discuss its limitations in practice. First, as mentioned in Section 4.2.3, Problem 4.3 does not always have a solution due to the depth and error constraints. If the constraints (4.1a) and (4.1b) taken alone preclude feasible $(s, \hat{\mathbf{h}})$ pairs from existing, then the $(s, \hat{\mathbf{h}})$ search space derived in Section 4.4.2 is empty and the algorithm will terminate quickly. Otherwise, HCUB_PRUNED would have to be executed for each pair in the $(s, \hat{\mathbf{h}})$ search space to determine if an $(s, \hat{\mathbf{h}}, R')$ triplet satisfying the additional constraints (4.1c) and (4.1d) exists.

This observation leads to the second limitation. That is, $\mathrm{H}_{\mathrm{cub}}$, in general, is a heuristic that does not always find an adder tree satisfying $\mathrm{depth}(R') \leq d$ even

Algorithm 4.4 Procedures called by JOINT_SOLVE

```
 1: procedure RECURSE ($n, \mathit{prevCost}_{\mathrm{LB}}, \mathit{prevDepth}_{\mathrm{LB}}$)
 2:     if $\exists i \in \{0, \ldots, n-1\}$ such that $|h_n| = |h_i|$ then        ▷ pruning by 4.4.4(c) to bypass loop
 3:         $\hat{h}_n \leftarrow |\hat{h}_i| \cdot \mathrm{sgn}(h_n)$
 4:         if $\hat{h}_n < h_n - \mathcal{E}_n$ or $\hat{h}_n > h_n + \mathcal{E}_n$ then        ▷ still should check within error bounds
 5:             return
 6:         end if
 7:         if $n = N - 1$ then
 8:             CHECK_IF_BETTER()
 9:         else
10:             $\mathcal{E}_{n+1} \leftarrow \mathcal{E}_n - |h_n - \hat{h}_n| + \xi_{n+1}$        ▷ from (4.22)
11:             RECURSE($n+1, \mathit{prevCost}_{\mathrm{LB}}, \mathit{prevDepth}_{\mathrm{LB}}$)
12:         end if
13:     else
14:         for $\left\{ \begin{array}{l} \hat{h}_n = \frac{\lceil \max\{0,(|h_n|-\mathcal{E}_n)2^s\} \rceil \cdot \mathrm{sgn}(h_n)}{2^s} \\ \quad \textbf{to} \quad \frac{\lfloor \min\{(|h_n|+\mathcal{E}_n)2^s, 2^m-1\} \rfloor \cdot \mathrm{sgn}(h_n)}{2^s} \end{array} \right\}$ do        ▷ from Line 5 and (4.23) and (4.24), pruned by 4.4.4(b)†
15:             $g \leftarrow \mathrm{GOF}(|\hat{h}_n| \cdot 2^s)$
16:             if $g \in \mathit{funds} \cup \{0\}$ then        ▷ nonunique fundamental
17:                 if $n = N - 1$ then
18:                     CHECK_IF_BETTER()
19:                 else
20:                     $\mathcal{E}_{n+1} \leftarrow \mathcal{E}_n - |h_n - \hat{h}_n| + \xi_{n+1}$        ▷ from (4.22)
21:                     RECURSE($n+1, \mathit{prevCost}_{\mathrm{LB}}, \mathit{prevDepth}_{\mathrm{LB}}$)
22:                 end if
23:             else
24:                 $\mathit{funds} \leftarrow \mathit{funds} \cup \{g\}$
25:                 if $\mathrm{cost}_{\mathrm{LB}}(\mathit{funds}) < \mathit{cost}^\star$ and $\mathrm{depth}_{\mathrm{LB}}(\mathit{funds}) \leq d$ then        ▷ pruning by 4.4.4(d) to skip current
26:                     if $n = N - 1$ then
27:                         CHECK_IF_BETTER()
28:                     else
29:                         $\mathcal{E}_{n+1} \leftarrow \mathcal{E}_n - |h_n - \hat{h}_n| + \xi_{n+1}$        ▷ from (4.22)
30:                         RECURSE($n+1, \mathrm{cost}_{\mathrm{LB}}(\mathit{funds}), \mathrm{depth}_{\mathrm{LB}}(\mathit{funds})$)
31:                     end if
32:                     $\mathit{funds} \leftarrow \mathit{funds} \setminus \{g\}$
33:                 end if
34:             end if
35:         end for
36:     end if
37: end procedure
```

†The loop steps by increments of $\mathrm{sgn}(h_n)/2^s$ in line 14.

```
1: procedure CHECK_IF_BETTER ()
2:     $(R', \mathit{valid}) \leftarrow$ HCUB_PRUNED($\mathit{funds}, d, \mathit{cost}^\star$)        ▷ $\mathrm{H_{cub}}$ pruned by 4.4.4(e)
3:     if $\mathit{valid}$ then
4:         $(s^\star, \hat{\mathbf{h}}^\star, R'^\star) \leftarrow (s, \hat{\mathbf{h}}, R')$
5:         $\mathit{cost}^\star \leftarrow \mathrm{cost}(R')$
6:     end if
7: end procedure
```

if one exists, nor does it guarantee any returned adder tree is optimal in terms of cost. On the other hand, analysis of $\mathrm{H}_{\mathrm{cub}}$ in [13] shows that these two situations occur infrequently. Nevertheless, JOINT_SOLVE inherits these limitations and, thus, it cannot always find a feasible triplet even if one exists, and a returned triplet may not be optimal. However, the algorithm does guarantee that any returned triplet is indeed a feasible triplet of Problem 4.3.

Next, we acknowledge that the asymptotic complexity of JOINT_SOLVE is large, being exponential in both the number of bits m and the vector length N. In fact, even the smaller problem of single constant multiplication, which is a special case of Problem 4.2 with $|F'| = 1$, has been shown to be NP-complete by Cappello and Steiglitz in [47]. On the other hand, the computation time of JOINT_SOLVE may be acceptable for most real-world applications because, in practice, the algorithm needs to be executed only *once* for a given system design, as opposed to a continuous, real-time operation. Furthermore, as will be shown in Section 4.5, even with a short runtime limit imposed on JOINT_SOLVE, it can still return useful results highlighting its benefits and, more generally, the benefit of the joint optimization framework.

4.5 Experiments and Results

In this section, we describe two extensive randomized experiments, which establish the benefit of the joint optimization framework and, in particular, our JOINT_SOLVE algorithm. Section 4.5.1 details the experimental setup, while Sections 4.5.2 and 4.5.3 describe the results obtained.

4.5.1 Experimental Setup

Consider two different experiments comparing the joint framework, which uses JOINT_SOLVE to solve Problem 4.3, with a disjointed one, which solves Problems 4.1 and 4.2 separately. The first experiment examines *arbitrary constants*, while the second examines *low-pass filter coefficients*. The specifics of each experiment are detailed following the general setup.

For each of the two experiments, we let N take on every fourth value between 4 and 80 to provide a good range of vector lengths. For each N, we let a parameter $\mathcal{E}'$ take on 10 values between 10^{-1} and 10^{-4}, spaced logarithmically evenly, from which the error bound $\mathcal{E}$ is derived according to the specific experiment being run (to be seen shortly). For each $(N, \mathcal{E}')$ pair, we perform 100 independent trials, each having an associated ideal constant vector $\mathbf{h}$ (either arbitrary constants or low-pass filter coefficients).

For simplicity and uniformity, Problem 4.1 was defined in Section 4.2.1 to take both an error bound $\mathcal{E}$ and a bitwidth m as constraints. However, the two constraints are not completely independent. For instance, as $\mathcal{E}$ gets smaller, m must become correspondingly larger to accomodate the error and allow for feasible solutions. In these experiments, we leverage this connection between $\mathcal{E}$ and m to more compactly represent the experiment parameters using only the $(N, \mathcal{E}')$ pair and also to guarantee feasibility in both the disjointed and joint frameworks. For this purpose, we define Problem 4.1′ having m as a variable to be minimized rather than as given. With this definition, Problem 4.1′ returns a triplet $(m^\star, s^\star, \hat{\mathbf{h}}^\star)$ which is the bit-optimal rounding of $\mathbf{h}$ satisfying the error bound.

For each trial, we compare the joint and disjointed frameworks in the following manner: For the disjointed framework, we first solve Problem 4.1′ by rounding to obtain $m^\star_{\text{disjointed}}$, $s^\star_{\text{disjointed}}$, and $\hat{\mathbf{h}}^\star_{\text{disjointed}}$. Then, we solve Problem 4.2 by invoking H_{cub} with the fundamentals corresponding to $s^\star_{\text{disjointed}}$ and $\hat{\mathbf{h}}^\star_{\text{disjointed}}$, and with zero depth. Upon completion, H_{cub} returns an $R'^\star_{\text{disjointed}}$ with minimum depth. For the joint framework, in order to ensure a fair comparison, we adopt the same error bound $\mathcal{E}$, same bitwidth $m = m^\star_{\text{disjointed}}$, and same depth allowance $d = \text{depth}(R'^\star_{\text{disjointed}})$.

Last, we invoke JOINT_SOLVE to solve Problem 4.3 and produce $R'^{\star}_{\text{joint}}$. In both frameworks, we record the adder tree costs for comparison.

The JOINT_SOLVE algorithm of Algorithm 4.3 is implemented in C++ and needs to be executed 40,000 times (since there are 2 experiments, 20 N's, 10 $\mathcal{E}'$'s, and 100 trials), but the execution is fully parallelizable. Hence, we carry out these 40,000 tasks at the University of Oklahoma Supercomputing Center for Education and Research, where each task is executed on its own dedicated processor (64-bit 2.0 GHz quad-core Intel E5405 [48]). It follows that the execution of JOINT_SOLVE on a given parameter set is performed in the same manner as an end user invoking JOINT_SOLVE to optimize a particular system design. Since the individual runtime of a task may vary depending on its parameters, a 1-hour time cap is imposed on each task to ensure that the experiments are completed in a reasonable time. Note that, in practice, a larger time cap may be affordable, since only one invocation of JOINT_SOLVE is required for a given system design. Moreover, we stress that the parallelization does not shorten the runtime of each individual task.

Now, for the first experiment, each $\mathbf{h} \in \mathbb{R}^N$ is a vector of arbitrary constants drawn randomly and independently from a uniform distribution on $(-1, 1)$. We choose a uniform distribution because we are examining the general case where there is no presumed relation amongst the target constants, though this same experiment may be performed using any other application-specific distribution. Also, we let $\mathcal{E} = \mathcal{E}' \cdot \| \mathbf{h} \|_1$, so that the constraint $\| \mathbf{h} - \hat{\mathbf{h}} \|_1 \leq \mathcal{E}$ becomes $\frac{\| \mathbf{h} - \hat{\mathbf{h}} \|_1}{\| \mathbf{h} \|_1} \leq \mathcal{E}'$, allowing $\mathcal{E}'$ to take on the meaning of percentage error (i.e., $\mathcal{E}' = 10^{-1}$ may be interpreted as a maximum of 10% error).

Next, for the second experiment, each $\mathbf{h} \in \mathbb{R}^N$ is a vector of low-pass filter coefficients generated by the Parks-McClellan algorithm [29] with normalized passband and stopband frequencies drawn randomly[1] from a uniform distribution on $(0, \pi)$. Given

[1] Since we are generating low-pass filters, the passband frequency is chosen as the smaller of the

a piecewise-linear frequency response and a filter length N, the Parks-McClellan algorithm generates the coefficiens of an N-tap linear-phase filter that minimizes the frequency response error in the minimax sense. We invoke this algorithm by calling the firpm() function in MATLAB [49] to generate an N-tap filter $\mathbf{h}$ to match a frequency response having unity gain in the passband and zero gain in the stopband. Note that the algorithm generates linear-phase filters and, thus, the coefficients are symmetric, leading to only half of the N coefficients being unique. Also, we let $\mathcal{E} = \mathcal{E}'$, instead of $\mathcal{E} = \mathcal{E}' \cdot \|\mathbf{h}\|_1$ as in the first experiment, because here $\mathbf{h}$ represents the coefficients of a unity-gain low-pass filter, and because in filter design, frequency response error in the sense of (3.3) is of interest more-so than the percentage error between $\mathbf{h}$ and $\hat{\mathbf{h}}$ (i.e., $\mathcal{E} = \mathcal{E}' = 10^{-1}$ may be interpreted as a maximum of 10% error in frequency response).

4.5.2 Results with Arbitrary Constants

For comparison purposes, we look at the adder tree costs resulting from each framework. The cost comparisons are expressed as the ratio $\text{cost}(R'^{\star}_{\text{joint}})/\,\text{cost}(R'^{\star}_{\text{disjointed}})$ in order to show the adder tree cost reduction associated with using Joint_Solve and the joint framework. For this first experiment, all of the ratios are collected for each $(N, \mathcal{E}')$ pair, averaged over 100 trials, and tabulated in Table 4.1. Also, a graphical depiction is presented in Figure 4.8. As a result of the manner in which the experiments are defined, the joint problem solution can be no worse than the disjointed one because the disjointed problem solution is guaranteed to be a feasible triplet of Problem 4.3. Thus, the ratios in Table 4.1 are never greater than 1. Note that, if $\text{cost}(R'^{\star}_{\text{disjointed}}) = \text{cost}(R'^{\star}_{\text{joint}}) = 0$ for a particular trial, the ratio is set to 1 to avoid dividing zero by zero.

Observe from Table 4.1 and Figure 4.8 that there is an improvement of 20% to

two randomly drawn values.

Table 4.1: Arbitrary constants—The ratio cost($R'^{\star}_{\text{joint}}$) / cost($R'^{\star}_{\text{disjointed}}$) for each pair ($N$, ε') averaged over 100 trials.

	$\boldsymbol{\varepsilon'}$									
$\boldsymbol{N}$	**1.0E-1**	**4.6E-2**	**2.2E-2**	**1.0E-2**	**4.6E-3**	**2.2E-3**	**1.0E-3**	**4.6E-4**	**2.2E-4**	**1.0E-4**
4	0.955	0.978	0.959	0.977	0.966	0.969	0.983	0.980	0.967	0.978
8	0.895	0.837	0.882	0.926	0.926	0.937	0.957	0.937	0.920	0.927
12	0.790	0.780	0.824	0.854	0.880	0.903	0.912	0.899	0.915	0.898
16	0.708	0.726	0.754	0.804	0.857	0.900	0.894	0.898	0.876	0.890
20	0.663	0.698	0.714	0.775	0.863	0.883	0.898	0.896	0.888	0.895
24	0.680	0.642	0.709	0.775	0.843	0.873	0.872	0.892	0.895	0.886
28	0.577	0.644	0.641	0.761	0.819	0.870	0.890	0.913	0.892	0.906
32	0.620	0.613	0.661	0.744	0.810	0.871	0.883	0.918	0.912	0.912
36	0.540	0.599	0.646	0.724	0.804	0.843	0.888	0.921	0.910	0.919
40	0.567	0.580	0.637	0.723	0.782	0.850	0.894	0.929	0.918	0.922
44	0.533	0.603	0.606	0.717	0.782	0.849	0.896	0.923	0.922	0.925
48	0.473	0.570	0.618	0.704	0.798	0.836	0.895	0.924	0.926	0.924
52	0.500	0.580	0.607	0.698	0.777	0.847	0.889	0.923	0.934	0.933
56	0.447	0.561	0.597	0.699	0.774	0.838	0.889	0.929	0.937	0.934
60	0.487	0.548	0.599	0.692	0.789	0.843	0.893	0.931	0.940	0.935
64	0.533	0.565	0.596	0.684	0.778	0.843	0.884	0.929	0.943	0.940
68	0.447	0.564	0.584	0.672	0.773	0.838	0.896	0.929	0.943	0.944
72	0.473	0.527	0.596	0.664	0.774	0.845	0.890	0.928	0.950	0.946
76	0.420	0.578	0.596	0.685	0.777	0.840	0.875	0.930	0.948	0.948
80	0.520	0.562	0.601	0.684	0.770	0.833	0.893	0.923	0.949	0.950

60% for moderate values of N and ε', with the best results occurring when N and ε' are large. In addition, these results are obtained with the 1-hour time cap in place, suggesting that enforcing a reasonable time limit does not prevent the possibility of large improvement by using Joint_Solve. To see where the 1-hour limit may affect the results, the execution time of each task was recorded as well. On average, the 1-hour time limit was only approached when $N > 40$ and $\varepsilon' < 4.6 \times 10^{-4}$. As expected, the computation time increases as N grows. Moreover, it also increases as ε' gets smaller due to more bits being required in the approximations when the ε' constraint is more restrictive.

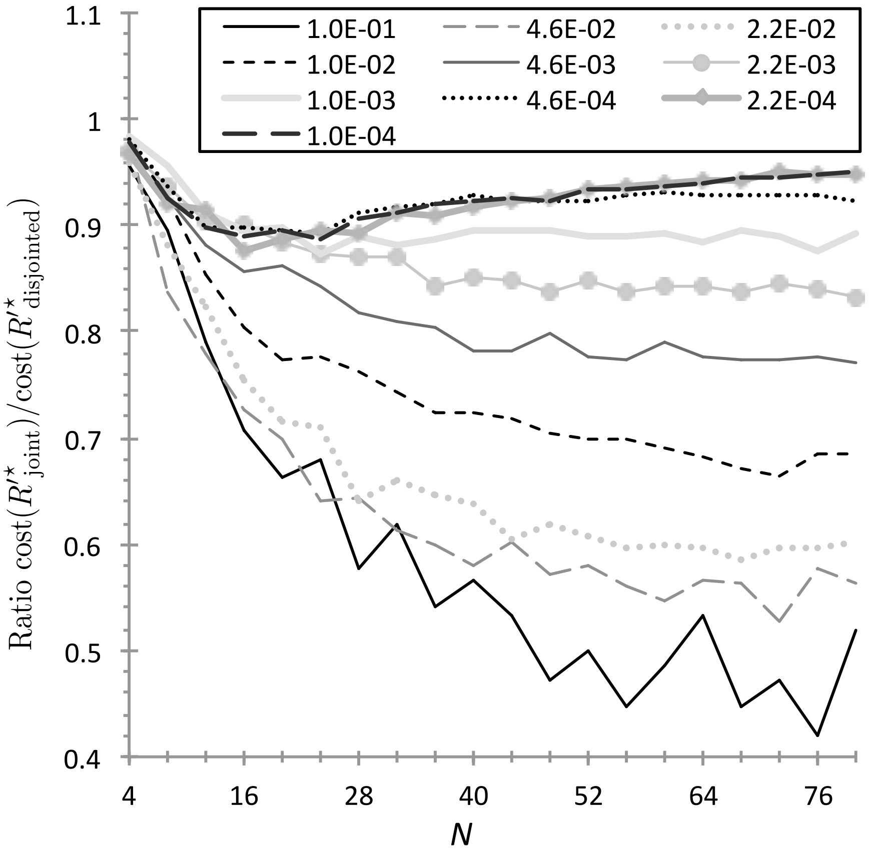


Figure 4.8: Arbitrary constants: The ratio $\text{cost}(R'^{\star}_{\text{joint}})/\ \text{cost}(R'^{\star}_{\text{disjointed}})$ versus vector length N for each fixed ε' (the values of ε' are specified in the legend). The ratios are averaged over 100 trials.

4.5.3 Results with Low-pass Filter Coefficients

As in Section 4.5.2, for this second experiment, all of the ratios are collected for each (N, ε'), averaged over 100 trials, and shown in Table 4.2 as well as Figure 4.9.

Observe from Table 4.2 and Figure 4.9 that there is an improvement of 15% to 30% for moderate values of N and ε', with the best results occurring when N and ε' are large. Similar to the first experiment, on average, the 1-hour time limit was only approached when $N > 64$ (32 unique coefficients) and $\varepsilon' < 4.6 \times 10^{-4}$. The cost improvement results, with respect to ε', turn out to be not as good as with arbitrary constants, since the coefficients have properties that are not exploited by JOINT_SOLVE. We note that while JOINT_SOLVE may be used in the case of

Table 4.2: Low-pass filter coefficients—The ratio cost($R'^{\star}_{\text{joint}}$) / cost($R'^{\star}_{\text{disjointed}}$) for each pair $(N, \boldsymbol{\varepsilon}')$ averaged over 100 trials.

	$\boldsymbol{\varepsilon'}$									
$\boldsymbol{N}$	**1.0E-1**	**4.6E-2**	**2.2E-2**	**1.0E-2**	**4.6E-3**	**2.2E-3**	**1.0E-3**	**4.6E-4**	**2.2E-4**	**1.0E-4**
4	0.990	1.000	1.000	0.995	1.000	1.000	1.000	1.000	0.993	0.997
8	0.968	0.976	0.961	0.984	0.962	0.963	0.981	0.985	0.978	0.986
12	0.931	0.878	0.947	0.923	0.959	0.977	0.964	0.957	0.970	0.945
16	0.877	0.894	0.947	0.931	0.948	0.938	0.926	0.931	0.923	0.929
20	0.923	0.918	0.899	0.878	0.880	0.869	0.884	0.920	0.929	0.926
24	0.856	0.875	0.820	0.829	0.850	0.891	0.927	0.941	0.939	0.911
28	0.852	0.792	0.781	0.843	0.894	0.930	0.945	0.924	0.891	0.881
32	0.795	0.780	0.795	0.856	0.905	0.910	0.885	0.886	0.878	0.865
36	0.799	0.809	0.828	0.837	0.886	0.884	0.844	0.862	0.854	0.878
40	0.754	0.821	0.827	0.852	0.834	0.806	0.852	0.854	0.880	0.892
44	0.757	0.795	0.788	0.785	0.806	0.834	0.839	0.872	0.892	0.923
48	0.774	0.768	0.776	0.795	0.803	0.819	0.862	0.891	0.907	0.930
52	0.733	0.758	0.726	0.757	0.790	0.820	0.878	0.907	0.928	0.933
56	0.694	0.744	0.777	0.786	0.787	0.849	0.901	0.928	0.930	0.919
60	0.696	0.703	0.763	0.773	0.794	0.869	0.912	0.930	0.922	0.910
64	0.681	0.736	0.719	0.748	0.840	0.881	0.922	0.929	0.928	0.918
68	0.694	0.705	0.729	0.769	0.844	0.892	0.913	0.928	0.918	0.918
72	0.715	0.694	0.732	0.785	0.847	0.885	0.907	0.920	0.915	0.911
76	0.683	0.713	0.734	0.784	0.839	0.872	0.900	0.911	0.917	0.924
80	0.697	0.748	0.753	0.779	0.816	0.864	0.894	0.913	0.920	0.925

FIR filters, we are unable to compare against the joint optimization algorithms from [12, 23, 24, 34–40] because they formulate the problem differently starting with some given frequency response specifications, rather than with an ideal real coefficient vector to approximate. For this reason, when designing FIR filters, jointly optimal solutions in the sense of Problem 4.3 might have higher adder tree costs than solutions obtained by algorithms in [12, 23, 24, 34–40], which are tailored specifically to FIR filters.

With that said, Joint_Solve still achieves 15%–30% improvement over a disjointed approach in the case of low-pass filter coefficients, and an even higher improvement of 20%–60% in the general case. Overall, the results demonstrate the substantial advantage of using the joint quantization and MCM optimization frame-

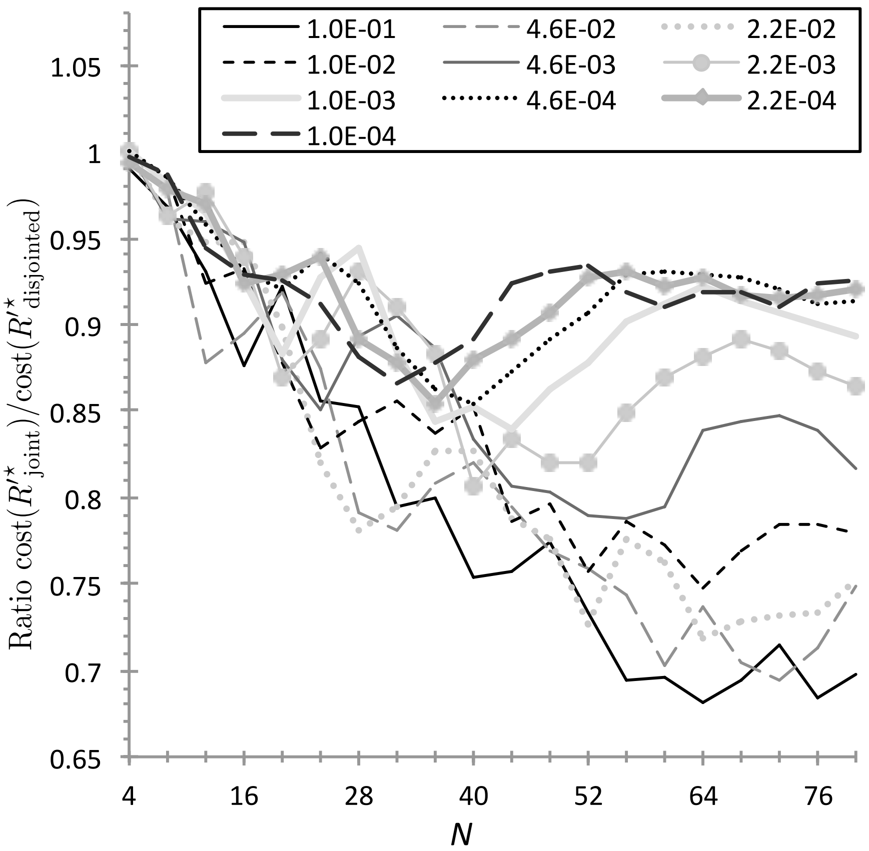


Figure 4.9: Low-pass filter coefficients: The ratio $\text{cost}(R'^{\star}_{\text{joint}})/\text{cost}(R'^{\star}_{\text{disjointed}})$ versus filter length N for each fixed ε' (the values of ε' are specified in the legend). The ratios are averaged over 100 trials.

work in conjunction with JOINT_SOLVE over a disjointed framework.

4.5.4 Additional Results

A question that may be of interest to the reader is how much impact the pruning strategies of Section 4.4.4 have on the execution time of the JOINT_SOLVE algorithm. To answer this question, we have recorded the time to execute the algorithm, both with and without pruning, on a subset of the trials from the first experiment outlined in Section 4.5.1. The trial specifications as well as the resulting execution times are listed in Table 4.3.

As expected, when JOINT_SOLVE is executed with the pruning strategies in effect,

Table 4.3: Execution times for JOINT_SOLVE with and without pruning for a select set of trials using arbitrary constants.

		Execution Time (s)	
N	ε'	**Without pruning**	**With pruning**
4	4.6E-2	7.4377	12.661
8	1.0E-1	6.7002	12.751
24	4.6E-2	122.73	12.643
28	2.2E-2	15.689	12.569
36	1.0E-1	161.85	13.700
40	1.0E-1	123.45	12.692

the total execution time is significantly less, in general. There is, however, an exception that occurs with the first two trials listed in Table 4.3. It can be understood that for small problem sizes such as these, the additional overhead from implementing the pruning strategies might outweigh any reduction in computational complexity, causing the pruned search to execute more slowly than the one without pruning.

4.6 Summary

In this chapter, we have examined the possibility of an MRCM approach to the quantization and MCM problems in a general case, going beyond the case of FIR filters which has been well-studied in the current literature. We have formalized the joint optimization problem and provided a relatively efficient algorithm for solving it. In doing so, we have shown the benefit that can arise with such a joint framework, namely, a notable reduction in the adder tree cost, which translates into reduced hardware or software. Indeed, this cost reduction has been shown via experiment to be quite significant in the general case, reaching 20%–60% for moderate vector lengths and error constraints. Furthermore, we have shown through a parallel experiment that our algorithm is useful in the particular case of FIR filters, achieving a reduction of 15%–30% for moderate filter lengths and ripple constraints.

Chapter 5

Multiple Real-Constant Multiplication with Bit-Count Cost Model

5.1 Introduction

This chapter formulates and solves the MRCM problem using an *adder-bit* cost model. Recall that, in Chapter 4, we provided an MRCM problem formulation using the traditional *adder-count* cost model and an algorithm that could be used to solve it. However, the problem formulation used the traditional *adder-count* cost model for optimization, that is, the number of adders in a shift-add network, whereas [24–27] show that, along with the number of adders, the adders' bitwidths play an important role in predicting hardware area, latency, and power. Also, the algorithm for solving the MRCM problem in Chapter 4 is limited in its efficiency and optimality, because it uses an embedded heuristic for solving the MICM portion of the problem.

To alleviate these issues, in Section 5.2, we define an improved cost model that incorporates adder bitwidth and, in Section 5.3, we provide two new algorithms for solving it, termed *greedy search* and *optimal search*, each incorporating searching strategies of high-performing MICM-targeted solvers [13, 21]. Additionally, we provide a new MRCM-aware heuristic to efficiently guide the searches. Next, Section 5.4 discusses a randomized experiment, in which we compare both algorithms to an MICM-targeted heuristic algorithm called $\mathrm{H}_{\mathrm{cub}}$ [13]. In doing so, we observe that the greedy search finds solutions with an average cost improvement of 13% over the MICM solution with the trials considered, and the optimal search finds solutions with an average cost improvement of 19% over the MICM solution with the same trials. Last, Section 5.5 summarizes the chapter's development and results.

5.2 Problem Formulation

As with Chapter 4, we note that much of the problem and algorithm development in this chapter is based on deprecated terminology that has been replaced in more recent work, because the research of this chapter comes from a conference paper that has been accepted for publication [50]. However, to reduce the possibility of introducing errors by rewriting every definition and developing the algorithm in a completely different manner, we maintain the same terminology of the conference version in lieu of the more recent terminology introduced in Chapter 3 and carried on in Chapter 6. As a result, we again caution the reader to be aware of certain terms taking on new definitions in this chapter.

The MRCM problem can be described as: given a vector of real constants $\mathbf{c}$ and a scalar fixed-point signal x, find a shift-add network having minimal hardware cost that computes the product $\mathbf{c} \cdot x$ subject to a given error bound. To precisely define the MRCM problem used in this chapter, we start with the following definitions. As with the MCM literature [13,16,24], we define a *fundamental* as a positive odd integer and we let $\mathbb{F}$ be the set of all fundamentals. Also, let the set of all positive fixed-point integers be defined as $\mathbb{P} = \{f \cdot 2^s : f \in \mathbb{F}, s \in \mathbb{Z}\}$. Now, given a fixed-point integer $p \in \mathbb{P}$, the *fundamentalization* of p, $\mathrm{fund}(p)$, is the unique fundamental that can be recovered from p. Furthermore, given a fixed-point vector $\mathbf{p} = (p_0, \ldots, p_{N-1}) \in \mathbb{P}^N$, we define $\mathrm{fund}(\mathbf{p})$ as the set of unique fundamentals that may be recovered from the components of $\mathbf{p}$.

We now define what it means for a set of fundamentals to be *MCM-capable.* Consider a shift-add network having N adders. At the output of these adders, the circuit produces the products $p_0 \cdot x \ldots p_{N-1} \cdot x$ which comprise the vector $\mathbf{p} \cdot x$. We say the set of fundamentals $\mathrm{fund}(\mathbf{p})$ is MCM-capable, since it can correspond to the outputs of a shift-add network. Let the set of all MCM-capable fundamental sets be denoted as ψ.

The hardware cost of such shift-add networks is now defined. Bit-level cost models are shown in [24, 26] to be more accurate than simple adder count models for estimating hardware cost of shift-add networks. Accordingly, for our problem formulation, we define the hardware cost as the total number of bits required for all of the adders/subtractors in the network. The models in [24, 26] consider the alignment of adder inputs in order to reduce the estimated cost, whereas our model in comparison uses a worst-case estimate. Using this model, the hardware cost of the aforementioned shift-add network producing $\mathbf{p} \cdot x$ at the adder outputs can be calculated as follows: The number of bits required to produce one binary output $p_n \cdot x$ is $\lfloor \log_2[\mathrm{fund}(p_n \cdot x)] \rfloor + 1$. Therefore the total cost to produce all of the outputs is $\sum_{n=0}^{N-1} \lfloor \log_2[\mathrm{fund}(p_n \cdot x)] \rfloor + N$.

With this setup, we formally define the MRCM problem as:

Problem 5.1 — MRCM problem with bit-count cost model: Given a vector of target constants $\mathbf{c} = (c_0, \ldots, c_{N-1}) \in (0, \infty)^N$, a corresponding vector of error tolerances $\boldsymbol{\varepsilon} = (\varepsilon_0, \ldots, \varepsilon_{N-1}) \in (0, \infty)^N$, and an input signal bitwidth b_x, find a target-approximation vector $\hat{\mathbf{c}} = (\hat{c}_0, \ldots, \hat{c}_{N-1}) \in \mathbb{P}^N$ and a set of MCM-capable fundamentals $F = \{f_1, \ldots, f_M\} \in \psi$ that minimize $\sum_{m=1}^{M} \lfloor \log_2[f_m \cdot (2^{b_x} - 1)] \rfloor + M$ such that $|c_n - \hat{c}_n| \leq \varepsilon_n \; \forall n$ and $(F \cup \{1\}) \supseteq \mathrm{fund}(\hat{\mathbf{c}})$.

The target-approximation vector $\hat{\mathbf{c}}$ must satisfy the error constraint. Note that because the cost function operates only on the fundamentals corresponding to adder outputs and because two fixed-point numbers with the same fundamentalization can be produced from each other with no additional adders, the problem considers only the shift-add networks that produce a unique set of fundamentals at their adder outputs. When these fundamental outputs include the target-approximation fundamentals then a candidate solution to the MRCM problem is obtained. Also note as a special case that the product $1 \cdot x$ can be computed from a shift-add network using no additional adders, and therefore F need never contain "1." The problem is guaranteed to have a solution, as discussed in Section 5.3.2.

5.3 Proposed Algorithms

In this section, two algorithms are proposed for solving the MRCM problem defined in the previous section. First, a scheme for enumerating the search space for MCM-capable F's is presented. Then, we describe a greedy algorithm that uses a heuristic for guiding the search of the aforementioned search space. Last, we describe an optimal algorithm that enumerates the search space in an efficient manner so as to reduce the required number of F's considered.

5.3.1 MCM-capable Search Tree Construction

This section describes the search space for ψ, that is, the space of MCM-capable fundamental sets. Not every $F \subset \mathbb{F}$ is in ψ, thus a strategy is needed to enumerate the space. As with the MICM algorithms of [13, 21], we enumerate the space by taking each $F \in \psi$ with cardinality $|F| = M$ and from it derive fundamental sets of the form $F' \in \psi$ with $|F'| = M + 1$, and so on, in an iterative manner. In doing so we construct a tree, in a similar manner to [21], where the nodes represent the elements of ψ.

Consider a node corresponding to $F = \{f_1, \ldots, f_M\} \in \psi$. Since $F \in \psi$, there is a shift-add network that produces the outputs $f_1 \cdot x, \ldots, f_M \cdot x$. Let the successor nodes of F be those that represent the set $F' = F \cup \{f_{M+1}\}$ with $f_{M+1} \notin F$ such that the

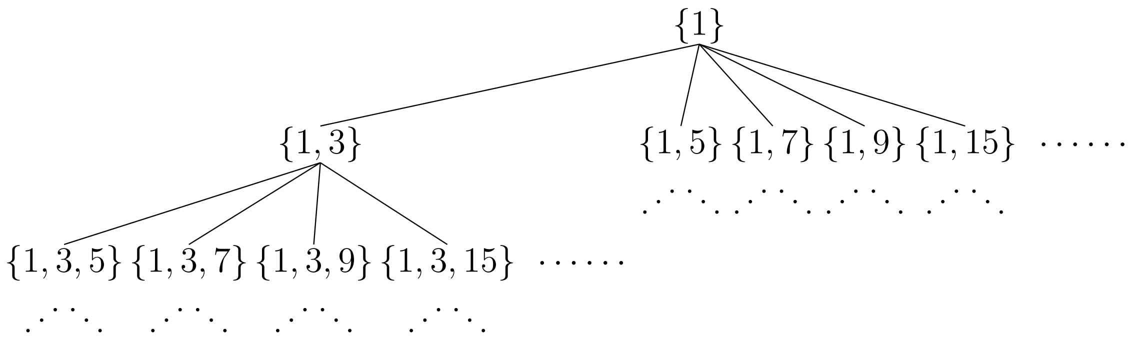


Figure 5.1: A portion of the ψ search tree illustrating node expansion.

product $f_{M+1} \cdot x$ can be computed by adding only one additional adder/subtractor to the aforementioned shift-add network. It can be shown that all possible f_{M+1} satisfying this property are of the form $f_{M+1} = (2^{l_1} f_i \pm 2^{l_2} f_j)2^{-r}$, where $i, j \leq M$ and $l_1, l_2, r \in \{0, 1, 2, \dots\}$. Operations of this form are called $\mathcal{A}$-operations [13]. Iterating through all possible $\mathcal{A}$-operations, allows for the generation of all of the successor nodes of F. The resulting tree structure is depicted in Figure 5.1.

Note that the ψ search tree is limitless in both breadth and depth and it grows exponentially at each level. Thus, only a small portion of the tree can be enumerated in a feasible amount of time. This fact necessitates that intelligent pruning and expansion ordering be used to reduce the number of nodes that are expanded before a solution F is found. Such intelligent search strategies are discussed in Sections 5.3.2 and 5.3.3.

5.3.2 Greedy Algorithm

This section describes an algorithm that uses a greedy strategy for traversing the ψ search tree to find a candidate solution to the problem defined in Section 5.2. A *candidate solution* is a pair $(F, \hat{\mathbf{c}}) \in \psi \times \mathbb{P}^N$ satisfying the problem constraints $|c_n - \hat{c}_n| \leq \varepsilon_n \; \forall n$ and $(F \cup \{1\}) \supseteq \text{fund}(\hat{\mathbf{c}})$. Let any F that exists as part of a candidate solution be termed a *candidate fundamental set*.

Using a greedy searching strategy, one never expands more than one node at a particular level of the search tree, i.e., there is no backtracking. Instead, given a set of successor nodes, a heuristic estimator is used to determine which is the *best* node for expansion. Therefore, greedy strategies can lead to a fast traversal of the search tree since the number of expanded nodes is pruned substantially. However, since there is no backtracking, a suboptimal candidate solution may be found. In fact, a poor heuristic can lead to candidate solutions that are far from optimal.

For our algorithm, we define a heuristic as follows: Given an F, let $\mathbf{c}' \in (0, \infty)^{N'}$ be composed of the target constants that cannot yet be approximated within their error bounds using F. In other words, $\mathbf{c}'$ contains those components c_n such that there does not exist a $\hat{c}_n \in \mathbb{P}$ where $\text{fund}(\hat{c}_n) \in F$ and $|c_n - \hat{c}_n| \leq \varepsilon_n$. Let $\boldsymbol{\varepsilon}' \in (0, \infty)^{N'}$ be the corresponding error tolerances.

Next, consider the fundamental sets that can be used to approximate the remaining target constants, i.e., the elements of $F'_{\hat{\mathbf{c}}} = \{\text{fund}(\hat{c}') : \hat{c}' \in \mathbb{P}^{N'} \text{ and } |c'_n - \hat{c}'_n| \leq \varepsilon'_n \ \forall n\}$. Let us assume that for any set $F' \in F'_{\hat{\mathbf{c}}}$, the union $(F \cup F') \in \psi$ and is thus a candidate fundamental set. With this assumption, we can estimate the minimum additional adder cost required to produce a candidate solution as:

$$H(F) = \min_{F' = \left(f'_1, \ldots, f'_{|F'|}\right) \in F'_{\hat{\mathbf{c}}}} \text{cost}(F'),$$

$$\text{where } \text{cost}(F) = \sum_{m=1}^{|F|} \lfloor \log_2[f_m(2^{b_x} - 1)] \rfloor + |F|.$$

Likewise, we can estimate the minimum total adder cost of a candidate solution as $T(F) = H(F) + \text{cost}(F)$.

Finally, the greedy search uses this heuristic estimator in its expansion scheme. When choosing between successor nodes, the node with the smallest H is expanded. If there are ties then the tiebreaker is to expand a node with smallest T. Then, after a node is expanded, its successors are generated and one is chosen for expansion. The search continues in this manner until a candidate solution is found. This greedy strategy can be shown to always terminate with a candidate solution, because the expansion scheme leads to a path $F_1 \Rightarrow F_2 \Rightarrow \cdots \Rightarrow F_K$ with $H(F_k)$ eventually decreasing until $H(F_K) = 0$.

As an artifact of the greedy strategy, at the termination of the algorithm it is possible that extra nodes exist in the path that are not needed to produce the final fundamental set. To cope with this, a redundant removal algorithm similar to the one in [21] is applied at the end of the search to remove fundamentals that are not used for approximating the target constants and are not needed to produce the final fundamental set.

5.3.3 Optimal Algorithm

This section describes a search algorithm which also uses a heuristic estimator for intelligent node expansion ordering. However, the search is performed in an optimal manner, rather than in a greedy manner as the previous section's algorithm does. Because the heuristic estimator is not exact, an optimal search requires that every search node that could possibly lead to an optimal solution be expanded. Thus, the key to designing an efficient optimal search is to use an intelligent strategy for determining node expansion order and to prune nodes when possible based on known cost upper bounds.

The search strategy used for the algorithm of this section is the so-named A* search [51]. This strategy uses a heuristic to estimate the best remaining cost to reach a candidate solution from a node. Then it adds the result to the node's current path cost to estimate the overall minimum path cost obtainable from that node. If the heuristic is admissible, meaning that it never overestimates the obtainable cost, then A* is optimal [51].

The heuristic $H(F)$ from the previous section is such a heuristic. It is admissible because the only assumption made in estimating the remaining adder cost is that for any $F' \in F'_{\hat{\mathbf{c}}}$, $(F \cup F') \in \psi$. This assumption leads to a cost estimate that is never greater that the optimal remaining cost.

With this property satisfied, H is suitable for use in the A* search. To perform the search, a priority queue is implemented, where the nodes with smallest T have highest priority for expansion. When a node is expanded, all of its successors are added to the queue based on their priority (T), then the highest priority node is dequeued and expanded next.

Pruning based on cost is accomplished as follows: Let cost_{min} be the current best candidate solution cost, which is initially set equal to the result of the greedy algorithm. Because the estimator T provides and underestimate of the optimal total cost that could be obtained by expanding a node, no node with $T \geq \text{cost}_{\text{min}}$ is ever added to the queue. Furthermore, when cost_{min} is updated from a new candidate solution, any node with $T \geq \text{cost}_{\text{min}}$ already in the queue can be removed. Note that keeping a list of visited nodes is important, because the MCM-capable search tree contains many redundant nodes (having the same fundamental set) that need only be visited once. The search ends when the queue is empty and the final cost_{min} is guaranteed to be optimal since every node that could possibly lead to a solution with less cost than cost_{min} is expanded.

5.4 Experiments and Results

This section describes the experimental setup and results, whereby the efficacy of the algorithms presented in Sections 5.3.2 and 5.3.3 is validated. A set of trial specifications were randomly generated and then, for each trial, a solution to the MRCM problem of Section 5.2 is obtained using three different algorithms.

To begin, the trial specifications are defined in Table 5.1. The four parameters N, b_x, $\mathbf{c}$, and $\boldsymbol{\varepsilon}$ specify the givens of the problem to be solved. The problem parameter

Table 5.1: Trial specifications using bit-count cost model.

Trial	**N**	$\boldsymbol{b_f}$	$\boldsymbol{b_x}$	**pairs** $\boldsymbol{(c_0, \varepsilon_0), \ldots, (c_{N-1}, \varepsilon_{N-1})}$
1	4	16	16	(1.516, 3.51E-3), (1.641, 3.63E-2), (1.837, 7.27E-3), (1.393, 2.12E-1)
2	4	16	32	(1.715, 8.69E-3), (1.130, 6.41E-3), (1.842, 5.13E-2), (1.366, 1.27E-1)
3	8	16	16	(1.905, 1.54E-1), (1.212, 4.71E-2), (1.581, 2.97E-3), (1.733, 2.36E-1), (1.754, 4.73E-3), (1.397, 1.25E-2), (1.524, 5.48E-3), (1.861, 1.25E-2)
4	8	12	32	(1.232, 6.78E-3), (1.015, 8.91E-2), (1.400, 2.25E-2), (1.242, 7.30E-4), (1.269, 1.19E-1), (1.160, 8.32E-3), (1.628, 1.03E-3), (1.406, 8.87E-4)
5	8	8	8	(1.025, 5.89E-3), (1.703, 1.51E-2), (1.141, 1.15E-2), (1.490, 2.23E-1), (1.337, 3.91E-2), (1.106, 1.81E-1), (1.302, 4.11E-2), (1.073, 2.37E-2)
6	8	8	16	(1.505, 3.96E-3), (1.783, 3.00E-2), (1.907, 3.18E-2), (1.143, 6.51E-2), (1.046, 9.24E-3), (1.337, 4.38E-3), (1.943, 1.16E-2), (1.014, 2.05E-2
7	16	8	8	(1.428, 1.31E-1), (1.323, 9.22E-2), (1.863, 7.62E-2), (1.220, 5.55E-3), (1.384, 3.18E-2), (1.351, 4.91E-3), (1.226, 8.63E-2), (1.241, 2.13E-2), (1.589, 1.46E-1), (1.942, 4.22E-3), (1.792, 3.21E-2), (1.329, 1.45E-1), (1.126, 1.01E-1), (1.658, 1.04E-2), (1.822, 7.08E-2), (1.804, 8.03E-2
8	16	8	16	(1.216, 1.23E-2), (1.097, 2.63E-2), (1.332, 4.84E-3), (1.450, 2.14E-2), (1.388, 6.38E-3), (1.149, 1.65E-1), (1.017, 8.01E-2), (1.803, 8.31E-2) (1.757, 7.44E-2), (1.107, 2.22E-2), (1.568, 1.64E-1), (1.361, 1.45E-1), (1.835, 5.46E-2), (1.170, 2.25E-1), (1.983, 1.98E-1), (1.821, 5.91E-3

N and the extra parameter b_f determine the *hardness* of the problem as follows: N random target constants $c_0, \ldots, c_{N-1}$ are drawn uniformly from the range $(1, 2)$. Because the fractional point location of a constant c_n has no effect on the problem solution, there is no need to consider a larger range. Next, the error tolerance vector $\boldsymbol{\varepsilon}$ is randomly chosen so that each error tolerance ε_n is such that 2^{ε_n} is drawn uniformly from the range $(-b_f, -2)$. In this manner, the parameter b_f roughly corresponds to an upper limit on the bitwidths of fundamentals needed to obtain a candidate solution. Table 5.1 provides trials with varying values of the parameters N, b_f, and b_x for comparison purposes.

Next, for each random trial, three algorithms are used to solve the MRCM problem. The first algorithm used is the $\mathrm{H_{cub}}$ algorithm presented in [13]. $\mathrm{H_{cub}}$ is an algorithm that solves the MICM problem using a greedy strategy, and it was chosen because it is one of the highest performing heuristic MICM solvers, and its source code is publicly available [46]. However, because $\mathrm{H_{cub}}$ solves the MICM problem and not our MRCM problem, the real constant vector $\mathbf{c}$ must first be transformed into a

Table 5.2: Summary of results using bit-count cost model.

Trial	**Rounding + $\mathbf{H_{cub}}$[13] solution**		**Greedy solution**		**Optimal solution**	
	fundamentals	**cost**	**fundamentals**	**cost**	**fundamentals**	**cost**
1	1 ⇒ 3 ⇒ 13 ⇒ 97 ⇒ 253 ⇒ 59	107	1 ⇒ 3 ⇒ 13 ⇒ 97 ⇒ 7 ⇒ 59	102	1 ⇒ 7 ⇒ 13 ⇒ 97 ⇒ 59	84
2	1 ⇒ 5 ⇒ 9 ⇒ 15 ⇒ 55	145	1 ⇒ 5 ⇒ 9 ⇒ 15 ⇒ 55	145	1 ⇒ 5 ⇒ 9 ⇒ 15 ⇒ 55	145
3	1 ⇒ 3 ⇒ 5 ⇒ 7 ⇒ 45 ⇒ 101 ⇒ 119 ⇒ 195	148	1 ⇒ 5 ⇒ 7 ⇒ 119 ⇒ 357 ⇒ 101 ⇒ 389	134	1 ⇒ 7 ⇒ 119 ⇒ 5 ⇒ 45 ⇒ 101 ⇒ 195	130
4	1 ⇒ 5 ⇒ 37 ⇒ 45 ⇒ 79 ⇒ 159 ⇒ 675 ⇒ 417	273	1 ⇒ 5 ⇒ 37 ⇒ 45 ⇒ 159 ⇒ 315 ⇒ 13 ⇒ 417	269	1 ⇒ 9 ⇒ 37 ⇒ 157 ⇒ 159 ⇒ 417 ⇒ 45	233
5	1 ⇒ 3 ⇒ 11 ⇒ 17 ⇒ 21 ⇒ 73 ⇒ 109 ⇒ 131	94	1 ⇒ 17 ⇒ 21 ⇒ 145 ⇒ 435 ⇒ 527	77	1 ⇒ 17 ⇒ 21 ⇒ 147 ⇒ 131 ⇒ 109	73
6	1 ⇒ 9 ⇒ 31 ⇒ 29 ⇒ 67 ⇒ 101 ⇒ 171 ⇒ 193	156	1 ⇒ 9 ⇒ 31 ⇒ 29 ⇒ 67 ⇒ 3 ⇒ 193 ⇒ 343	152	1 ⇒ 9 ⇒ 135 ⇒ 171 ⇒ 31 ⇒ 29 ⇒ 193	134
7	1 ⇒ 3 ⇒ 5 ⇒ 7 ⇒ 9 ⇒ 11 ⇒ 15 ⇒ 29⇒ 39 ⇒ 53 ⇒ 173 ⇒ 249	141	1 ⇒ 5 ⇒ 39 ⇒ 29 ⇒ 53 ⇒ 693 ⇒ 995	88	1 ⇒ 5 ⇒ 39 ⇒ 29 ⇒ 53 ⇒ 693 ⇒ 995	88
8	1 ⇒ 3 ⇒ 7 ⇒ 9 ⇒ 15 ⇒ 23 ⇒ 35 ⇒ 39 ⇒ 85 ⇒ 89 ⇒ 233	212	1 ⇒ 3 ⇒ 23 ⇒ 35 ⇒ 39 ⇒ 85 ⇒ 89 ⇒ 935	155	1 ⇒ 3 ⇒ 23 ⇒ 35 ⇒ 39 ⇒ 85 ⇒ 89 ⇒ 233	153

suitable vector of integers (fixed-point numbers). To accomplish this transformation, each real constant is rounded to a fixed-point constant using the least number of bits, such that the corresponding error tolerances are met. Last, $\mathrm{H_{cub}}$ solves the MICM problem using these transformed constants. The second and third algorithms used are the greedy algorithm and optimal algorithm presented in Sections 5.3.2 and 5.3.3 respectively.

The solutions obtained using all three algorithms are tabulated in Table 5.2. The results show that the greedy solution is strictly better than the $\mathrm{H_{cub}}$ solution in 7 out of the 8 trials, which is expected because the greedy algorithm tries to minimize the specified cost metric when expanding fundamentals, while $\mathrm{H_{cub}}$ tries to minimize the number of fundamentals. Also as expected, the optimal solution is always better than (or the same as) the other algorithms, because the algorithm is indeed optimal. In 6 out of the 8 trials, the optimal solution is strictly better than that obtained by the suboptimal greedy algorithm. Of course, the optimal algorithm takes much more computation time than the other two algorithms which use a greedy strategy.

In addition, Trial 7 in particular illustrates the benefit of using the MRCM problem as compared to the MICM problem, because the MRCM problem allows a real target constant to be approximated using a lesser number of fundamentals in situations where multiple targets can map to the same fundamental. Overall, for these trials, the greedy solutions showed an average cost improvement of 13% over the *Rounding + H_{cub}* strategy. Likewise, the optimal solutions showed an average improvement of 7% over the greedy strategy. Finally, the optimal solutions showed an average improvement of 19% over the *Rounding + H_{cub}* strategy.

5.5 Summary

In this chapter, we further showed the efficacy of the MRCM problem that we introduced in Chapter 3 with regard to an adder-bit cost model. Compared to the those of Chapter 4, we provided an improved problem formulation and two new algorithms for solving it, referred to as the *greedy* and *optimal* algorithms, respectively. The improved problem formulation uses the bitwidth of adders in its cost model, since they have been shown to be more accurate predictors of actual hardware cost. Last, an experiment was performed in which the greedy and optimal algorithms obtained solutions with significant improvement over those obtained by rounding constants and using an MICM-targeted algorithm $\mathrm{H_{cub}}$.

Chapter 6

Multiple Real-Constant Multiplication with Gate-Level Cost Model

6.1 Introduction

In Section 2.1.3, gate-level cost models for MCM were discussed. Experiments from [24–27] have shown that such lower-level cost models have a higher correspondence to synthesized hardware area and power utilization, as compared to the traditional adder-count cost model.

As with Chapters 4 and 5, it can be expected that in a general MCM setting with real constants there is potential for lower-costing circuits when utilizing an MRCM framework rather than the traditional MICM framework. For this reason, this chapter discusses a joint MRCM approach to MCM optimization using this gate-level cost model in order to show its advantages over a disjointed rounding+MICM approach presented in [26].

The chapter is organized as follows: Section 6.2 provides a detailed problem formulation for this MRCM approach; Section 6.3 discusses a greedy algorithm for solving the defined problem; Section 6.4 details an experiment for validating the joint approach's advantages; and, lastly, Section 6.5 summarizes the experiment's findings.

6.2 Problem Formulation

In this chapter's problem formulation, note that we return to the original terminology defined in Chapter 3 and discard any of the local modifications of the terminology that were introduced in Chapters 4 and 5.

The cost model used in the MRCM problem of this chapter is the gate-level cost model defined in [25, 26] It is chosen over the model defined in [24, 27] because it has incurred more recent development in the literature and it is defined in a way that better fits with the MCM terminology used in this chapter.

In Section 2.1.3, a cursory description was given of gate-level cost models, however, in [26] a very detailed model is presented. In that paper, it was shown that $\mathcal{A}$-configurations (and thus $\mathcal{A}$-relations) can be divided into five basic types. For each type, the authors of [26] derive a formula for computing a gate-level hardware cost based on the relative bitwidths of the $\mathcal{A}$-relation's inputs and output along with the bit-level alignment of the inputs arising from shifts.

Definition 6.1 (cost_{GL})**:** Since the formula definitions for computing the cost of an $\mathcal{A}$-relation require much more development than is available here, we simply provide the terminology $\text{cost}_{\text{GL}}(\mathbf{r})$ to refer to the cost that is computed for a particular $\mathcal{A}$-relation $\mathbf{r}$. We do, however, define the cost of the root $\mathcal{A}$-relation as $\text{cost}_{\text{GL}}(\mathbf{1}) = 0$ to align with the problem definition of this chapter. Additionally, given an adder tree $\mathbf{T} = \{\mathbf{r}_0, \ldots, \mathbf{r}_{M-1}\}$, the total gate-level cost can be computed by summing up the costs of its component $\mathcal{A}$-relations to obtain $\text{cost}_{\text{GL}}(\mathbf{T}) = \sum_{m=0}^{M-1} \text{cost}_{\text{GL}}(\mathbf{r}_m)$.

With regards to the error metric, in the MRCM problem of this chapter uses the 1-norm as defined in Section 3.2.1.

With the cost model and error metric established, the general MRCM problem statement of Section 3.4 (Problem 3.2) can be fully specified as follows:

Problem 6.1 — The MRCM problem using a gate-level cost model and ∞-norm error metric: Given a vector length $N \in \mathbb{N}$, an ideal constant vector $\mathbf{c} \in \mathbb{R}_b^N$, an error bound $\varepsilon > 0$, and a number of bits $b \in \mathbb{N}$, find a finite-bitwidth constant vector $\hat{\mathbf{c}} \in \mathbb{Z}_b^N$, and an adder tree $\mathbf{T} \in \psi(\Phi(\hat{\mathbf{c}}))$ that minimize $\text{cost}_{\text{GL}}(\mathbf{A})$

subject to $\|\mathbf{c} - \hat{\mathbf{c}}\|_1 \leq \varepsilon$.

6.3 Proposed Algorithm

In this section, an algorithm is developed for solving Problem 6.1. First, in Section 6.3.1, a framework is developed for specifying the problem solution in a functional sense. Then, in Section 6.3.2, a greedy algorithm is generated based on this framework.

6.3.1 Complete Functional Problem Definition

In this section, Problem 6.1 is transformed into a completely specified functional programming description. The functional programming paradigm is distinct from the traditional imperative programming paradigm where programs are organized as a sequence of instructions to be executed in order. With functional programming, programs are organized instead as a set of functions having no program state. Thus, a functional approach to algorithm implementation allows for a very simple algorithm description, and can, in a sense, directly translate to code using a functional language without having ambiguities in the implementation. With that concept in mind, we now define a functional description that encompasses Problem 6.1.

To begin, solutions of Problem 6.1 take the form of pairs $(\hat{\mathbf{c}}, \mathbf{T})$. Feasible pairs are those pairs from the set:

$$\{(\hat{\mathbf{c}}, \mathbf{T}) \in \mathbb{Z}_b^N \times N \mid \mathbf{T} \in \psi(\Phi(\hat{\mathbf{c}})), \|\mathbf{c} - \hat{\mathbf{c}}\|_1 \leq \varepsilon\}. \tag{6.1}$$

In order to enumerate the search space of feasible pairs, we note that for any adder tree $\mathbf{T} = \{\mathbf{r}_0, \ldots, \mathbf{r}_{M-1}\}$ with $M > 1$, the ordered set $\mathbf{T}' = \{\mathbf{r}_0, \ldots, \mathbf{r}_{M-2}\}$ having one less element than $\mathbf{T}$ is also an adder tree by Definition 4.6. Thus, it follows that the set of all adder trees $\mathbb{T}$ can be built up incrementally by starting with $\mathbf{T} = \{\mathbf{1}\}$ and appending $\mathcal{A}$-relations one at a time while ensuring that all properties of Definition 4.6 hold for each new set. To this end, we borrow some results and terminology from [13] to facilitate this incremental build-up of $\mathbb{T}$, modifying the definitions as necessary to fully support Problem 6.1.

Definition 6.2 (Successor relations, $\mathrm{S}_{\mathcal{A}}$)**:** Given an adder tree $\mathbf{T} = \{\mathbf{r}_0, \ldots, \mathbf{r}_{M-1}\}$, let $\mathrm{S}_{\mathcal{A}}$ be the set of $\mathcal{A}$-relations that can be added to $\mathbf{T}$ and have it still be an adder tree, that is, $\{\mathbf{r}' \mid \{\mathbf{r}_0, \ldots, \mathbf{r}_{M-1}, \mathbf{r}'\} \in \mathbb{T}\}$. We term these $\mathcal{A}$-relations, the *successor relations* of $\mathbf{T}$.

In [23, 26], it was shown that $\mathcal{A}$-configurations can be separated into five basic types corresponding to:

1) $2^{l_1}u + v$[1],

2) $\Phi(u + v)$,

3) $2^{l_1}u - v$[1],

4) $u - 2^{l_2}v$[1], and

5) $\Phi(u - v)$[1],

where $l_1, l_2 \neq 0$. Using this observation, given a $\mathbf{T} = \{\mathbf{r}_0, \ldots, \mathbf{r}_{M-1}\}$, we can generate the set of successor relations corresponding to each type of $\mathcal{A}$-configuration. Since we are only interested in those $\mathcal{A}$-relations with an output $w \in \mathbb{Z}_b$[2], the formulas defining the successor relation can be computed as:

1) $\mathrm{S}_{\mathcal{A},1}(\mathbf{T}) = \big\{(u, v, (l_1, 0, 0), w) \mid \exists i \in \{0, M{-}1\}\ u = \mathbf{r}_i.w, \exists j \in \{0, M{-}1\}\ v = \mathbf{r}_j.w,$ $w = \mathcal{A}_{(l_1,0,0)}(u, v),\ l_1 \in \mathbb{Z}^+, \text{ and } 1 \leq l_1 \leq \log_2 \frac{2^b-1-v}{u}\big\};$

[1] Of course, there also exist the operations that occur when the inputs u and v are switched.

[2] We are also only interested in $\mathcal{A}$-relations with $w \geq 3$ since an adder tree already contains the root $\mathcal{A}$-relation $\mathbf{1}$ having $\mathbf{1}.w = 1$.

2) $\mathrm{S}_{\mathcal{A},2}(\mathbf{T}) = \big\{(u, v, (0, 0, 0), w) \mid \exists i \in \{0, M-1\}\ u = \mathbf{r}_i.w,\ \exists j \in \{0, M-1\}\ v = \mathbf{r}_j.w,$ and $w = \mathcal{A}_{(0,0,0)}(u, v)\big\}$;

3) $\mathrm{S}_{\mathcal{A},3}(\mathbf{T}) = \big\{(u, v, (l_1, 0, 1), w) \mid \exists i \in \{0, M-1\}\ u = \mathbf{r}_i.w,\ \exists j \in \{0, M-1\}\ v = \mathbf{r}_j.w,$ $w = \mathcal{A}_{(l_1,0,1)}(u, v)$, $l_1 \in \mathbb{Z}^+$, and $\max\{1, \log_2 \frac{3+v}{u}\} \leq l_1 \leq \log_2 \frac{2^b-1+v}{u}\big\}$;

4) $\mathrm{S}_{\mathcal{A},4}(\mathbf{T}) = \big\{(u, v, (0, l_2, 1), w) \mid \exists i \in \{0, M-1\}\ u = \mathbf{r}_i.w,\ \exists j \in \{0, M-1\}\ v = \mathbf{r}_j.w,$ $w = \mathcal{A}_{(0,l_2,1)}(u, v)$, $l_2 \in \mathbb{Z}^+$, and $1 \leq l_2 \leq \log_2 \frac{u-3}{v}\big\}$; and

5) $\mathrm{S}_{\mathcal{A},5}(\mathbf{T}) = \big\{(u, v, (0, 0, 1), w) \mid \exists i \in \{0, M-1\}\ u = \mathbf{r}_i.w,\ \exists j \in \{0, M-1\}\ v = \mathbf{r}_j.w,$ $u \geq v + 6$, and $w = \mathcal{A}_{(0,0,1)}(u, v)\big\}$;

respectively. Combining 1)–5) allows the successor relations of $\mathbf{T}$ to be rewritten as $\mathrm{S}_{\mathcal{A}}(\mathbf{T}) = \mathrm{S}_{\mathcal{A},1}(\mathbf{T}) \cup \mathrm{S}_{\mathcal{A},2}(\mathbf{T}) \cup \mathrm{S}_{\mathcal{A},3}(\mathbf{T}) \cup \mathrm{S}_{\mathcal{A},4}(\mathbf{T}) \cup \mathrm{S}_{\mathcal{A},5}(\mathbf{T})$.

Definition 6.3 (Successor trees, $\mathrm{S}(\mathbf{T})$)**:** Furthermore, we define the *successor trees* of $\mathbf{T}$ as those adder trees that result from augmenting $\mathbf{T}$ with each successor relation. More precisely, given an adder tree $\mathbf{T} = \{\mathbf{r}_0, \ldots, \mathbf{r}_{M-1}\}$, the set of *successor trees* is defined as $\mathrm{S}(\mathbf{T}) = \{\{\mathbf{r}_0, \ldots, \mathbf{r}_{M-1}, \mathbf{r}'\} \mid \mathbf{r}' \in \mathrm{S}_{\mathcal{A}}(\mathbf{T})\}$.

With the successor function fully defined, the entire set of adder trees can be computed recursively as follows:

$$\mathbb{T} = \{\mathbf{1}\} \cup \left(\bigcup_{\mathbf{T}' \in \mathrm{S}(\mathbf{1})} \left[\{\mathbf{T}'\} \cup \left(\bigcup_{\mathbf{T}'' \in \mathrm{S}(\mathbf{T}')} [\{\mathbf{T}''\} \cup \ldots] \right) \right] \right).$$

Definition 6.4 ($\mathrm{S}_{\mathrm{rec}}$)**:** To simplify the recursion, let

$$\mathrm{S}_{\mathrm{rec}}(\mathbf{T}) = \{\mathbf{T}\} \cup \left(\bigcup_{\mathbf{T}' \in \mathrm{S}(\mathbf{T})} \mathrm{S}_{\mathrm{rec}}(\mathbf{T}') \right). \tag{6.2}$$

Then it follows that

$$\mathbb{T} = \mathrm{S}_{\mathrm{rec}}(\{\mathbf{1}\}). \tag{6.3}$$

Next, observe that for any feasible pair $(\mathbf{T}, \hat{\mathbf{c}}) \in \mathbb{T} \times \mathbb{Z}_b$,

$$\mathbf{T} \in \psi(\Phi(\hat{\mathbf{c}})). \tag{6.4}$$

However, since an enumeration scheme for $\mathbf{T} \in \mathbb{T}$ can be inferred from (6.3), it would be useful to have a method to inversely derive $\hat{\mathbf{c}}$, given a $\mathbf{T}$, such that (6.4) holds. To this end, we now provide a series of derivations to accomplish this task.

Definition 6.5 (Υ)**:** We first define $\Upsilon(\mathbf{T})$ to be the set of output fundamentals that an adder tree produces, that is, given an adder tree $\mathbf{T}$, let $\Upsilon(\mathbf{T}) = \{\mathbf{r}.w \in \mathbb{F} \mid \mathbf{r} \in \mathbf{T}\} \cup \{0\}$.

In a way, Υ acts as an inverse function of ψ, because for any $\mathbf{T}$ and F, $\mathbf{T} \in \psi(F)$ if and only if $\Upsilon(\mathbf{T}) \supseteq F$.

Using this definition, the constraint in (6.4) can be rewritten as $\Phi(\hat{\mathbf{c}}) \subseteq \Upsilon(\mathbf{T})$, which defines a search space of $\hat{\mathbf{c}}$ as a function of $\mathbf{T}$. Decomposing $\Phi(\hat{\mathbf{c}})$ using Definition 3.4, the search space of each $\hat{c}_i$ is then $\{\hat{c}_i = f \cdot 2^l \mid f \in \Upsilon(\mathbf{T}), l \in \mathbb{Z}^+, \hat{c}_i \in \mathbb{Z}_b\}$, which is referred to as $\hat{\mathbf{C}}_{\text{space}}(\mathbf{T})$ in the following derivations. Next, since the cost function of Problem 6.1 does not depend on $\hat{\mathbf{c}}$, given a $\mathbf{T}$, we need only to find a $\hat{\mathbf{c}}$ that satisfies the error constraint ($\|\mathbf{c} - \hat{\mathbf{c}}\|_1 \le \varepsilon$) if possible. For this reason, we need only consider the $\hat{\mathbf{c}}$ that minimizes $\|\mathbf{c} - \hat{\mathbf{c}}\|_1$, which we term $\varsigma(\mathbf{T})$.

Definition 6.6 ($\varsigma(\mathbf{T})$, $\varsigma_i(\mathbf{T})$)**:** Given an adder tree $\mathbf{T}$, the vector $\varsigma(\mathbf{T})$ is defined as $\varsigma(\mathbf{T}) = \operatorname{argmin}_{\hat{\mathbf{c}} \in [\hat{\mathbf{C}}_{\text{space}}(\mathbf{T})]^N} \|\mathbf{c} - \hat{\mathbf{c}}\|_1$ and can be computed component-wise as follows:

$$\varsigma(\mathbf{T}) = \left(\operatorname*{argmin}_{\hat{c}_0 \in \hat{\mathbf{C}}_{\text{space}}} |c_0 - \hat{c}_0|, \ldots, \operatorname*{argmin}_{\hat{c}_{N-1} \in \hat{\mathbf{C}}_{\text{space}}} |c_{N-1} - \hat{c}_{N-1}| \right) = (\varsigma_0(\mathbf{T}), \ldots, \varsigma_{N-1}(\mathbf{T})) \tag{6.5}$$

Further reduction of (6.5) can be performed by observing that, given a fundamental, the number of shifts l that minimizes each $|c_i - \hat{c}_i|$ can be computed as follows:

Definition 6.7 ($l_{\text{best},i}$)**:** Given a fundamental $f \in \mathbb{F}$, let

$$l_{\text{best},i}(f) = \begin{cases} \max\Big\{0, \min\Big\{1 - \Big\lfloor \log_2 \frac{3f}{c_i} \Big\rfloor, b - 1 - \lfloor \log_2 f \rfloor \Big\}\Big\} & \text{if } f \neq 0, \\ 0 & \text{if } f = 0. \end{cases} \tag{6.6}$$

Note that this shift value also ensures that $\hat{c}_i \in \mathbb{Z}_b$. As a result we have,

$$\varsigma_i(\mathbf{T}) = \underset{\{\hat{c}_i = f \cdot 2^{l_{\text{best},i}(f)} \mid f \in \Upsilon(\mathbf{T})\}}{\operatorname{argmin}} |c_i - \hat{c}_i|. \tag{6.7}$$

To summarize, for each target constant c_i, the function $\varsigma(\mathbf{T})$ scales each fundamental f_m produced by $\mathbf{T}$ by a power-of-two 2^l to get $f_m \cdot 2^l$ as close to c_i as possible. Then the overall closest scaled fundamental is chosen to be $\hat{c}_i$. Because $\varsigma(\mathbf{A})$ computes the $\hat{\mathbf{c}}$ with smallest error, there is no need to consider any other $\hat{\mathbf{c}} \in \mathbb{Z}_b$.

Combining the results from (6.3), (6.5), and (6.7), we can now rewrite the feasibility set of (6.1) as:

$$\{(\hat{\mathbf{c}}, \mathbf{T}) \mid \mathbf{T} \in \mathrm{S}_{\text{rec}}(\{\mathbf{1}\}), \hat{\mathbf{c}} \in \varsigma(\mathbf{T}), \|\mathbf{c} - \hat{\mathbf{c}}\|_1 \leq \varepsilon\}. \tag{6.8}$$

This new expression of the feasibility set is now fully defined in a functional sense, wit $\mathbf{T}$ being the only enumerated variable. Consequently, if a solution exists to Problem 6.1, then the problem can be optimally solved by noting that the cost function is additive and nonnegative. In other words, if at any point while recursing the S_{rec} function, there is no $\mathbf{T}$ found with cost($\mathbf{T}$) less than the current best solution, then the problem is solved. Such an optimal solution to Problem 6.1 can be written as:

$$(\hat{\mathbf{c}}, \mathbf{T})^{\star} = \underset{\{(\hat{\mathbf{c}}, \mathbf{T}) \mid \mathbf{T} \in \mathrm{S}_{\text{rec}}(\{\mathbf{1}\}), \hat{\mathbf{c}} \in \varsigma(\mathbf{T}), \|\mathbf{c} - \hat{\mathbf{c}}\|_1 \leq \varepsilon\}}{\operatorname{argmin}} \text{cost}(\mathbf{T}). \tag{6.9}$$

Next, Section 6.3.2 builds upon this section's results to derive a more intelligent way of enumerating the $\mathbb{T}$ space than simply exhaustively searching.

6.3.2 Applying a Heuristic

In Section 6.3.1, we formed a complete functional description of Problem 6.1, culminating in the optimal solution definition provided in (6.9). However, no algorithm for computing the solution was prescribed other than an exhaustive search. Thus, in this section, a heuristic implementation is provided for solving Problem 6.1 that, while suboptimal, is relatively efficient.

Consider a *greedy* heuristic for computing a solution in (6.9). With a greedy heuristic, enumeration of a search space is limited to having a branching factor of 1. Also, enumeration of the search space ends once the first feasible element found. Because of these two properties, a greedy heuristic might lead to a suboptimal solution, but it can often find a feasible solution much more efficiently than with an exhaustive search. Optimality of the solutions can be improved by developing a sufficiently intelligent formula for choosing the branch direction at each stage.

Consider, our problem as defined in (6.9). The search space uses the $\mathrm{S}_{\mathrm{rec}}$ function defined in (6.2). This equation shows a branch factor that can be very large, because at each level of recursing $\mathrm{S}_{\mathrm{rec}}(\mathbf{T})$, the recursion creates a new branch for each element in $\mathrm{S}(\mathbf{T})$. Figure 5.1 illustrates this exponentially growing branch factor.

Definition 6.8 ($\mathrm{S}_{\mathrm{greedy}}$)**:** On the other hand, using a greedy heuristic, (6.2) can be replaced by:

$$\mathrm{S}_{\mathrm{greedy}}(\mathbf{T}) = \begin{cases} \mathbf{T} & \text{if } \|\, \mathbf{c} - \varsigma(\mathbf{T}) \,\|_1 \leq \varepsilon, \\ \mathrm{S}_{\mathrm{greedy}}\left(\underset{\mathbf{T}' \in \mathrm{S}(\mathbf{T})}{\operatorname{argmin}} \mathrm{h}(\mathbf{T}') \right) & \text{otherwise,} \end{cases} \tag{6.10}$$

where $\mathrm{h}(\mathbf{T}')$ is some evaluator function that is ideally minimized for the *best* choice of successor tree $\mathbf{T}'$.

As a simple, yet effective, heuristic evaluator, we consider the following definition of h:

Definition 6.9 (h)**:** Given an adder tree $\mathbf{T}$, let $\mathrm{h}(\mathbf{T}) = \|\mathbf{c} - \varsigma(\mathbf{T})\|_1$.

Using this heuristic evaluator, the successor adder tree with the smallest error is always chosen. In this manner, the algorithm greedily moves towards a feasible pair $(\hat{\mathbf{c}}, \mathbf{T})$ by reducing the quantization error as much as possible at each stage. This heuristic evaluator does not consider the actual cost of $\mathbf{T}$ at any stage but can effectively lead to reduced solution costs since the total number of $\mathcal{A}$-relations is lessened.

Finally, the greedy solution to Problem 6.1 is computed as:

$$(\hat{\mathbf{c}}, \mathbf{T})_{\mathrm{greedy}} = (\varsigma(\mathrm{S}_{\mathrm{greedy}}(\mathbf{1})), \mathrm{S}_{\mathrm{greedy}}(\mathbf{1})). \tag{6.11}$$

As a result of fully specifying the functional description of Problem 6.1 in Sections 6.3.1 and 6.3.2, the greedy algorithm can be implemented directly from the function definitions into functional programming code. Accordingly, Algorithm 6.1 provides an F# [52, 53] implementation. The code is generated from translating the function definitions into proper F# syntax, and no other algorithmic considerations are needed to convert from the mathematical description into the program code. For this reason, the functional programming implementation is clear and concise and is less prone to errors than an imperative programming implementation.

6.4 Experiments and Results

In Section 6.2, an MRCM problem, Problem 6.1, is formulated to address the optimization of an MCM circuit using a gate-level cost model, and in Section 6.3, a greedy algorithm GREEDY_SOLVE is developed for solving this problem. Therefore, in this section, an experiment is conducted to evaluate the MRCM approach using

Algorithm 6.1 SOLVE_GREEDY: F# implementation

```
let solve_greedy c ep (b:int) =
    let rootrel = (1.0, 1.0, (0, 0, 0), 1.0)
    let snap T =
        let invaddertreesof = [for (_,_,_,w) in T do yield w ; yield 0.0]
        let snap_i c_i =
            let l_besti f =
                if f = 0.0 then
                    0
                else
                    max 0.0 (min (1.0 - floor (log2 (3.0 * f / c_i))) (float b - 1.0 - floor (log2 f)))
                        |> int
            [for f in invaddertreesof -> f * twoTo (l_besti f)] |>
                List.minBy (fun cHat_i -> abs (c_i - cHat_i))
        List.map snap_i c
    let rec S_greedy T =
        let S =
            let S_al =
                let Al (l_1,l_2,s) u v =
                    let rec fundamentalize f =
                        if f % 1.0 = 1.0 then
                            f
                        else
                            fundamentalize (f / 2.0)
                    fundamentalize (twoTo l_1 * u + (-1.0)**(float s) * (twoTo l_2) * v)
                [
                    for (_,_,_,u) in T do
                    for (_,_,_,v) in T do
                        for l_1 in (range 1.0 (log2 ((twoTo b - 1.0 - v) / u) )) ->
                            (u, v, (l_1, 0, 0), Al (l_1, 0, 0) u v)
                        yield (u, v, (0, 0, 0), Al (0, 0, 0) u v)
                        for l_1 in (range (max 1.0 (log2 ((3.0 + v) / u)))
                                          (log2 ((twoTo b + v - 1.0) / u))) ->
                            (u, v, (l_1, 0, 1), Al (l_1, 0, 1) u v)
                        for l_2 in (range 1.0 (log2 ((u - 3.0) / v))) ->
                            (u, v, (0, l_2, 1), Al (0, l_2, 1) u v)
                        if u >= v + 6.0 then yield (u, v, (0, 0, 1), Al (0, 0, 1) u v)
                ]
            [for r_prime in S_al -> r_prime :: T]
        let h T = List.map2 (fun c_i cHat_i -> abs (c_i - cHat_i)) c (snap T) |> List.sum
        if h T <= ep then
            T
        else
            S_greedy (List.minBy h S)
    (snap (S_greedy [rootrel]), S_greedy [rootrel])
```

GREEDY_SOLVE in comparison to the de facto disjointed approach of rounding the real constants and applying an MICM algorithm for optimization. Section 6.4.1 details the experimental setup and Section 6.4.2 provide the results of the experiment.

6.4.1 Experimental Setup

The experiment is divided into three sets of randomized trials designed to show different trends of the algorithm results as the parameters N, ε, and b vary. For the first experiment set, the number of bits is held constant at $b = 8$, while the number of constants N and the error bound ε vary. The parameter N takes the values 20, 40, 60, and 80. Each constant c_i of $\mathbf{c}$ is drawn from a uniform distribution on $\mathbb{R}_b$. For each N, ε takes on four values evenly spaced between $N/4$ and $N/2$. The reasoning is as follows: For a given bitwidth b and uniformly distributed constants in $\mathbb{R}_b$, the expected minimum value of ε such that a solution to Problem 6.1 exists is $\mathrm{E}[\|\mathbf{c} - \mathrm{round}(\mathbf{c})\|_1] = N/4$. This property also implies that when ε exceeds $N/2$ it is expected that a solution exists using only $b-1$ bits. Thus, for a fixed b, exercising the range $N/4$ to $N/2$ is appropriate for determining the effects that increased error has on algorithm performance. Because the range of ε depends on N, it is useful to define a quantity $\tilde{\varepsilon} = \varepsilon/N$ to be used when showing trends.

Similarly, for the second experiment set, the number of constants is fixed at $N = 60$. Then, the number of bits b takes the values 5–11 and $\tilde{\varepsilon}$ takes the same values as the first experiment set—four values evenly spaced between $N/4$ and $N/2$.

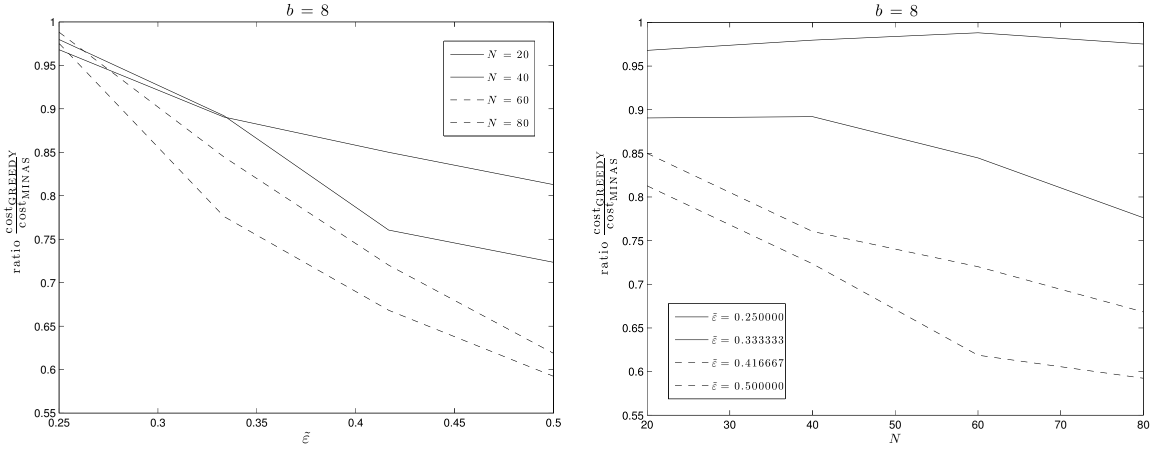
Last, for the third experiment set, the quantity $\tilde{\varepsilon}$ is fixed towards the middle of the range at $\tilde{\varepsilon} = 5/12$. Again, N takes the values 20, 40, 60, and 80 and b takes the values 5–11. Additionally, for averaging purposes, there are ten randomized trials run for each parameter set.

For each trial, the MRCM approach using GREEDY_SOLVE is compared with the de facto disjointed approach of rounding the real constants and applying an MICM algorithm for optimization. For the disjointed approach, the MICM algorithm MINAS developed by Aksoy et al. in [26] is considered, since it is a recent heuristic designed specifically for the gate-level cost model and is relatively efficient. To enable the use of an MICM algorithm, the random constants are first rounded to b bits. Then, both algorithms are executed on their respective real/integer constants. These executions generate an adder tree for each algorithm. Last, an integer linear programming (ILP) problem is formed for each adder tree and is solved using an ILP solver called SCIP [54,55]. This last step rearranges the fundamentals in the adder trees to optimize the gate-level cost with respect to the limitations discussed in [26], that is, the order of the output fundamentals in the adder tree cannot change[3].

6.4.2 Results

The results of these three experiment sets are now discussed. To compare the relative costs of the two algorithms GREEDY_SOLVE and MINAS in an empirical manner, we look at the ratio $\frac{\text{cost}_{\text{GREEDY}}}{\text{cost}_{\text{MINAS}}}$ for each parameter set, averaged over the ten trials.

Figure 6.1 shows the results of the first experiment set. The figure gives two different views of the same data to show trends over both of the variable trial parameters $\tilde{\varepsilon}$ and N. Observe from Figure 6.1(a) that the ratio has a negative trend as the error bound increases, starting around 1 for $\varepsilon = 0.25$ and decreasing monotonically as ε approaches 0.5. This means that the MRCM algorithm achieves better results when there is more flexibility in terms of error, which is to be expected. Furthermore, the plots show that as the number of constants increases, the overall improvement is better as well. This observation makes sense because, with a higher number of constants, there is more flexibility in how the error can be distributed between the constants.

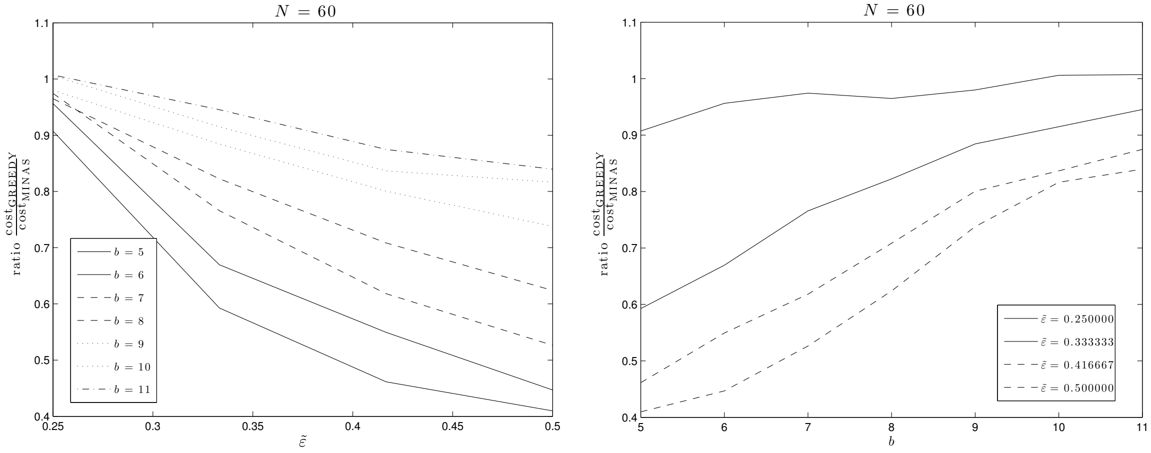


(a) ratio vs. $\tilde{\varepsilon}$

(b) ratio vs. N

Figure 6.1: Results of the first experiment set in terms of the ratio $\frac{\text{cost}_{\text{GREEDY}}}{\text{cost}_{\text{MINAS}}}$.

[3] However, output fundamentals may be removed from the adder tree if they are not needed



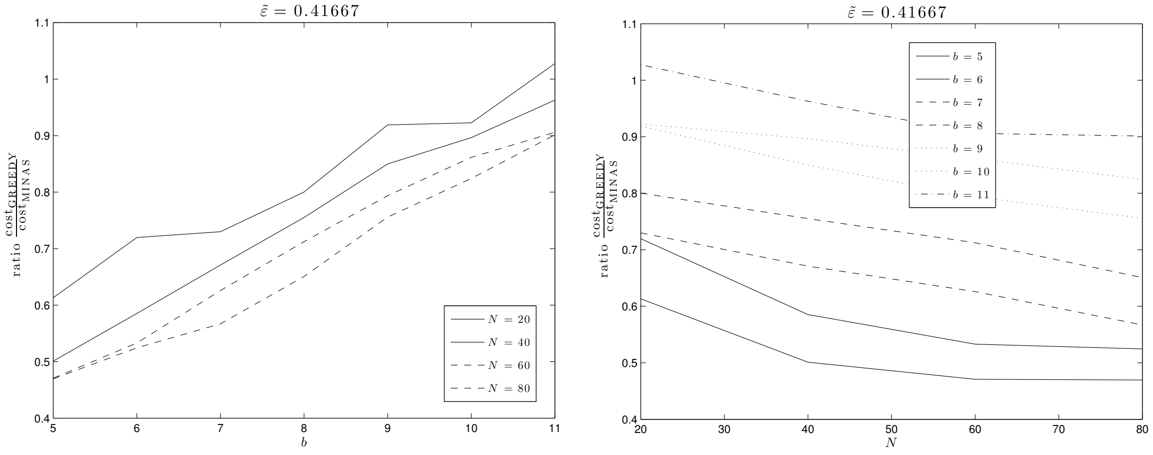
(a) ratio vs. $\tilde{\varepsilon}$

(b) ratio vs. b

Figure 6.2: Results of the second experiment set in terms of the ratio $\dfrac{\text{cost}_{\text{GREEDY}}}{\text{cost}_{\text{MINAS}}}$.

Next, Figure 6.2 shows the results of the second experiment set. This figure shows a very similar trend in terms of improvement versus the error bound. However, the figure also shows that, as the number of bits increases, the potential for improvement goes down (the cost ratio goes up). Where the ratio reaches a low of approximately 0.41 for $b = 5$, it only reaches a low of approximately 0.84 for $b = 11$. We believe that this result comes from the fact that the greedy heuristic $\mathrm{h}(\mathbf{T})$ defined in Section 6.3.2 does not take bitwidth into account, thus leading to possibly higher gate-level costs since the gate-level cost of individual $\mathcal{A}$-relations is heavily dependent on the bitwidths of the $\mathcal{A}$-relations inputs and output. Since the MINAS algorithm partially takes bitwidths of fundamentals into account, it is possible that the advantages that stem from flexibility in error of the greedy algorithm might be outweighed by the disadvantages that arise from ignoring fundamental bitwidths.

Last, Figure 6.3 shows the results of the third experiment set. This figure's plots agree with the first two experiment sets with respect to the trends associated with N and b.



(a) ratio vs. b

(b) ratio vs. N

Figure 6.3: Results of the third experiment set in terms of the ratio $\frac{\text{cost}_{\text{GREEDY}}}{\text{cost}_{\text{MINAS}}}$.

6.5 Summary

In summary, this chapter's results highlight the advantages of using an MRCM approach as opposed to a rounding+MICM approach to solving MCM problems using a gate-level cost model. In particular, a novel greedy algorithm was developed that uses a functional programming approach to implementing an MRCM solver. As the authors hypothesized, this algorithm that uses the MRCM framework exhibits cost improvement over the MICM algorithm, because it leverages the inherent flexibility allowed in the quantization of constants. As expected, greater error bounds lead to greater potential improvement using the MRCM framework. Specifically, the amount of potential improvement is dependent on the magnitude of the error bound relative to the number and magnitude of the constants. The greedy algorithm developed in this chapter was especially effective for bitwidths in the range 5–11, for which it was experimentally shown to offer an improvement of up to 18% on small instances ($N = 20, b = 8$). This improvement increased as the number of constants grew, reaching up to 59% improvement on larger instances ($N = 80, b = 5$).

Furthermore, we would like to thank Aksoy et al. for providing us with their implementation of the MINAS algorithm [26].

Chapter 7

Conclusion

7.1 Summary of Findings

In this dissertation, we provided an alternative framework for looking at the MCM problem. First, in Chapter 2, we reviewed the current literature with respect to the MCM problem, and noted a major shortcoming that leads to a potential loss of optimality in applications involving real, non-integer constants. Then, in Chapter 3, we provided the general MRCM problem formulation for addressing this shortcoming. In the next three chapters, we looked at the MRCM problem using three different cost models and compared our joint MRCM framework to the de facto disjoint MICM framework.

Specifically, in Chapter 4, we developed an algorithm for solving the MRCM problem with adder-count cost model and, through a randomized experiment, we showed that that our joint framework leads to a reduction on the number of adders by 15%–60% on moderate size problems. In particular, for vectors of arbitrary constants, we show a possibility for 20%–60% reduction with less than 10% vector approximation error for both frameworks, whereas for vectors of low-pass filter coefficients, a 15%–30% reduction is possible without exceeding 10% error in frequency response. Next, in Chapter 5, we developed two algorithms for solving the MRCM problem with bit-count cost model—a greedy algorithm and an optimal algorithm. Again, using a randomized experiment, we observed that the greedy search finds solutions with an average cost improvement of 13% over MICM algorithm solutions, and the optimal search finds an additional improvement of 6%. Last, in Chapter 6, we developed a functional programming implementation of an algorithm to solve the MRCM problem with gate-level cost model. In a round of randomized experiments, we showed this algorithm to offer an improvement of up to 18%, over a competing MICM algorithm, on small instances having 20 8-bit constants, increasing to up to 59% improvement on larger instances having 80 5-bit constants.

Regardless of the cost model used, the general MRCM problem framework was found to enable more optimal solutions than the MICM framework, because it jointly optimizes the quantization and shift-add network of a given MCM design.

7.2 Recommended Future Work

I recommend several extensions and derivations of this work to be investigated in future MCM research. First, along with the improvements that the algorithms of this dissertation produce, it can be seen that there is potential for further improvements by investigating alternative algorithms and heuristics for solving the various MRCM problems defined in this dissertation. For instance, when using the gate-level cost model, a better heuristic could be developed that takes into account bitwidths of intermediate fundamentals, thus allowing for more competitive results under higher bitwidths. Also, with the gate-level cost model, the basic algorithm can be factored somewhat to allow for more efficient computation.

In addition to algorithm improvements, I recommend considering different hardware cost models that would translate to different optimization problems for solving. One modification to the current MCM problem for linear time-invariant (LTI) filters might be to consider allowing a common factoring of the target constants to add another degree of freedom in the search space. Such a factor would merely cause a gain adjustment without altering the frequency response characteristics. Another drastically different hardware model could be investigated that considers a much finer grain in terms of its primitive operations, in order to abstract away from the high-level adders/multipliers and consider the underlying components that they are made up of.

Bibliography

[1] S. K. Mitra, *Digital Signal Processing: A Computer-Based Approach*, 4th ed. McGraw-Hill Higher Education, 2010.

[2] D. Shi and Y. J. Yu, "Low-complexity linear phase FIR filters in cascade form," in *Proc. IEEE Int. Symp. Circuits Syst.*, Jun. 2010, pp. 177–180.

[3] G. F. Franklin, J. D. Powell, and M. L. Workman, *Digital Control of Dynamic Systems*, 3rd ed. Addison-Wesley, 1997.

[4] A. Goldsmith, *Wireless Communications.* Cambridge University Press, 2005.

[5] H. Chen, "Transformation optics in orthogonal coordinates," *J. of Optics A: Pure and Appl. Optics*, vol. 11, no. 7, p. 075102, Jul. 2009.

[6] K. Wang, J. Chen, W. Cao, Y. Wang, L. Wang, and J. Tong, "A reconfigurable multi-transform VLSI architecture supporting video codec design," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 58, no. 7, pp. 432–436, Jul. 2011.

[7] C. D. Bailey and D. D. Aalfs, "Design of digital beamforming subarrays for a multifunction radar," in *Proc. IEEE Int. Radar Conf.–Surveilance for a Safer World*, Oct. 2009.

[8] A. T. Fam, "Digital beamforming with reduced number of phase shifting and time delay elements," in *Proc. IEEE Radar Conf.*, May 2010, pp. 1286–1288.

[9] F. Xikun and W. Yongliang, "Real-time implementation of airborne radar space-time adaptive processing on multi-DSP system," in *Proc. CIE Int. Conf. Radar*, Oct. 2006.

[10] M. Potkonjak, M. B. Srivastava, and A. P. Chandrakasan, "Multiple constant multiplications: Efficient and versatile framework and algorithms for exploring common subexpression elimination," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 15, no. 2, pp. 151–165, Feb. 1996.

[11] J. Q. Trelewicz and T. J. Trenary, "Multidimensional rational approximations with an application to linear transforms," *IEEE Trans. Signal Process.*, vol. 52, no. 5, pp. 1343–1351, May 2004.

[12] M. B. Yeary, W. Zhang, J. Q. Trelewicz, Y. Zhai, and B. McGuire, "Theory and implementation of a computationally efficient decimation filter for power-aware embedded systems," *IEEE Trans. Instrum. Meas.*, vol. 55, no. 5, pp. 1839–1849, Oct. 2006.

[13] Y. Voronenko and M. Püschel, "Multiplierless multiple constant multiplication," *ACM Trans. Algorithms*, vol. 3, no. 2, article 11, May 2007.

[14] M. Potkonjak, M. B. Srivastava, and A. P. Chandrakasan, "Efficient substitution of multiple constant multiplications by shifts and additions using iterative pairwise matching," in *Proc. IEEE/ACM Design Automation Conf.*, Jun. 1994, pp. 189–194.

[15] A. G. Dempster and M. D. Macleod, "Use of minimum-adder multiplier blocks in FIR digital filters," *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol. 42, no. 9, pp. 569 –577, Sep. 1995.

[16] A. Dempster, S. Demirsoy, and I. Kale, "Designing multiplier blocks with low logic depth," in *Proc. IEEE Int. Symp. Circuits Syst.*, vol. 5, Oct. 2002, pp. 773–776.

[17] O. Gustafsson, "A difference based adder graph heuristic for multiple constant multiplication problems," in *Proc. IEEE Int. Symp. Circuits Syst.*, May 2007, pp. 1097–1100.

[18] ——, "Lower bounds for constant multiplication problems," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 54, no. 11, pp. 974–978, Nov. 2007.

[19] L. Aksoy, E. da Costa, P. Flores, and J. Monteiro, "Exact and approximate algorithms for the optimization of area and delay in multiple constant multiplications," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 27, no. 6, pp. 1013–1026, Jun. 2008.

[20] Y.-H. A. Ho, C.-U. Lei, H.-K. Kwan, and N. Wong, "Global optimization of common subexpressions for multiplierless synthesis of multiple constant multiplications," in *Proc. IEEE Design Automation Conf., Asia South Pacific*, Mar. 2008, pp. 119–124.

[21] L. Aksoy, E. O. Gúneş, and P. Flores, "Search algorithms for the multiple constant multiplications problem: Exact and approximate," *Elsevier Microprocessors and Microsystems*, vol. 34, no. 5, pp. 151 – 162, Aug. 2010.

[22] L. Aksoy, E. da Costa, P. Flores, and J. Monteiro, "Optimization algorithms for the multiplierless realization of linear transforms," *ACM Trans. Design Automation Electron. Syst.*, vol. 17, no. 1, pp. 3:1–3:27, Jan. 2012.

[23] K. Johansson, L. S. DeBrunner, O. Gustafsson, and V. DeBrunner, "Design of multiplierless FIR filters with an adder depth versus filter order trade-off," in *Conf. Rec. 43rd Asilomar Conf. Signals Syst. and Comput.*, Nov. 2009, pp. 744–748.

[24] K. Johansson, O. Gustafsson, and L. Wanhammar, "Bit-level optimization of shift-and-add based FIR filters," in *Proc. IEEE Int. Conf. Electron. Circuits Syst.*, Dec. 2007, pp. 713–716.

[25] L. Aksoy, E. da Costa, P. Flores, and J. Monteiro, “Optimization of area and delay at gate-level in multiple constant multiplications,” in *Euromicro Conf. Digital Syst. Design*, Sep. 2010, pp. 3–10.

[26] ——, “Finding the optimal tradeoff between area and delay in multiple constant multiplications,” *Elsevier Microprocessors and Microsystems*, vol. 35, no. 8, pp. 729–741, 2011.

[27] K. Johansson, O. Gustafsson, and L. Wanhammar, “A detailed complexity model for multiple constant multiplication and an algorithm to minimize the complexity,” in *Proc. IEEE European Conf. Circuit Theory Design*, vol. 3, Aug. 2005, pp. 465–468.

[28] T. Parks and J. McClellan, “Chebyshev approximation for nonrecursive digital filters with linear phase,” *IEEE Trans. Circuit Theory*, vol. 19, no. 2, pp. 189–194, Mar. 1972.

[29] J. McClellan, T. Parks, and L. Rabiner, “A computer program for designing optimum FIR linear phase digital filters,” *IEEE Trans. Audio Electroacoust.*, vol. 21, no. 6, pp. 506–526, Dec. 1973.

[30] Y. C. Lim and S. R. Parker, “Discrete coefficient FIR digital filter design based upon an LMS criteria,” *IEEE Trans. Circuits Syst.*, vol. 30, no. 10, pp. 723–739, Oct. 1983.

[31] G. Evangelista, “Least mean squared error-design of complex FIR filters with quantized coefficients,” *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol. 48, no. 8, pp. 778–784, Aug. 2001.

[32] M. Magar and L. S. DeBrunner, “Balancing the tradeoffs between coefficient quantization and internal quantization in FIR digital filters,” in *Conf. Rec. 38th Asilomar Conf. Signals Syst. and Comput.*, vol. 1, Nov. 2004, pp. 493–497.

[33] R. Ito, T. Fujie, K. Suyama, and R. Hirabayashi, “A new heuristic approach for design of FIR filters with SP2 coefficients in a min-max sense,” in *Proc. IEEE Int. Conf. Signal Processing*, vol. 1, Aug. 2004, pp. 81–84.

[34] J. Yli-Kaakinen and T. Saramäki, “A systematic algorithm for the design of multiplierless FIR filters,” in *Proc. IEEE Int. Symp. Circuits Syst.*, vol. 2, May 2001, pp. 185–188.

[35] H.-J. Kang and I.-C. Park, “FIR filter synthesis algorithms for minimizing the delay and the number of adders,” *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol. 48, no. 8, pp. 770–777, Aug. 2001.

[36] D. L. Maskell, “Design of efficient multiplierless FIR filters,” *IET Circuits Devices Syst.*, vol. 1, no. 2, pp. 175–180, Apr. 2007.

[37] F. Xu, C. H. Chang, and C. C. Jong, "Design of low-complexity FIR filters based on signed-powers-of-two coefficients with reusable common subexpressions," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 26, no. 10, pp. 1898–1907, Oct. 2007.

[38] Y. J. Yu and Y. C. Lim, "Design of linear phase FIR filters in subexpression space using mixed integer linear programming," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 54, no. 10, pp. 2330–2338, Oct. 2007.

[39] M. Aktan, A. Yurdakul, and G. Dündar, "An algorithm for the design of low-power hardware-efficient FIR filters," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 55, no. 6, pp. 1536–1545, Jul. 2008.

[40] D. Shi and Y. J. Yu, "Design of linear phase FIR filters with high probability of achieving minimum number of adders," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 58, no. 1, pp. 126–136, Jan. 2011.

[41] M. B. Gately, M. B. Yeary, and C. Y. Tang, "Reduced-hardware digital filter design via joint quantization and multiple constant multiplication optimization," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process.*, May 2011, pp. 4368–4371.

[42] ——, "An algorithm for jointly optimizing quantization and multiple constant multiplication," *ACM Trans. Design Automation Electron. Syst.*, accepted for publication, 2012.

[43] S. K. Mitra, *Digital Signal Processing: A Computer-Based Approach*, 3rd ed. McGraw-Hill Higher Education, 2005.

[44] J. L. Gross and J. Yellen, Eds., *Handbook of Graph Theory*, ser. Discrete mathematics and its applications. CRC Press, 2004.

[45] Xilinx design tools. Xilinx. Accessed Mar. 9, 2011. [Online]. Available: http://www.xilinx.com/tools/designtools.htm

[46] Y. Voronenko. (2009) SPIRAL multiplier block generator. SPIRAL project, Carnegie Mellon University. Http://spiral.ece.cmu.edu/mcm/gen.html. Accessed Oct. 5, 2010. [Online]. Available: http://spiral.ece.cmu.edu/mcm/gen.html

[47] P. Cappello and K. Steiglitz, "Some complexity issues in digital signal processing," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 32, no. 5, pp. 1037–1041, Oct. 1984.

[48] Intel. (2011) Intel® Xeon® processor E5405 (12M cache, 2.00 GHz, 1333 MHz FSB)with SPEC code(s) SLAP2, SLBBP. Specifications. Intel Corporation. Http://ark.intel.com/Product.aspx?id=33079. Accessed Mar. 8, 2011. [Online]. Available: http://ark.intel.com/Product.aspx?id=33079

[49] MathWorks. (2011) Parks-McClellan optimal FIR filter design. R2011b Documentation: Signal Processing Toolbox. The MathWorks, Inc. Http://www.mathworks.com/help/toolbox/signal/ref/firpm.html. Accessed Nov. 8, 2011. [Online]. Available: http://www.mathworks.com/help/toolbox/signal/ref/firpm.html

[50] M. B. Gately, M. B. Yeary, and C. Y. Tang, "Multiple real-constant multiplication with improved cost model and greedy and optimal searches," in *Proc. IEEE Int. Symp. Circuits Syst.*, accepted for publication, May 2012.

[51] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Trans. Syst. Sci. Cybern.*, vol. 4, no. 2, pp. 100–107, Jul. 1968.

[52] D. Syme, A. Granicz, and A. Cisternino, *Expert F# 2.0.* Berkeley, CA, USA: Apress, 2010.

[53] Microsoft Research. (2012, Apr.) SPIRAL multiplier block generator. [Online]. Available: http://research.microsoft.com/en-us/um/cambridge/projects/fsharp/

[54] T. Berthold, S. Heinz, and M. E. Pfetsch, "Solving pseudo-boolean problems with SCIP," Zuse Institute Berlin (ZIB), ZIB-Report 08-12, Jul. 2009. [Online]. Available: http://opus.kobv.de/zib/volltexte/2008/1095/pdf/ZR_08_12.pdf

[55] Zuse Institute Berlin (ZIB). (2012, Apr.) SCIP: An open source MIP solver and constraint integer programming framework. [Online]. Available: http://scip.zib.de