

# Multiplierless Multiple Constant Multiplication

YEVGEN VORONENKO and MARKUS PÜSCHEL

Carnegie Mellon University

---

A variable can be multiplied by a given set of fixed-point constants using a multiplier block that consists exclusively of additions, subtractions, and shifts. The generation of a multiplier block from the set of constants is known as the multiple constant multiplication (MCM) problem. Finding the optimal solution, i.e., the one with the fewest number of additions and subtractions is known to be NP-complete. We propose a new heuristic algorithm for the MCM problem, which finds solutions that require up to 20% less additions and subtractions than the solutions found by the best previously known algorithm. At the same time, our algorithm is not limited by the constant bitwidths, in contrast to the closest competing algorithm. We present our algorithm using a unifying formal framework for the best, graph-based MCM algorithms and provide a detailed runtime analysis and experimental evaluation. We show that our algorithm can handle problem sizes as large as 100 32-bit constants in a time acceptable for most applications. The implementation of the new algorithm is available at [www.spiral.net](http://www.spiral.net).

Categories and Subject Descriptors: B.2.4 [**High-speed Arithmetic**]: Algorithms—*Algorithms*; Cost/Performance; F.2.1 [**Analysis Of Algorithms And Problem Complexity**]: Numerical Algorithms and Problems—*Number-theoretic computations*; I.1.2 [**Symbolic And Algebraic Manipulation**]: Algorithms—*Algebraic Algorithms*

General Terms: Algorithms, Design, Performance, Theory

Additional Key Words and Phrases: addition chains, directed graph, fixed-point arithmetic, strength reduction, FIR filter

---

## 1. INTRODUCTION

This paper addresses the problem of efficiently computing a set of products  $t_i x$ , for  $i = 1, \dots, n$ , of a variable  $x$  with several known fixed-point constants  $t_i$  *multiplierless*, that is using exclusively additions, subtractions, and shifts. This problem is known as multiple constant multiplication (MCM). Avoiding costly multipliers is particularly important in hardware implementations, for example, of digital signal processing functionality such as filters or transforms. However, replacing constant multiplications with additions and shifts can also be relevant in software implementations. For example, as optimization for speed, since integer multipliers often have a significantly lower throughput than adders, but also for embedded processors, which may not feature a multiplication unit at all. The MCM problem can be considered as a fundamental problem in computer arithmetic.

We propose a new algorithm for the MCM problem. Our algorithm generates solutions that are significantly better—in terms of the number of additions and subtractions of the solution—than the solutions found by any of the previously published algorithms. Further, the new algorithm is more generally applicable. To clearly state our contribution and to put it into the context of previous work, we first introduce the problem in greater detail.

### 1.1 Single Constant Multiplication (SCM)

The multiplication  $y = tx$  of a variable  $x$  by a known integer or fixed-point constant  $t$  can be decomposed into additions (adds), subtractions (subtracts), and binary shifts. The problem of finding the decomposition with the least number of operations is known as *single constant multiplication (SCM)* problem, and it is NP-complete as shown in [Cappello and Steiglitz 1984]. Without loss of generality, we assume that the constants are integers, because a fixed-point multiplication is equivalent to a multiplication by an integer followed by a right shift. The SCM problem is related to, but different from the addition chain problem [Knuth 1969], which multiplies by a constant using additions only. The permission of shifts fundamentally alters the problem and the strategies for its solution.

The straightforward method for decomposing the multiplication into adds and shifts translates 1's in the binary representation of the constant  $t$  into shifts, and adds up the shifted inputs. For example, for  $t = 71$ ,

$$71x = 1000111_2x = x \ll 6 + x \ll 2 + x \ll 1 + x,$$

which requires 3 adds. Alternatively, the multiplication can be decomposed into subtracts and shifts by translating 0's into shifts, and subtracting from the closest constant consisting of 1's only (i.e., of the form  $2^n - 1$ ):

$$71x = 1000111_2x = (x \ll 7 - x) - x \ll 5 - x \ll 4 - x \ll 3,$$

Taking the best of these two methods yields in the worst and in the average case a solution with  $\frac{b}{2} + O(1)$  adds/subtracts, where  $b$  is the bitwidth of  $t$ .

A better digit-based method decomposes into both adds and subtracts by recoding the number into the canonical signed digit (CSD) representation [Avizienis 1961], which allows negative digits  $\bar{1}$ . Using CSD, the previous example can be improved to use only 2 add/subtract operations:

$$1000111_2x = 100100\bar{1}_{\text{CSD}}x = x \ll 6 + x \ll 3 - x$$

Using CSD, the worst case cost remains  $\frac{b}{2} + O(1)$ , but the average case is now improved to  $\frac{b}{3} + O(1)$  [Wu and Hasan 1999].

The optimal decomposition in terms of add/subtract operations is in general not obtained with CSD, and its worst case and average costs are unknown. [Dempster and Macleod 1994] designed an exhaustive search algorithm to find the optimal decompositions for constants up to 12 bits. The authors also showed that using shifts no larger than  $b + 1$  is sufficient to yield the optimal solutions for 12 bit constants. Their work has been extended by [Gustafsson et al. 2002] to constants up to 19 bits, again yielding optimal results regardless of shift constraints. Although the asymptotic worst case cost of the optimal decomposition remains an open research problem, it appears to be asymptotically better than  $O(b)$  as shown in Fig. 1. The plot compares the three decomposition methods by showing the average number of adds/subtracts (y-axis) obtained for 300 uniformly distributed random constants of bitwidths from 2 to 19 (x-axis).

Consider the smallest constant for which the CSD decomposition is suboptimal, namely 45. Fig. 2 shows its CSD (3 add/subtract operations) and its optimal (2 add/subtract operations) decomposition both visualized as graphs. We observe

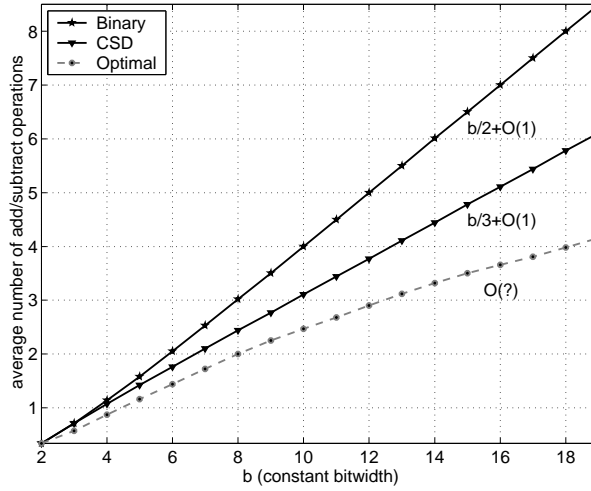


Fig. 1. Average number of add/subtract operations required for multiplying by a constant of different bitwidths.

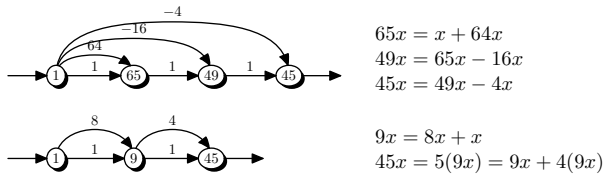


Fig. 2. Multiplication by 45 using 3 adds/subtracts (CSD, top) and 2 adds/subtracts (optimal, bottom). The vertices represent add/subtract operations labeled with their outputs, and the edges represent shifts labeled with the corresponding scaling (a 2-power). A negative scaling indicates that a subtraction is performed.

that the optimal decomposition uses a different graph topology than the CSD decomposition. Intuitively, digit-based methods, such as CSD, produce suboptimal results, because they only consider one type of graph topology. The exhaustive search methods in [Dempster and Macleod 1994; Gustafsson et al. 2002] on the other hand consider all possible graph topologies to find optimal decompositions.

### 1.2 Multiple Constant Multiplication (MCM)

An extension of SCM is the problem of multiplying a variable  $x$  by several constants  $t_1, \dots, t_n$  in parallel in a so-called *multiplier block* shown in Fig. 3. Since intermediate results of the constant decompositions may be shared, a multiple constant multiplier block may be decomposed into fewer operations than the sum of the single constant decompositions' operation counts. The problem of finding the decomposition with the fewest operations is known as *Multiple Constant Multiplication (MCM)*.

The potential savings from sharing intermediate results increase with the number

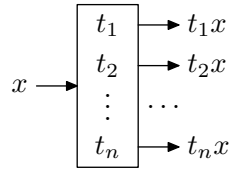


Fig. 3. (Multiple constant) Multiplier block.

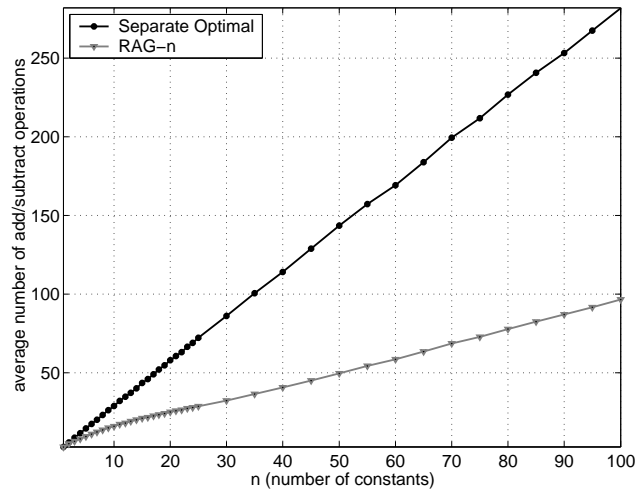


Fig. 4. Average number of add/subtract operations required for multiplying by each of the 12-bit coefficients in a set of given size.

of constants as illustrated in Fig. 4. The plot compares the number of add/subtract operations (y-axis) for varying sizes  $n$  (x-axis) of sets of 12-bit constants using separate optimal SCM decompositions and using RAG- $n$ , the heuristic MCM algorithm from [Dempster and Macleod 1995]. Since MCM is a generalization of SCM, it is also NP-complete.

The MCM problem is particularly relevant for the multiplierless implementation of digital finite impulse response (FIR) filters [Bull and Horrocks 1991], but also for matrix-vector products with a fixed matrix, which includes linear signal transforms [Püschel et al. 2004; Chen et al. 2002; Liang and Tran 2001], such as the discrete Fourier transform or the discrete cosine transform. In an  $n$ -tap FIR filter, every input sample is multiplied by all  $n$  taps. Discrete Fourier and trigonometric transform algorithms, on the other hand, involve  $2 \times 2$  rotations, which require the simultaneous multiplication by two constants.

Fig. 5 is an example of a multiplier block, which implements the parallel multiplication by 23 and 81 using only 3 add/subtract operations, although the separate optimal decompositions of 23 and 81 each require 2 add/subtract operations.

The different problem of *multiplexed* multiple constant multiplication was consid-

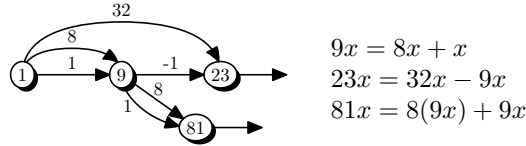


Fig. 5. Multiplier block with constants 23 and 81.

ered in [Tummeltshammer et al. 2004]. In this case, the multiplier block contains multiplexers that are switched by control logic to achieve multiplication by different constants. This way sequential multipliers can be fused. We will not consider this problem in this paper.

**Existing algorithms.** The existing MCM algorithms can be divided into four general classes:

- Digit-based recoding;
- Common subexpression elimination (CSE) algorithms;
- Graph-based algorithms;
- Hybrid algorithms.

*Digit-based recoding* includes simple methods like CSD and the binary method mentioned in Section 1.1. They generate the decomposition directly from the digit representation of the constant. These methods are the fastest and the worst-performing; however, a more recent approach [Coleman 2001] uses different number systems to yield considerably better solutions. The main advantage of digit-based recoding is their low computational cost, typically linear in the number of bits. As a consequence, these methods can be easily applied to constants with thousands of bits.

*Common subexpression elimination (CSE) algorithms* are direct descendants of digit-based recoding methods. The basic idea is to find common subpatterns in the representations of the constants after the constants are converted to a convenient number system such as CSD. Examples for this method include [Pasko et al. 1999; Lefèvre 2001; Hartley 1996]. The disadvantage, however, is that the performance of these algorithms depends on the number representation. Further, even though the considered CSE problem is NP-complete [Garey and Johnson 1979; Downey et al. 1980], its optimal solution does in general not provide the optimal MCM solution. More recently, [Dempster and Macleod 2004] proposes searching over alternative number representations to find considerably improved solutions using a CSE algorithm.

*Graph-based algorithms* are bottom-up methods that iteratively construct the graph (as in Fig. 5) representing the multiplier block. The graph construction is guided by a heuristic that determines the next graph vertex to add to the graph. Graph-based algorithms offer more degrees of freedom by not being restricted to a particular representation of the coefficients, or a predefined graph topology (as in digit-based algorithms), and typically produce solutions with the lowest number of operations. Examples of graph-based algorithms include [Bull and Horrocks 1991], RAG-n [Dempster and Macleod 1995], and [Bernstein 1986]. This paper proposes a new graph-based algorithm.

*Hybrid algorithms* combine different algorithms, possibly from different classes. For example, [Choo et al. 2004] constructs the multiplier block graph with fixed topology to compute the so-called *differential coefficients*, and then switches to a CSE algorithm for the multiplication by the differential coefficients.

Today, to the best of our knowledge, RAG-n yields the solutions with the smallest number of add/subtract operations among all algorithms. Graph-based algorithms are expected to outperform other methods since they have fewest restrictions. However, RAG-n relies on a lookup table of optimal single constant decompositions, which is currently limited to 19 bits as mentioned before.

**Contribution of this paper.** This paper first presents a general formal framework that captures the common structure of graph-based MCM algorithms. A crucial component in this framework is our notion of “ $\mathcal{A}$ -distance,” an extension of the concept of adder distance introduced in [Dempster and Macleod 1995], and its exact or heuristic estimation. We use the framework to develop a new graph-based MCM algorithm that outperforms the best available algorithms with respect to the number of add/subtract operations in the obtained multiplier blocks. In particular, we achieve an up to 20% lower average operation count than the best previous algorithm RAG-n. At the same time, our new algorithm is not bitwidth limited like RAG-n, and can thus be used to generate multiplier blocks for all practically relevant bitwidths. Finally, we perform a detailed runtime analysis of our new algorithm and other graph-based algorithms used for benchmarks. This analysis was not provided in the original papers.

**Other optimization metrics.** Besides reducing the number of add/subtract operations, it is often desirable to optimize for other metrics, for example, the critical path of the MCM block, or the register pressure in the generated code. Examples of such work include [Dempster et al. 2002] and [Kang et al. 2001]. This paper does not consider this type of optimization; however, the structure of our algorithm enables its adaptation to other target metrics.

**Organization.** This paper is organized as follows. Section 2 gives the mathematical background and notation used throughout the paper. Section 3 puts the existing graph-based algorithms for the MCM problem into a framework that identifies their common structure. Our new algorithm is presented in Section 4 including a discussion of its main properties. A crucial subroutine of our algorithm estimates the so-called  $\mathcal{A}$ -distance, which is explained in Section 5. Section 6 presents a runtime analysis of our and competing algorithms and shows various experimental results with generated MCM blocks. The results show that our algorithm outperforms the best available methods at the cost of a higher computation time. Finally, we offer conclusions in Section 7.

## 2. BACKGROUND

In this section we formally state the problem of multiple constant multiplication, describe the graphical representation of multiplier blocks, and explain the corresponding mathematical notation used in this paper. The notation introduced in this paper is summarized in Table I and will be used in the following sections to develop a unifying framework for the existing and our new graph-based algorithms.

**$\mathcal{A}$ -operation.** A multiplier block implements the parallel multiplication by a

Table I. Summary of the notation used in the paper.

Notation	Meaning	Defined in
$A$	Upper case letters denote sets	
$a$	Lower case letters denote integers	
$U \cup V$	Union of sets $U$ and $V$	
$U - V$	Difference of sets $U$ and $V$	Equation (3)
$U - a$	Difference of sets $U$ and $\{a\}$	Equation (3)
$U + a$	Union of sets $U$ and $\{a\}$	Equation (4)
$a + b$	Addition of two integers	
$a - b$	Subtraction of two integers	
$UV$	Set of all products	Equation (8)
$\frac{U}{V}$	Set of all integer quotients	Equation (9)
$\text{dist}(U, a)$	$\mathcal{A}$ -distance of $a$ from set $U$	Definition 2.4
$\mathcal{C}_n$	Set of complexity- $n$ constants	Definition 2.3
$\mathcal{A}_p(a, b)$	$\mathcal{A}$ -operation	Definition 2.1
$\mathcal{A}_*(a, b)$	Vertex fundamental set	Definition 2.5
$\mathcal{A}_*(U, V)$	Vertex fundamental set	Definition 2.6
$\mathcal{A}^{\text{odd}}$	$\mathcal{A}$ -operation with odd outputs	Section 2
$R$	Ready set	Section 3.1
$S$	Successor set of $R$	Equation (10)
$S^n$	Set of distance- $n$ constants w.r.t. $R$	Equation (11)
$T$	Targets set	Section 3.1
$B(R, s, t)$	Benefit	Equation (15)
$\bar{B}(R, s, t)$	Weighted benefit	Equation (16)
$H_{\text{maxb}}(R, S, T)$	Maximum benefit heuristic	Section 4.3
$H_{\text{cub}}(R, S, T)$	Cumulative benefit heuristic	Section 4.3
$\text{dist}(U, a) \simeq d$	$d$ is an estimate for $\text{dist}(U, a)$	Section 5.6
$\text{Est}(z)$	Auxiliary cost measure of a constant $z$	Equation (21)
$\text{Est}(Z)$	Minimum auxiliary cost measure for a set $Z$	Equation (22)

given set of constants, which we call *fundamentals* following [Dempster and Macleod 1994], or simply constants. The implementation uses adds, subtracts, and shifts, but to streamline the search process we consolidate these operations into a single parameterized operation called an  $\mathcal{A}$ -operation.

We define an  $\mathcal{A}$ -operation as an operation on the fundamentals. An  $\mathcal{A}$ -operation performs a single addition or a subtraction, and an arbitrary number of shifts, which do not truncate non-zero bits of the fundamental. Because we can merge two consecutive shifts, we state the most general definition as follows:

**DEFINITION 2.1 GENERAL  $\mathcal{A}$ -OPERATION.** *Let  $l_1, l_2 \geq 0$  be integers (left shifts),  $r \geq 0$  be an integer (right shift) and let  $s \in \{0, 1\}$  (sign). An  $\mathcal{A}$ -operation is an operation with two integer inputs  $u, v$  (fundamentals) and one output fundamental, defined as*

$$\begin{aligned} \mathcal{A}_p(u, v) &= |(u \ll l_1) + (-1)^s (v \ll l_2)| \gg r = \\ &= |2^{l_1} u + (-1)^s 2^{l_2} v| 2^{-r}, \end{aligned} \quad (1)$$

where  $\ll$  is a left binary shift,  $\gg$  is a right binary shift, and  $p = (l_1, l_2, r, s)$  is the parameter set or the  $\mathcal{A}$ -configuration of  $\mathcal{A}_p$ . To preserve all significant bits of the

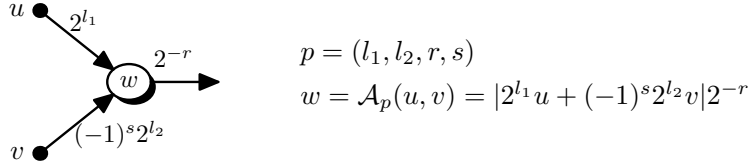


Fig. 6.  $\mathcal{A}$ -operation:  $u$  and  $v$  are the input fundamentals and  $w$  is the output fundamental.  $\mathcal{A}$ -operations directly connected to the input of the multiplier block have  $u = v = 1$ .

output,  $2^r$  must divide  $2^{l_1}u + (-1)^s 2^{l_2}v$ .

Without loss of generality, we consider positive fundamentals only. The absolute value in the definition of  $\mathcal{A}$  enforces positive fundamentals, and enables the subtraction to be done in only one direction, which simplifies the definition.

We represent an  $\mathcal{A}$ -operation graphically as shown in Fig. 6. Note that the result  $w$  is written inside the node, even though the right shift by  $r$  has already been applied to  $w$ . The reason for this notation is that what matters later in the considered MCM algorithms is the unique odd fundamental at each node. Also, the final right shift by  $r$  will be fused in the full  $\mathcal{A}$ -graph with a subsequent left shift in the next  $\mathcal{A}$ -operation. Originally, the final right shift  $r$  in the  $\mathcal{A}$ -operation was not used; however, it is necessary to obtain the full range of possible outputs. For example, for odd  $u$  and  $v$  and  $l_1 = l_2 = 0$ ,  $u + v$  is even and a right shift can be applied to obtain a new odd fundamental that may not be possible to obtain otherwise with only one  $\mathcal{A}$ -operation.

All MCM algorithms impose constraints on the  $\mathcal{A}$ -configuration. In particular, an upper bound on the left shifts  $l_1$  and  $l_2$  is necessary to make the set of choices finite. In practice, it is sufficient to limit these shifts by the bitwidth of the target constants. Experimental results from [Dempster and Macleod 1994; Lefèvre 2003; Gustafsson et al. 2002] show that allowing larger shifts does not improve upon the optimal solutions<sup>1</sup> obtained with the former limits. However, in the general case, the smallest obtained theoretical limit on the shifts sufficient to obtain optimal multiplier blocks is larger [Dempster and Macleod 1994]. Additional insights are given in [Lefèvre 2003]. Although the authors do not compute this particular bound ( $c_{\text{inf}}$  in their notation), they prove that optimal multiplier blocks cannot have shifts larger than a certain finite value ( $c_{\text{sup}}$  in their notation). The results of [Lefèvre 2003] and [Dempster and Macleod 1994] are not directly comparable, since Lefèvre did not take into account right shifts.

Some constraints on  $p$  may lead to different problem classes. For example, if we restrict  $l_1 = l_2 = r = s = 0$ , then the SCM problem reduces to the well known problem of finding the shortest addition chain [Knuth 1969], since the  $\mathcal{A}$ -operation in this case is an ordinary addition. However, even a lesser restriction, like  $s = 0$ , which disallows subtractions, can require major changes of the MCM algorithm. Thus, it is crucial that the  $\mathcal{A}$ -operation and the  $\mathcal{A}$ -configuration constraints are explicitly defined for any MCM algorithm.

Since in this paper we are only interested in reducing the number of add/subtract

<sup>1</sup>with respect to the number of add/subtract operations.

operations in a multiplier block, we will refer to  $\mathcal{A}$ -operations also as add/subtract operations, although in reality they include also shifts.

The rationale for neglecting shifts is that in hardware shifts can be implemented as wires which require virtually no area, and in software, some CPUs (e.g., Intel XScale) support combined add/shift and subtract/shift instructions, which execute as fast as a single add or subtract. If desired, the minimization of the number of shifts can be incorporated as a secondary criteria in our MCM algorithm.

**$\mathcal{A}$ -graph.** As we have already seen in Fig. 2 and Fig. 5, the structure of a multiplier block can be represented as a directed graph. We call such graph an  $\mathcal{A}$ -graph, since it is built out of the  $\mathcal{A}$ -operations shown in Fig. 6. The vertices of an  $\mathcal{A}$ -graph are labeled with their respective fundamentals; hence the input vertex has label 1. The edges are labeled with a 2-power scaling factor equivalent to the performed shift. Negative edge values are used to indicate subtractions at the following vertex.

**Formal problem statement.** Now we can formally state the problem of constructing multiplier blocks.

**DEFINITION 2.2 MCM PROBLEM.** **Given** a set of positive target constants  $T = \{t_1, \dots, t_n\} \subset \mathbb{N}$ . **Find** the smallest set  $R = \{r_0, r_1, \dots, r_m\}$  with  $T \subset R$ , such that  $r_0 = 1$ , and for all  $r_k$  with  $1 \leq k \leq m$  there exist  $r_i, r_j$  with  $0 \leq i, j < k$ , and an  $\mathcal{A}$ -configuration  $p_k$  such that

$$r_k = \mathcal{A}_{p_k}(r_i, r_j). \quad (2)$$

The set of  $\mathcal{A}$ -graph fundamentals  $R$  and the set of  $\mathcal{A}$ -configurations  $p_k$  uniquely define an  $\mathcal{A}$ -graph for an MCM block with  $m = |R| - 1$  add/subtract operations.

The number of  $\mathcal{A}$ -operations in an optimal solution for a given set  $T$  is called the  $\mathcal{A}$ -complexity of  $T$ . If any  $\mathcal{A}$ -graph for  $T$  is given, optimal or not, we refer to its number of required  $\mathcal{A}$ -operations as cost of this graph.

All constants can be divided into complexity classes.

**DEFINITION 2.3 COMPLEXITY-N CONSTANTS.** We denote by  $C_n$  the set of all constants with complexity  $n$ , i.e. those, for which an optimal SCM solution requires exactly  $n$   $\mathcal{A}$ -operations. For example,  $C_0 = \{2^a \mid a \geq 0\}$ , because precisely all 2-power constants require a single left shift and no adds/subtracts.

Although the sets  $C_n$  are infinite, we will always limit our discussion to constants up to certain bitwidth  $b$ , which is always explicitly stated. The set of complexity  $n$  constants obeying this constraint is then finite and, by abuse of notation, will also be denoted by  $C_n$ .

Next, we introduce the notion of  $\mathcal{A}$ -distance, the key component in our proposed algorithm.

**DEFINITION 2.4  $\mathcal{A}$ -DISTANCE.** Let  $c \in \mathbb{N}$  be a constant, and let  $R \subset \mathbb{N}$  be a set of constants (fundamentals of an  $\mathcal{A}$ -graph). Then, the  $\mathcal{A}$ -distance of  $c$  from the set  $R$ , denoted with  $\text{dist}(R, c)$ , is the minimum number of extra  $\mathcal{A}$  operations required to obtain  $c$  given  $R$ .

The  $\mathcal{A}$ -distance corresponds to the notion of “adder distance” in [Dempster and Macleod 1995].

For example,  $\text{dist}(\{1\}, T)$  is equal to the complexity of  $T$ , and for all  $c \in \mathcal{C}_n$ ,  $\text{dist}(\{1\}, c) = n$ .

For simplicity, we write in this paper the difference of two sets  $U$  and  $V$  as

$$U - V = U \setminus V, \quad \text{and for one element } U - a = U \setminus \{a\}. \quad (3)$$

For visual appeal we will also use a  $+$  for the union of a set with a single element, i.e.,

$$U + a = U \cup \{a\}. \quad (4)$$

To express the degree of freedom in the output of an  $\mathcal{A}$ -operation when different  $\mathcal{A}$ -configurations are chosen, we define next the *vertex fundamental set*.

**DEFINITION 2.5 VERTEX FUNDAMENTAL SET.** *The set of all possible outputs (not equal to the inputs) of an  $\mathcal{A}$ -operation with fixed inputs ( $u$  and  $v$ ) under different  $\mathcal{A}$ -configurations is called the vertex fundamental set, written as*

$$\mathcal{A}_*(u, v) = \{\mathcal{A}_p(u, v) \mid p \text{ is a valid configuration}\} - u - v. \quad (5)$$

The definition of *valid*  $\mathcal{A}$ -configurations  $p$  will be introduced later for each discussed algorithm.

It is useful to extend the definition of  $\mathcal{A}_*$  to sets of inputs.

**DEFINITION 2.6.** *If  $U, V \subset \mathbb{N}$  are sets of fundamentals, then*

$$\mathcal{A}_*(U, V) = \bigcup_{\substack{u \in U \\ v \in V}} \mathcal{A}_*(u, v) - U - V. \quad (6)$$

From this definition it follows that for sets  $U, V$ , and  $W$ :

$$\begin{aligned} \mathcal{A}_*(U \cup V, W) &= (\mathcal{A}_*(U, W) - V) \cup (\mathcal{A}_*(V, W) - U) = \\ &= \mathcal{A}_*(U, W) \cup \mathcal{A}_*(V, W) - U - V. \end{aligned} \quad (7)$$

Further we define the product of sets  $U, V$  in the usual way

$$UV = \{uv \mid u \in U, v \in V\}, \quad (8)$$

and their quotient as

$$\frac{U}{V} = \left\{ \frac{u}{v} \mid u \in U, v \in V, v \text{ divides } u \right\}. \quad (9)$$

**Odd fundamental graphs.** Any  $\mathcal{A}$ -graph can be converted into an  $\mathcal{A}$ -graph of equal cost, which has only odd fundamentals [Dempster and Macleod 1994]. Such graphs are called *odd fundamental graphs*. As an example, the graph in Fig. 5 is an odd fundamental graph. This reduction is possible, because any even constant can be obtained from an odd constant by a suitable shift. Odd fundamental graphs are important because they reduce the degree of freedom in choosing graph fundamentals without affecting the cost of the graph.

To obtain odd fundamental graphs, the valid  $\mathcal{A}$ -configuration must be constrained to allow at most one non-zero left shift  $l_1$  or  $l_2$ , and if  $l_1 = l_2 = 0$  force  $r$  to be the unique right shift that produces an odd value. We will sometimes use  $\mathcal{A}^{\text{odd}}$  to indicate these constraints. For given  $u$  and  $v$ , the only free parameters in  $\mathcal{A}^{\text{odd}}$  are  $s$  and the non-zero left shift (either  $l_1$  or  $l_2$ ), in contrast to the general  $\mathcal{A}$ -operation,

---

**Algorithm 1. High-Level Structure of Graph-Based MCM Algorithms.** Given the target set of constants  $T$ . Compute (synthesize) the set  $R = \{r_1, \dots, r_m\}$ , with  $T \subset R$ , as given in Definition 2.2.

---

**SynthesizeMultiplierBlock( $T$ )**

- 1:  $R \leftarrow \{1\}$
- 2: **while**  $T \neq \emptyset$  **do**
- 3:   compute the successor set  $S$  of  $R$
- 4:   select  $s \in S$  based on a heuristic
- 5:   Synthesize( $s$ )

**Synthesize( $s$ )**

- 1:  $R \leftarrow R + s$
  - 2:  $T \leftarrow T - s$
- 

where  $l_1$ ,  $l_2$ ,  $r$ , and  $s$  can vary. Thus, the space of valid  $\mathcal{A}$ -configurations  $p$  is considerably reduced.

Algorithms that use  $\mathcal{A}^{\text{odd}}$  preprocess all target constants with a suitable right shift to make them odd.

### 3. OVERVIEW OF MCM ALGORITHMS

Using the notation introduced in the previous section, we put the existing graphical MCM algorithms into a common context and identify their common structure. Then we discuss the most important algorithms in greater detail.

#### 3.1 General Framework

The existing graph-based algorithms for multiplier block synthesis share the same high-level structure, which is shown in Algorithm 1 and explained in the following. As said before, it is necessary that the notion of  $\mathcal{A}$ -operation considered by the algorithm is precisely defined including all constraints.

The input to Algorithm 1 is the *target set*  $T$  of constants. The set  $R$  used in the algorithm is called the *ready set*. It is initialized in step 1 with the first fundamental 1 and iteratively augmented in the loop in step 2 with additional fundamentals. Upon termination, i.e., when  $T \subset R$ ,  $R$  is output as the solution. In each iteration of the loop in step 2, an element of the *successor set*  $S$  of  $R$  is chosen as the next fundamental based on a heuristic. Formally,

$$S = \{s \mid \text{dist}(R, s) = 1\} = \mathcal{A}_*(R, R) \quad (10)$$

is the set of all constants of distance 1 from  $R$ . Even though  $S$  depends on  $R$  we do not write  $S_R$  or  $S(R)$  to simplify the notation. In Algorithm 1 we do not specify whether or how  $S$  is computed, which is discussed later.

Once  $s \in S$  is chosen, it is added to  $R$ , and, if  $s \in T$ , removed from  $T$ . We call this process *synthesizing*  $s$ . Alternatively, the algorithm may use a heuristic to choose constants  $s$  at a higher distance,

$$s \in S^n = \{s \mid \text{dist}(R, s) = n\}. \quad (11)$$

In this case, all  $n - 1$  intermediate fundamentals have to be synthesized as well. We call  $S^n$  the distance- $n$  set (of  $R$ ). Clearly,  $S^1 = S$ . Although  $S$  does not have to be explicitly enumerated and stored, some algorithms do so.

The procedure is repeated until the target set is empty, i.e., all of the target constants are synthesized.

The heuristic used in step 4, which determines the next fundamental or the next vertex in the  $\mathcal{A}$ -graph to be synthesized, is highly dependent on the  $\mathcal{A}$ -operation used within an MCM algorithm. Further, the heuristic is what differentiates the various algorithms and what determines their performance.

In the following we discuss the three most important graph-based MCM algorithms, each of which is an instantiation of Algorithm 1.

### 3.2 Bull-Horrocks Algorithm (BHA)

[Bull and Horrocks 1991] designed four MCM algorithms: for add, add/subtract, add/shift, and add/subtract/shift decompositions. Here we discuss the latter one, since it addresses the problem considered in this paper. We refer to this algorithm as BHA.

The  $\mathcal{A}$ -operation considered in BHA imposes constraints on the configuration by requiring  $r = 0$ , and  $\mathcal{A}_p(u, v) \leq \min(T)$ . In words, right shifts are not allowed and as intermediate fundamentals only numbers smaller as the current  $\min(T)$  are synthesized. This also imposes an implicit bound on the shifts  $l_1$  and  $l_2$  in the  $\mathcal{A}$ -configuration.

The heuristic synthesizes targets  $T$  in ascending order. Since targets are removed from  $T$  in the synthesize step, the next target to be synthesized is always  $\min(T)$ .

The heuristic used in BHA keeps track of the so-called “error”

$$\epsilon = \min(T) - \max(R). \quad (12)$$

If  $\epsilon \in R$ , then the candidate target can be directly synthesized, and the algorithm proceeds. Otherwise, two successors  $s_1$  and  $s_2$  that minimize the error are synthesized, chosen as follows:

$$\begin{aligned} s_1 &= \arg \min_{s \in S, s \leq \epsilon} (\epsilon - s), \text{ and} \\ s_2 &= \max(R) + s_1. \end{aligned}$$

In particular, when  $\epsilon \in S$ , then  $s_1 = \epsilon$  and  $s_2 = \epsilon + \max(R) = \min(T)$ , i.e., the candidate target is synthesized.

The algorithm considers only the magnitude of the error, and the binary representation of the constants is not taken into account, unlike in CSE algorithms.

### 3.3 Bull-Horrocks Modified Algorithm (BHM)

[Dempster and Macleod 1995] improved BHA and called it the *Bull-Horrocks Modified Algorithm (BHM)*.

The  $\mathcal{A}$ -operation considered in BHM is  $\mathcal{A}^{\text{odd}}$ . Accordingly, all targets  $T$  are preprocessed by right-shifts to become odd. The  $\mathcal{A}$ -configuration constraints are relaxed to allow fundamentals larger than constants in  $T$ , namely up to  $\mathcal{A}_p^{\text{odd}}(u, v) \leq 2 \max(T)$ , which stimulates the use of subtractions.

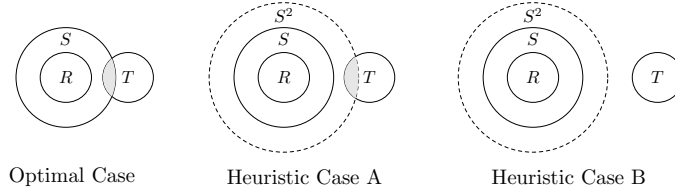


Fig. 7. The three cases considered by the RAG-n heuristic. The dashed circle means that  $S^2$  is not computed explicitly.

The heuristic in BHM is changed from BHA in the following way. First, the targets are synthesized in the order of increasing  $\mathcal{A}$ -complexity, which is obtained from a precomputed lookup table, or estimated by, e.g., the CSD cost. Second, the error in (12) is allowed to be negative. Finally, because  $\mathcal{A}^{\text{odd}}$  is used and all elements of  $R$  are odd, minimization applies left shifts to candidate successors.

Let  $\text{minc}(T)$  denote the next candidate target (target of minimal complexity or cost), and let  $r_c \in R$  denote the closest (magnitude-wise) fundamental to  $\text{minc}(T)$ , i.e.  $r_c = \text{minc}(T) \pm \epsilon$ . Then the chosen successors  $s_1$  and  $s_2$  are determined as follows:

$$(s_1, k) = \arg \min_{\substack{s \in S \\ 0 \leq k \leq \lceil \log_2 \epsilon \rceil}} |\epsilon - s \ll k|, \quad (13)$$

$$s_2 = r_c \pm s_1 \ll k.$$

Just as in BHA, two successors are synthesized per iteration, unless  $\epsilon \in R$ .

Due to these modifications, BHM performs consistently better than BHA in terms of the number of  $\mathcal{A}$ -operations in the generated  $\mathcal{A}$ -graphs.

### 3.4 n-dimensional Reduced Adder Graph (RAG-n)

RAG-n [Dempster and Macleod 1995] is a graph-based MCM algorithm that requires a precomputed table of optimal SCM decompositions, obtained by exhaustive search using the method in [Dempster and Macleod 1994].

The target  $\mathcal{A}$ -operation in RAG-n is  $\mathcal{A}^{\text{odd}}$ , and as in BHM all targets  $T$  are first right-shifted to become odd. The  $\mathcal{A}$ -configuration constraints are less restrictive than in BHM, with the only constraint being  $\mathcal{A}_p(x, y) \leq 2^{b+1}$ , where  $b$  is the maximum bitwidth of the targets. The RAG-n heuristic considers three different cases, graphically illustrated in Fig. 7, and discussed next.

- (1) *Optimal Case.* If  $T \cap S \neq \emptyset$ , then there is a target in the successor set, and it is synthesized. If the entire set  $T$  is synthesized this way, then the solution is optimal, since it is impossible to use less than one  $\mathcal{A}$ -operation for each odd target. Thus, this case is called optimal.
- (2) *Heuristic Case A.* If  $T \cap S = \emptyset$  and  $T \cap S^2 \neq \emptyset$ , then there is a target at an  $\mathcal{A}$ -distance of 2 from  $R$ . This target is synthesized along with the distance-1 intermediate fundamental.
- (3) *Heuristic Case B.* If no distance-1 or distance-2 targets are available, then RAG-n synthesizes the target of least complexity using the precomputed optimal SCM table. In this case three or more constants are synthesized.

RAG-n computes the entire set  $S$  and then finds the intersection  $S \cap T$  to detect distance-1 targets. Distance-2 targets, on the other hand, are detected using a *heuristic distance test* only. This test is cheaper than the full computation of  $S^2$  but does not detect all distance-2 numbers. We express this in Fig. 7 by using a dashed circle for  $S^2$ .

The last case (Heuristic Case B) uses the precomputed SCM decomposition to decompose a target. In general, this does not lead to any subexpression sharing. With the current methods, the largest available optimal SCM table is restricted to constants up to 19 bits [Gustafsson et al. 2002], which also limits the applicability of RAG-n.

RAG-n applied to a single constant set will always yield an optimal decomposition if the constant resides in the precomputed optimal SCM lookup table. Constants not in the lookup table are only synthesized at all (optimal or not) if they have complexity 1 or 2, i.e., have a distance 1 or 2 from the initial  $R = \{1\}$ . Since RAG-n uses the lookup table it can only be considered applicable to target sets with 2 or more constants. Other MCM algorithms, including BHA, BHM, and the proposed new algorithm, can be applied to the SCM problem directly.

### 3.5 Hybrid Graph-Based Algorithms

The common structure of graph-based algorithms makes it possible to easily mix different algorithms to obtain hybrids. For example, in any given iteration, one can change the heuristic, or use a different algorithm to synthesize a target.

For example, RAG-n itself is a hybrid with three components. When its optimal, distance-1 test fails, the algorithm switches to a distance-2 heuristic, and when this also fails, RAG-n reverts to a lookup table to synthesize a fundamental. Similarly, RAG-n can be easily modified to use a CSE based algorithm, instead of a lookup table, to synthesize the targets not detected by its distance-1 and distance-2 tests.

## 4. NEW ALGORITHM

This section first describes the limitations of the existing algorithms introduced in Section 3, and then presents in detail the proposed new algorithm. The heuristic in the new algorithm assumes that there is a function that will compute exactly or estimate the  $\mathcal{A}$ -distance. This is a non-trivial problem and will be addressed separately in Section 5.

### 4.1 Limitations of Existing Graph-based Algorithms.

BHA was designed more than a decade ago, when workstation computers had several orders of magnitude less computing power, and thus uses a computationally cheap heuristic by today's standards. Although BHM improved on BHA, the heuristic was not changed significantly. So it can be expected that on today's computers we can improve on BHA and BHM by choosing a computationally more complex and thus more precise heuristic.

RAG-n performs reasonably well, however, one main disadvantage is the dependence on a precomputed table of optimal single constant add/subtract/shift decompositions. First, this table takes a time exponential in the number of bits to construct. Second, the best available method to compute this table [Gustafsson et al. 2002] works only up to 19 bits (even though one could also use a good

suboptimal table, which was not investigated). Finally, the size of the table is also exponential in the number bits (it must have an entry for every odd constant). For instance, the table for all odd constants up to 32 bits requires  $2^{31} > 10^9$  entries. Thus, RAG-n is not applicable to large bitwidths.

Another shortcoming of RAG-n (and BHA, BHM) is that the heuristic does not try to select intermediate fundamentals to *jointly* optimize for all target constants. This often results in suboptimal answers, even in very simple cases. For example, RAG-n applied to the target set  $T = \{23, 81\}$  synthesizes a multiplier block that uses 4  $\mathcal{A}$ -operations, without any subexpression sharing (both 23 and 81 are complexity-2 constants). In contrast, Fig. 5 shows a better solution with only 3 operations produced by our algorithm.

## 4.2 New Algorithm

The main idea behind our new algorithm is to use a better heuristic for synthesizing intermediate fundamentals. Our algorithm is computationally more expensive than BHA, BHM, and RAG-n, since it explores a very large space of possible intermediate vertices. Unlike RAG-n, it does not require a pregenerated optimal SCM lookup table. Thus, our algorithm is storage efficient and in its applicability only limited by the computation time.

**Target  $\mathcal{A}$ -operation.** The  $\mathcal{A}$ -operation in our algorithm is  $\mathcal{A}^{\text{odd}}$  with the same  $\mathcal{A}$ -configuration constraint as RAG-n, namely allowing  $\mathcal{A}_p(u, v) \leq 2^{b+1}$ , where  $b$  is the maximal bitwidth of constants in  $T$ . When describing the algorithm we will use for simplicity  $\mathcal{A}$  to denote  $\mathcal{A}^{\text{odd}}$  with the above constraint.

**Outline of the new algorithm.** Our algorithm follows the general structure of Algorithm 1 and is shown in Algorithm 2. The heuristic is split into two parts: the optimal and the heuristic part, similarly to the RAG-n algorithm. The optimal part is practically identical to the optimal part in RAG-n, but in Algorithm 2 we give a more detailed explanation of how to efficiently construct the successor set  $S$ . The heuristic part uses  $S$  and the  $\mathcal{A}$ -distance tests and estimators developed in Section 5 to select new successors  $s$  to be added to  $R$ . Unlike in RAG-n, our heuristic part only adds a single successor to  $R$  at each iteration.

When we talk about a single iteration of the algorithm (e.g., later in its analysis), we refer to the outer loop consisting of steps 5–18.

Next, we discuss Algorithm 2 in greater detail.

**Construction of  $S$  and the optimal part.** The optimal part of our algorithm is equivalent to the optimal part of RAG-n. Recall, that the optimal part of RAG-n synthesizes at each iteration all distance-1 targets, i.e.,  $S \cap T$ .

To avoid computing in each iteration the entire set  $S$ , which can become rather large, we compute it incrementally. This necessitates an additional set, the worklist  $W$ . When a constant is synthesized it is added to  $W$ , first without being accounted for in neither  $R$  nor  $S$ . In steps 9–10 we then perform an incremental update of  $R$  and  $S$  based on  $W$ . The update of  $R$  is straightforward (step 9):

$$R_{\text{new}} = R \cup W.$$

---

**Algorithm 2. New MCM Algorithm.** Given the target set of constants  $T$ . Compute the set  $R = \{r_1, \dots, r_m\}$ , with  $T \subset R$ , as given in Definition 2.2 in Section 2. There is a degree of freedom in choosing the heuristic function  $H(R, S, T)$  for the algorithm. We consider two alternatives  $H_{\max b}$  and  $H_{\text{cub}}$  discussed in Section 4.3.

---

**SynthesizeMultiplierBlock( $T$ )**

```

1: Right shift elements of  $T$  until odd
2:  $R \leftarrow \{1\}$ 
3:  $W \leftarrow \{1\}$ 
4:  $S \leftarrow \{1\}$ 
5: while  $T \neq \emptyset$  do
6:   {optimal part}
7:   while  $W \neq \emptyset$  do
8:     {update  $S$  and  $R$ }
9:      $R \leftarrow R \cup W$ 
10:     $S \leftarrow (S \cup \mathcal{A}_*(R, W)) - W$ 
11:     $W \leftarrow \emptyset$ 
12:    {if  $S$  contains targets, synthesize them}
13:    for  $t \in S \cap T$  do
14:      Synthesize( $t$ )
15:    {heuristic part}
16:    if  $T \neq \emptyset$  then
17:       $s \leftarrow H(R, S, T)$ 
18:      Synthesize( $s$ )

```

**Synthesize( $s$ )**

```

1:  $W \leftarrow W + s$ 
2:  $T \leftarrow T - s$ 

```

---

The update formula for  $S$  is derived as follows using (7) and (10).

$$\begin{aligned}
S_{\text{new}} &= \mathcal{A}_*(R_{\text{new}}, R_{\text{new}}) = \mathcal{A}_*(R_{\text{new}}, R \cup W) \\
&= \mathcal{A}_*(R_{\text{new}}, R) \cup \mathcal{A}_*(R_{\text{new}}, W) \\
&= \mathcal{A}_*(R \cup W, R) \cup \mathcal{A}_*(R_{\text{new}}, W) \\
&= \mathcal{A}_*(R, R) \cup \mathcal{A}_*(W, R) \cup \mathcal{A}_*(R_{\text{new}}, W) - W \\
&= S \cup \mathcal{A}_*(R, W) \cup \mathcal{A}_*(R_{\text{new}}, W) - W.
\end{aligned}$$

Since  $\mathcal{A}_*(R, W) \subset \mathcal{A}_*(R_{\text{new}}, W)$ , we get

$$S_{\text{new}} = (S \cup \mathcal{A}_*(R_{\text{new}}, W)) - W, \quad (14)$$

which is step 10 in Algorithm 2.

**Heuristic part.** When no more targets are found in  $S$ , the optimal part of the algorithm cannot synthesize any constants. This means that all targets are more than one  $\mathcal{A}$ -operation away, and a heuristic function  $H(R, S, T)$  is used to find the next successor to add to  $R$ . Adding a successor to  $R$  creates new successors, possibly enabling the optimal part to then synthesize new targets.

We have developed two different heuristic functions  $H_{\max b}$  and  $H_{\text{cub}}$ , discussed next.

### 4.3 Heuristics.

We present two heuristic functions  $H$  considered in our algorithm, called *Maximum Benefit* and *Cumulative Benefit*. Both heuristics assume that one can compute or estimate the  $\mathcal{A}$ -distance. This is a non-trivial problem, and the next section is dedicated to  $\mathcal{A}$ -distance computation/estimation.

**Heuristic 1: Maximum benefit.** The obvious and natural heuristic that comes to mind, assuming that it is possible to compute or estimate the  $\mathcal{A}$ -distance, is to pick the successor  $s \in S$  closest to the target set. However, it is useful to also take into account the current estimate of the distance between  $R$  and  $T$ . Thus, to build our heuristic, we first define the *benefit function*  $B(R, s, t)$  to quantify how much adding a successor  $s$  to the ready set  $R$  improves the distance to a fixed, but arbitrary target  $t$ :

$$B(R, s, t) = \text{dist}(R, t) - \text{dist}(R + s, t). \quad (15)$$

(Recall that we write  $R + s$  for  $R \cup \{s\}$ .) If  $\text{dist}$  is exact, then  $B$  is at most 1. For targets farther away, however, the  $\mathcal{A}$ -distance can only be estimated, and with these estimated distances the benefit can be larger than 1. Moreover, for remote targets the estimate becomes less accurate (refer to Section 5), but also less important. We take this into account by introducing the *weighted benefit* function

$$\bar{B}(R, s, t) = 10^{-\text{dist}(R+s,t)} (\text{dist}(R, t) - \text{dist}(R + s, t)), \quad (16)$$

where the weight factor  $10^{-\text{dist}(R+s,t)}$  is exponentially decreasing as the distance to  $t$  grows. Initially, we always favored closer targets, which is equivalent to having a very large exponent base, but later it was experimentally found that 10 slightly improves the solutions obtained with our second heuristic.

The *maximum benefit heuristic*  $H_{\max b}(R, S, T)$  used in our algorithm picks the successor  $s \in S$  that maximizes the weighted benefit over all targets  $t \in T$ :

$$H_{\max b}(R, S, T) = \arg \max_{s \in S} \left( \max_{t \in T} \bar{B}(R, s, t) \right), \quad (17)$$

Maximizing  $\bar{B}$  tends to prefer the successors that are closest to  $T$ , but between two successors that are equally far from  $T$ , it prefers the one with the greater benefit value.

**Heuristic 2: Cumulative benefit.** The key observation about the benefit function is that benefits for different targets  $t$  can be added up to enable joint optimization for all targets. This leads to the second and, as it turns out, superior *cumulative benefit* heuristic, formally defined as

$$H_{\text{cub}}(R, S, T) = \arg \max_{s \in S} \left( \sum_{t \in T} \bar{B}(R, s, t) \right). \quad (18)$$

The cumulative benefit heuristic adds up the weighted benefits with respect to all targets in  $T$ , instead of taking the maximum, and thus accomplishes joint optimization for all targets.

**Remarks.** In a sense, our first heuristic  $H_{\max b}$  corresponds to a maximum norm  $\|\cdot\|_{\infty}$ , and the second heuristic  $H_{\text{cub}}$  to the 1-norm  $\|\cdot\|_1$  (of course, a proper norm would require an underlying vector space). We have also considered the equivalent of the 2-norm  $\|\cdot\|_2$  but it did not produce results significantly different from  $H_{\text{cub}}$ .

#### 4.4 Termination and Distance Function Admissibility

The inner loop (steps 7–14 in Algorithm 2) of the optimal part is guaranteed to terminate, since there is a finite number of targets, and at each iteration either a target is synthesized or the optimal part finishes.

The heuristic part, on the other hand, does not synthesize target constants. Since one constant at a time is synthesized, it would have to be a distance-1 target. However, if the test for the optimal part fails, i.e.,  $T \cap S = \emptyset$ , it is guaranteed that no distance-1 targets exist. Thus, the heuristic part only synthesizes intermediate vertices, and the algorithm is not guaranteed to terminate, unless the heuristic  $H$  meets certain conditions.

For  $H_{\max b}$  and  $H_{\text{cub}}$ , the termination is guaranteed, if for an arbitrary ready set  $R \supseteq \{1\}$ , the corresponding successor set  $S$ , and any  $t \in T$ , the  $\mathcal{A}$ -distance estimation function  $\text{dist}$  is *admissible*, which is defined next.

**DEFINITION 4.1 ADMISSIBLE  $\mathcal{A}$ -DISTANCE ESTIMATION FUNCTION.** *We call an  $\mathcal{A}$ -distance estimation function  $\text{dist}$  admissible, if the following holds:*

- (1)  $\text{dist}(R, t)$  is a finite nonnegative integer;
- (2)  $\text{dist}(R, t) = 0$  iff  $t \in R$ ;
- (3)  $\text{dist}(R, t) > 0$  iff  $t \notin R$ ;
- (4) For an arbitrary set  $U$ ,  $\text{dist}(R \cup U, t) \leq \text{dist}(R, t)$ ;
- (5) If  $t \notin R$ , then there exists  $s \in S$  such that

$$B(R, s, t) = \text{dist}(R, t) - \text{dist}(R + s, t) > 0 \quad \text{or} \quad \text{dist}(R + s, t) < \text{dist}(R, t).$$

**THEOREM 4.2 ALGORITHM TERMINATION.** *Algorithm 2 with the heuristic  $H = H_{\max b}$  or  $H = H_{\text{cub}}$  terminates if  $\text{dist}$  is admissible.*

**PROOF.** Consider the sum of estimated distances to all unsynthesized targets  $D = \sum_{t \in T} \text{dist}(R, t)$ . The admissibility conditions 1–3 in Definition 4.1 imply that  $D$  is finite, and  $D > 0$  for  $T \neq \emptyset$ , and that  $D = 0$  iff  $T = \emptyset$ , i.e., when all targets are synthesized.

Both heuristics choose a successor with positive benefit if it exists.<sup>2</sup> According to condition 5 such a successor  $s$  always exists and

$$\text{dist}(R + s, t) < \text{dist}(R, t).$$

Thus, the intermediate fundamental chosen by the heuristic will decrease at least one distance estimate, and since according to condition 4, the estimates cannot increase with an addition of new elements to  $R$ , the sum  $D$  will also decrease.

<sup>2</sup>Although  $H_{\text{cub}}$  and  $H_{\max b}$  use the weighted benefit  $\bar{B}$ , it can be easily seen that if  $\bar{B}(R, s, t) > 0$ , then also  $B(R, s, t) > 0$ .

Table II. Worst case set sizes for the new algorithm.

Set	Worst case size
$\mathcal{A}_*(u, v)$	$O(b)$
$\mathcal{C}_1$	$O(b)$
$\mathcal{C}_2$	$O(b^2)$
$T$	$n$
$R$	$O(nb)$
$S$	$O(n^2b^3)$

Since at each iteration  $D$  is decreased, eventually it will become 0. Then, according to condition 2, all targets are synthesized and the algorithm terminates. Observe also that  $D$  is the maximum number of iterations of the heuristic part. The optimal part decreases  $D$  by at least 1 for each synthesized target, and the heuristic part decreases  $D$  by at least 1 for each synthesized intermediate constant, thus  $D$  is also the maximum number of synthesized constants or  $|R|$ .  $\square$

Obviously, if  $\text{dist}$  is exact and not an estimate it is admissible.

#### 4.5 Algorithm Properties

Let  $n = |T|$ , and let  $b$  be the maximal bitwidth of all constants in  $T$ . We derive the worst case sizes of  $\mathcal{A}_*$ ,  $R$  (i.e., the worst case solution),  $S$ , and  $\mathcal{C}_1$  and  $\mathcal{C}_2$ . These bounds are necessary for the runtime analysis of the  $\mathcal{A}$ -distance computation in Section 5 and hence of the algorithm. Table II summarizes these bounds. We will also identify scenarios in which the algorithm produces an optimal solution. Both worst case set sizes and optimality hold under the constraint  $\mathcal{A}_p(u, v) \leq 2^{b+1}$ .

**Worst case size of  $\mathcal{A}_*$ .**  $\mathcal{A}^{\text{odd}}$  allows only one non-zero left shift  $l_1$  or  $l_2$ , and since the fundamental values are limited to  $2^{b+1}$ , the shifts can be between 0 and  $b + 1$ . The only other parameter that can vary is  $s = \{0, 1\}$ . Therefore, for a fixed  $u$  and  $v$ , there are at most  $O(4(b+2)) = O(b)$  possible  $\mathcal{A}$ -configurations, and hence at most  $O(b)$  elements in  $\mathcal{A}_*(u, v)$ .

**Worst case solution ( $|R|$ ).** As mentioned in the proof of Theorem 4.2, the size of the solution is bounded as

$$|R| \leq D = \sum_{t \in T} \text{dist}(R, t). \quad (19)$$

The  $\mathcal{A}$ -distance estimation function, presented in the next section, is bounded by the CSD cost of  $t$ . Therefore for  $b$  bit constants (with a CSD cost of  $O(b)$ ), and  $n$  targets, the worst case is

$$|R| = O(nb).$$

**Worst case size of  $S$ .** Recall that  $S = \mathcal{A}_*(R, R)$ . The number of pairs in  $R$  (with repetition) is at most

$$|R| + \binom{|R|}{2} = O(nb) + \binom{O(nb)}{2} = O(n^2b^2).$$

Since for each  $r, r' \in R$ ,  $\mathcal{A}_*(r, r')$  contains  $O(b)$  elements, we have

$$|S| = O(n^2b^2 \cdot b) = O(n^2b^3).$$

**Sizes of  $\mathcal{C}_1$  and  $\mathcal{C}_2$ .** These principally infinite sets (see Definition 2.3) become finite, if we take into account the restriction of  $\mathcal{A}_p(u, v) \leq 2^{b+1}$  imposed by the algorithm.

Each odd constant in  $\mathcal{C}_1$  is in  $\mathcal{A}_p(1, 1)$ , since 1 is the only odd constant with complexity zero. Thus,

$$|\mathcal{C}_1| = |\mathcal{A}_*(1, 1)| = O(b).$$

Each constant in  $\mathcal{C}_2$  is either in  $\mathcal{A}_p(1, c)$  or in  $\mathcal{A}_p(c, c)$  for a suitable  $c \in \mathcal{C}_1$ . Thus,

$$\mathcal{C}_2 = \mathcal{A}_*(\mathcal{C}_1, 1) \cup \bigcup_{c \in \mathcal{C}_1} \mathcal{A}_*(c, c).$$

It is easy to see that  $\mathcal{A}_*(c, c) = c \cdot \mathcal{A}_*(1, 1)$ , and thus

$$\bigcup_{c \in \mathcal{C}_1} \mathcal{A}_*(c, c) = \bigcup_{c \in \mathcal{C}_1} c \cdot \mathcal{A}_*(1, 1) = \mathcal{C}_1 \mathcal{A}_*(1, 1) = \mathcal{C}_1 \cdot \mathcal{C}_1.$$

We have  $\mathcal{C}_2 = \mathcal{A}_*(\mathcal{C}_1, 1) \cup \mathcal{C}_1 \cdot \mathcal{C}_1$  and  $|\mathcal{C}_2| = O(b^2)$ .

**Optimality.** In certain situations Algorithm 2 produces an optimal solution, i.e., with the minimum possible number of  $\mathcal{A}$ -operations.

**THEOREM 4.3 SINGLE CONSTANT OPTIMALITY.** *If the  $\mathcal{A}$ -distance function  $\text{dist}$  is exact, then Algorithm 2 with either  $H_{\text{cub}}$  or  $H_{\text{maxb}}$  is optimal for a single target ( $n = 1$ ).*

**PROOF.** Denote the single target with  $t$ . Then  $|R| \leq \text{dist}(\{1\}, t)$  from (19). Since the distance function is exact,  $\text{dist}(\{1\}, t)$  is the  $\mathcal{A}$ -complexity of  $t$  and the result follows.  $\square$

**THEOREM 4.4 MULTIPLE CONSTANT OPTIMALITY.** *If the optimal part of Algorithm 2 synthesizes the entire set  $T$ , then the solution is optimal.*

**PROOF.** This was already shown for the RAG-n algorithm in [Dempster and Macleod 1995], which uses the same optimal part.

If after the first pass of the algorithm  $T$  is empty, then  $R$  contains all targets, and the solution uses exactly  $n = |T|$   $\mathcal{A}$ -operations if  $T$  has distinct odd constants. It is not possible to use less operations, because each unique odd target requires at least one  $\mathcal{A}$ -operation.

Observe, that  $n$  is also the lower bound for the number of  $\mathcal{A}$ -operations for  $n$  odd constants. Asymptotically, due to the optimal part of the algorithm, the number of  $\mathcal{A}$ -operations will approach  $n$ , when  $b$  is fixed. Further discussion can be found in [Dempster and Macleod 1995].  $\square$

## 5. COMPUTING THE $\mathcal{A}$ -DISTANCE

The previous section described the proposed algorithm and the two heuristics  $H_{\text{cub}}$  and  $H_{\text{maxb}}$  both of which are based on a function  $\text{dist}$  that computes or estimates the  $\mathcal{A}$ -distance. Our design of this  $\text{dist}$  function is described in this section.

First, we show that the  $\mathcal{A}$ -distance computation is an NP-complete problem to motivate the use of estimation. Second, we discuss the special cases, in which the  $\mathcal{A}$ -distance can be computed exactly, and then describe a general method for its estimation. Finally, we prove the admissibility (see Definition 4.1) of the proposed

distance function, which guarantees the termination of our algorithm through Theorem 4.2.

**THEOREM 5.1 COMPLEXITY OF COMPUTING  $\mathcal{A}$ -DISTANCE.** *The problem of computing the  $\mathcal{A}$ -distance under the constraint  $\mathcal{A}_p(u, v) \leq 2^{b+1}$  is NP-complete.*

**PROOF.** We prove this by reducing the NP-complete problem of finding the optimal decomposition for a single constant<sup>3</sup> to the problem of  $\mathcal{A}$ -distance computation in polynomial time.

If the  $\mathcal{A}$ -distance function is exact, then Algorithm 2 is optimal for a single constant (see Theorem 4.3). The heuristic is invoked in the algorithm once per iteration. There are  $O(nb) = O(b)$  iterations ( $n = 1$  for a single constant), and  $O(|S|) = O(n^2b^3) = O(b^3)$  weighted benefits to compute per iteration. Thus, the  $\mathcal{A}$ -distance is computed  $O(b^4)$  times. Therefore, the optimal single constant decomposition is reduced to  $\mathcal{A}$ -distance computation in polynomial time. Hence,  $\mathcal{A}$ -distance computation is NP-complete.  $\square$

Note that computing the  $\mathcal{A}$ -distance without the shift constraint is a more general problem and, thus, at least as hard.

We proceed by giving algorithms for computing the *exact*  $\mathcal{A}$ -distance for distances  $\leq 3$ , and then give a general method that *estimates* the  $\mathcal{A}$ -distance  $> 3$ .

### 5.1 $\mathcal{A}$ -Equations and Exact $\mathcal{A}$ -Distance Tests

The algorithm for finding the exact value of  $\text{dist}(R, t)$  is based on testing specific distances  $d$  for feasibility. First, all possible  $\mathcal{A}$ -graph topologies that synthesize  $t$  using exactly  $d$   $\mathcal{A}$ -operations are enumerated. Then, these topologies are converted to the so called  *$\mathcal{A}$ -equations*, which relate values at the input, output, and intermediate nodes of the topology. If it is determined that the equation has a solution, then the  $\mathcal{A}$ -distance is  $\leq d$ . If we perform these tests in the order of increasing distance  $d$ , the exact  $\mathcal{A}$ -distance can be determined. Since the number of graph topologies for a given distance grows quickly [Gustafsson et al. 2002], this approach is feasible only for very small values of  $d$ . We consider  $d = 1, 2, 3$ , and only estimate the large distances.

Before we start, we list below a few useful properties of the  $\mathcal{A}$ -operation  $\mathcal{A} = \mathcal{A}^{\text{odd}}$  including constraints (as defined in Section 4.2) used in our algorithm. For other choices of the  $\mathcal{A}$ -operation the properties may not hold.

**LEMMA 5.2.** *If  $w = \mathcal{A}_p(u, v)$ , then there exists an  $\mathcal{A}$ -configuration  $p'$  such that  $u = \mathcal{A}_{p'}(w, v)$ .*

**PROOF.** Using the definition of  $\mathcal{A}_p$ :

$$w = |2^{l_1}u + (-1)^s 2^{l_2}v|2^{-r}, \quad p = (l_1, l_2, s, r).$$

Solving for  $u$  we obtain

$$\begin{aligned} u &= |2^r w + (-1)^{s'} 2^{l_2} v|2^{-l_1} = \mathcal{A}_{p'}(w, v), \\ p' &= (r, l_2, s', l_1) \text{ for a suitable } s'. \end{aligned}$$

The value of  $s'$  is 1 if  $s = 0$ , and either 0 or 1 if  $s = 1$ .  $\square$

<sup>3</sup>The SCM problem with the shift constraint is still NP-complete.

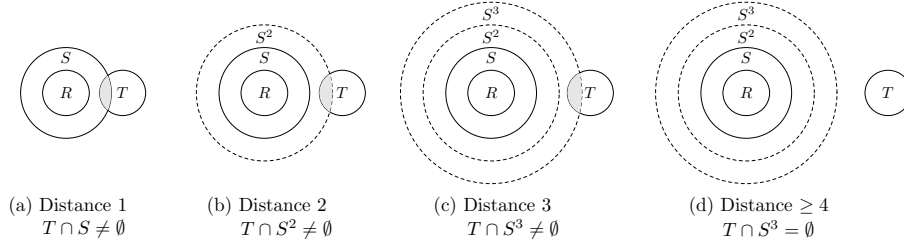


Fig. 8. Special distance cases handled by the heuristic. Solid circles denote the available sets, and dashed circles denote the sets that are not computed.

LEMMA 5.3. *If  $w = \mathcal{A}_p(u, v)$ , then there exists an  $\mathcal{A}$ -configuration  $p'$  such that  $w = \mathcal{A}_{p'}(v, u)$ .*

PROOF. Obviously, the  $\mathcal{A}$ -operation is symmetric, and it suffices to switch the left shifts to obtain  $p'$ .  $\square$

The following two corollaries follow immediately, using the definition of  $\mathcal{A}_*$  (Definition 2.5).

COROLLARY 5.4. *For any  $u$  and  $v$*

$$\mathcal{A}_*(u, v) = \mathcal{A}_*(v, u).$$

COROLLARY 5.5. *If  $w \in \mathcal{A}_*(u, v)$ , then*

$$\begin{aligned} u &\in \mathcal{A}_*(w, v) \text{ and } u \in \mathcal{A}_*(v, w), \\ v &\in \mathcal{A}_*(u, w) \text{ and } v \in \mathcal{A}_*(w, u). \end{aligned}$$

We use these properties of  $\mathcal{A}$  for solving  $\mathcal{A}$ -equations that arise in designing the distance tests.

To test for a specific  $\mathcal{A}$ -distance  $\text{dist}(R, t) = d$  we need all graph topologies with  $d$  nodes, one or more inputs, and a single output (corresponding to  $t$ ). We construct these topologies from the cost 1–3 SCM topologies from Fig. 5 in [Gustafsson et al. 2002] by splitting the single input node into multiple input nodes.

The tests for  $\text{dist}(R, t)$  proceed by assigning a value  $r_i \in R$  to each input node, assigning  $t$  to the output node, and solving for all possible values at the first successor node. If any of those values do exist in  $S$ , then the test succeeds. We consider the distances  $d = 1, 2, 3$  separately next.

## 5.2 Distance-1 Tests

Fig. 8(a) displays the case of a distance-1 target,  $\text{dist}(R, t) = 1$ . For distance-1 there is only one possible topology, shown in Fig. 9(a).

Although distance-1 targets can be detected with an  $\mathcal{A}$ -equation based test, it is not necessary. The optimal part constructs the entire  $S$  (the set of distance-1 constants) and immediately synthesizes all distance-1 targets, i.e., the intersection  $S \cap T$ .

Assuming that  $S$  is sorted, the runtime of the test is dominated by set intersection, which can be done in  $O(|T| \log |S|) = O(n \log(n^2 b^3)) = O(n \log(nb))$  time.

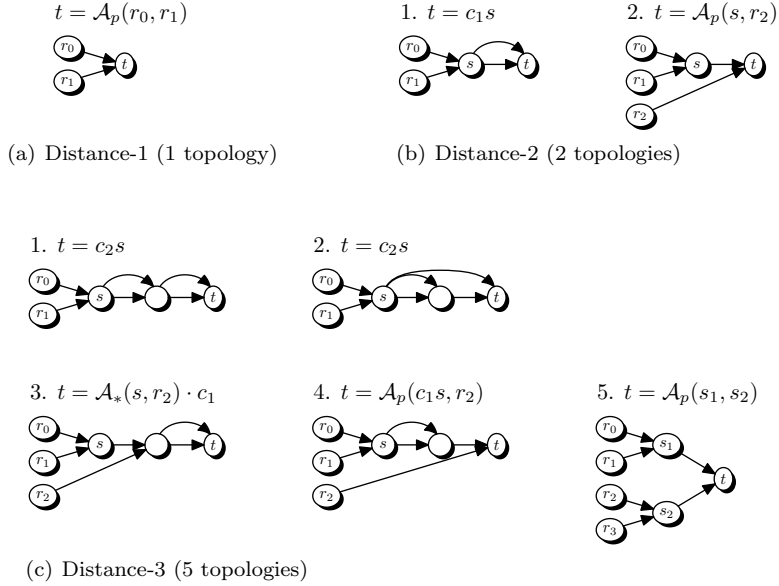


Fig. 9. Graph topologies for exact distance tests.

### 5.3 Distance-2 Tests

Fig. 8(b) shows the scenario where distance-2 targets exist. A distance-2 constant can be realized with two possible topologies, shown in Fig. 9(b). Below we construct an  $\mathcal{A}$ -equation for each of the cases.

**Case 1.** The subgraph from  $s$  to  $t$  has a single input and a single output, and hence is a multiplier block with one  $\mathcal{A}$ -operation, which multiplies by a complexity-1 constant. Thus  $t$  can be expressed as  $t = c_1 s$ , where  $c_1 \in \mathcal{C}_1$ . Alternatively,  $s = \frac{t}{c_1}$ , which has a solution iff

$$\frac{t}{\mathcal{C}_1} \cap S \neq \emptyset.$$

Since  $|\mathcal{C}_1| = O(b)$ , and intersection with  $S$  (assumed to be sorted) gives another factor of  $O(\log |S|) = O(\log(nb))$ , this test requires  $O(b \log(nb))$  time.

**Case 2.** Given  $t = \mathcal{A}_p(s, r_2)$ , we obtain  $s = \mathcal{A}_{p'}(t, r_2)$  using Lemma 5.2, which has a solution iff

$$\mathcal{A}_*(t, R) \cap S \neq \emptyset.$$

$\mathcal{A}_*$  has  $O(b)$  elements (and takes the same amount of time to compute) for each pair of inputs. Thus, for each  $O(nb)$  elements in  $R$  we have to perform  $O(b)$  operations. Intersection with (sorted)  $S$  gives another factor of  $O(\log |S|) = O(\log(nb))$ , and the total time for this test is  $O(nb^2 \log(nb))$ .

### 5.4 Distance-3 Tests

Fig. 8(c) shows the scenario where distance-3 targets exist. A distance-3 constant can be realized with five possible topologies, shown in Fig. 9(c).

[Gustafsson et al. 2002] presents all graphs in the so-called *reduced* form, which allows vertices to have an in-degree larger than 2. For topologies 2, 3, 4, and 5, we have chosen to split these vertices into binary vertices. It is shown by the authors that different splittings have equivalent possible outputs at the output vertex. We have chosen splittings that minimize the runtime of the distance test.

**Cases 1 and 2.** In both of these cases, the subgraph from  $s$  to  $t$  has a single input and a single output, and hence is a multiplier block, which multiplies by a complexity-2 constant. Cases 1 and 2 consider different complexity-2 constants, but both can be covered, if the target value is written as  $t = c_2 s$ , where  $c_2 \in \mathcal{C}_2$ . Alternatively,  $s = \frac{t}{c_2}$ , which has a solution iff

$$\frac{t}{c_2} \cap S \neq \emptyset.$$

There are  $O(b^2)$  complexity-2 constants (see Table II). Test membership in  $S$  can be done in  $O(\log(nb))$  operations, so this test requires  $O(b^2 \log(nb))$  time.

**Case 3.** The  $\mathcal{A}$ -equation can be rewritten as  $\mathcal{A}_p(s, r_2) = \frac{t}{c_1}$ . Using Lemma 5.2 we get  $\mathcal{A}_{p'}\left(\frac{t}{c_1}, r_2\right) = s$ , which has a solution iff

$$\mathcal{A}_*\left(\frac{t}{c_1}, R\right) \cap S \neq \emptyset.$$

Since  $|\mathcal{C}_1| = O(b)$ , the time for this test is  $O(b \cdot nb \cdot b \cdot \log(nb)) = O(nb^3 \log(nb))$ .

**Case 4.** Given  $t = \mathcal{A}_p(c_1 s, r_2)$  and using Lemma 5.2, we obtain  $c_1 s = \mathcal{A}_{p'}(t, r_2)$ , or, alternatively,  $s = \frac{\mathcal{A}_{p'}(t, r_2)}{c_1}$ , which has a solution iff

$$\frac{\mathcal{A}_*(R, t)}{c_1} \cap S \neq \emptyset.$$

This test is similar to the second distance-2 test, but for each element of  $\mathcal{A}_*(R, t)$  it has to go through all elements of  $\mathcal{C}_1$ . Since there are  $O(b)$  constants in  $\mathcal{C}_1$ , the requires time is  $O(b \cdot nb^2 \log(nb)) = O(nb^3 \log(nb))$ .

**Case 5.** Given  $t = \mathcal{A}_p(s_1, s_2)$  and using Lemma 5.2, we obtain  $s_2 = \mathcal{A}_{p'}(s_1, t)$ , which has a solution iff

$$\mathcal{A}_*(S, t) \cap S \neq \emptyset.$$

For each  $s_1 \in S$ , we have to perform  $O(b \log(nb))$  operations. Thus, the time for this test is  $O(|S| \cdot b \log(nb)) = O(n^2 b^4 \log(nb))$ .

## 5.5 Summary of Exact Tests

Table III shows the set intersections that need to be computed for each of the exact  $\mathcal{A}$ -distance tests, and the corresponding asymptotic runtime per each tested target. In all cases, the runtime per target is equal to the worst case set size times  $\log(nb)$  overhead for the intersection with  $S$ , assuming that  $S$  is sorted.

The tests above yield the exact value of  $\text{dist}(R, t)$ . Adding a single element to  $R$  can decrease the  $\mathcal{A}$ -distance by at most 1; this implies that  $\text{dist}(R + s, t)$ , needed for calculating  $\bar{\mathbf{B}}(R, s, t)$ , does not have to be computed at all.

All given tests compute the set, call it  $X$ , of possible values at the successor node  $s$ , and then check whether  $X \cap S \neq \emptyset$ . For all  $s \in X \cap S$  it holds that

Table III. Set intersections computed for exact distance tests and computation time per target.

Distance	Case	Formula	Time per target
1	-	$t \cap S$	$O(\log(nb))$
2	1	$\mathcal{A}_*(R, t) \cap S$	$O(nb^2 \log(nb))$
	2	$\frac{t}{c_1} \cap S$	$O(b \log(nb))$
3	1,2	$\frac{t}{c_2} \cap S$	$O(b^2 \log(nb))$
	3	$\mathcal{A}_*\left(\frac{t}{c_1}, R\right) \cap S$	$O(nb^3 \log(nb))$
	4	$\frac{\mathcal{A}_*(R, t)}{c_1} \cap S$	$O(nb^3 \log(nb))$
	5	$\mathcal{A}_*(S, t) \cap S$	$O(n^2 b^4 \log(nb))$

$\text{dist}(R + s, t) = \text{dist}(R, t) - 1$ .

The algorithm requires an admissible distance function (Section 4.4). For  $\mathcal{A}$ -distances up to 3, we use the exact distance function which is admissible. For larger distances we estimate the  $\mathcal{A}$ -distance, which is explained next.

### 5.6 Estimation (For Distance-4 and Higher)

Fig. 8(d) shows the scenario where no targets of distance 3 or lower exist. We do not use an exact distance computation in this case. However, the exact test is still feasible, and could be designed with the method shown above from the 15 possible distance-4 topologies.

Our approach estimates distances of 4 or higher using several estimators, each of which overestimates the exact distance. The final estimated distance is then the minimum of these overestimates.

Recall that, for the weighted benefit  $\bar{B}(R, s, t)$ , we need both  $\text{dist}(R, t)$  and  $\text{dist}(R + s, t)$ . For the exact tests, described above,  $\text{dist}(R, t)$  is computed, and  $\text{dist}(R + s, t)$  for each successor is obtained as a side effect. For the estimation, the converse is true. The distances  $\text{dist}(R, t)$  are obtained from  $\text{dist}(R + s, t)$  and then cached, as explained below.

Initially, for targets  $t$  which are not covered by exact distance tests, the cached value of  $\text{dist}(R, t)$  is set to the largest possible distance value. Each time the benefit function has to be computed, and the exact tests do not apply,  $\text{dist}(R, t)$  is obtained from the cache, and  $\text{dist}(R + s, t)$  is estimated using the method given here. If a particular successor  $s$  is chosen to be synthesized, the computed value of  $\text{dist}(R + s, t)$  replaces  $\text{dist}(R, t)$  in the cache.

For exact tests, both the  $H_{\text{cub}}$  and  $H_{\text{maxb}}$  heuristics only need a single  $\text{dist}(R, t)$  computation for each target  $t$ . Estimation, on the other hand, is much more expensive, because it requires  $\text{dist}(R + s, t)$  for each of the  $O(|S|) = O(n^2 b^3)$  successors  $s$  for each target  $t$ .

**Estimating  $\text{dist}(R + s, t)$ .** To estimate the  $\mathcal{A}$ -distance, we try to find an answer to the following two questions: what constant  $z$  do we need to reduce the distance to a target? and, how expensive is  $z$ ?

To answer the first question, we construct solutions to the problem of synthesizing  $t$  using 2 and 3  $\mathcal{A}$ -operations. These partial solutions are graph topologies, as in the exact test cases in Fig. 9(b) and 9(c), but have one of the inputs designated

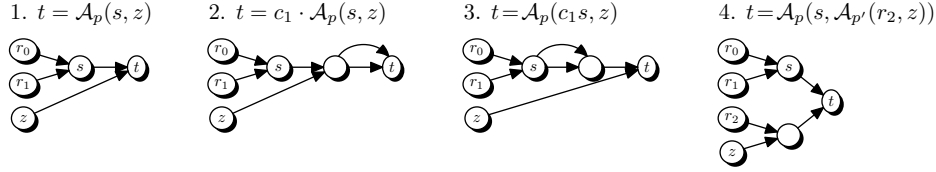


Fig. 10. Partial graph topologies for  $\text{dist}(R + s, t)$  estimation.  $z$  denotes the unsynthesized part. Estimation proceeds by determining all possible values of  $z$ , finding the “cheapest” (we use CSD cost), and adding the cost of the cheapest element to the number of operations in the topology minus 1 (since  $s$  is assumed to be available).

as an unknown and unsynthesized constant  $z$ . Since besides a successor node  $s$  we also need  $z$ , the graphs must have at least 3 inputs. Note, that there are exactly 4 topologies with 3 or more inputs in Fig. 9: distance-2 topology 2, and distance-3 topologies 3, 4, and 5. These are the graph topologies used for distance estimation, and Fig. 10 repeats them showing also the designated unknown input  $z$ . Using these topologies and Lemmas 5.2 and 5.3, as we did for exact distance tests, we can compute the set of all possible values of  $z$ .

To answer the second question we use a crude estimate for  $\text{dist}(R + s, z)$ , called a single constant *auxiliary cost measure*  $\text{Est}(z)$ , as the cost.

For a given partial topology and a given value of  $z$  an overestimated distance can be obtained as

$$\begin{aligned} \text{dist}(R + s, t) &\leq \text{dist}(R + s, z) + \text{dist}(R + s + z, t) = \\ &= \text{Est}(z) + \#\text{ops} - 1, \end{aligned}$$

where  $\#\text{ops}$  is the number of nodes (i.e., number of  $\mathcal{A}$ -operations) in the topology, and, since  $s$  is assumed to be available, 1 is subtracted.

However, since the value of  $z$  is unknown, we compute the set  $Z$  of all possible values of  $z$  for a given partial topology and values of  $s$  and  $t$ , and then use the “cheapest” (with respect to  $\text{Est}$ ) value, denoted with  $\text{Est}(Z)$ :

$$\begin{aligned} \text{dist}(R + s, t) &\leq \min_{z \in Z} \text{Est}(z) + \#\text{ops} - 1 \\ &= \text{Est}(Z) + \#\text{ops} - 1. \end{aligned} \tag{20}$$

There is a degree of freedom in choosing the auxiliary cost measure  $\text{Est}$ . However, first, we must ensure the distance function is admissible, and second, it has to be computationally efficient. For example, the constant complexity cannot be used, because the resulting distance function will not be admissible. Among the possible choices are the number of non-zero CSD bits (CSD cost) of the constant, or the number of non-zero binary bits of the constant. We have chosen the CSD cost as an auxiliary estimator:

$$\text{Est}(z) = \text{CSD-Cost}(z), \tag{21}$$

$$\text{Est}(Z) = \min_{z \in Z} \text{Est}(z) = \min_{z \in Z} \text{CSD-Cost}(z). \tag{22}$$

Computing the CSD cost takes  $O(b)$  time, and we show later that the resulting  $\mathcal{A}$ -distance estimation function is admissible.

Table IV. Sets computed for distance estimation and estimation runtime per each successor-target pair.

Case	Set $Z$	Set size	Est( $Z$ ) runtime
1	$\mathcal{A}_*(s, t)$	$O(b)$	$O(b^2)$
2	$\mathcal{A}_*(s, \frac{t}{c_1})$	$O(b^2)$	$O(b^3)$
3	$\mathcal{A}_*(C_1 s, t)$	$O(b^2)$	$O(b^3)$
4	$\mathcal{A}_*(R, \mathcal{A}_*(s, t))$	$O(nb^3)$	$O(nb^4)$

Using (20) for distance estimation, and Lemma 5.2 to compute  $Z$  using computations similar to the exact distance test, we obtain the following  $\mathcal{A}$ -distance overestimates for each case in Fig. 10:

**Case 1.**  $\text{dist}(R + s, t) \leq 1 + \text{Est}(\mathcal{A}_*(s, t)) = E_1$ .

**Case 2.**  $\text{dist}(R + s, t) \leq 2 + \text{Est}(\mathcal{A}_*(s, \frac{t}{c_1})) = E_2$ .

**Case 3.**  $\text{dist}(R + s, t) \leq 2 + \text{Est}(\mathcal{A}_*(C_1 s, t)) = E_3$ .

**Case 4.**  $\text{dist}(R + s, t) \leq 2 + \text{Est}(\mathcal{A}_*(R, \mathcal{A}_*(s, t))) = E_4$ .

The estimates  $E$  provide an upper bound and thus can be larger than  $\text{dist}(R, t)$ . When all estimates are larger than  $\text{dist}(R, t)$ , the benefit  $B(R, s, t)$  must be 0 (and not negative), and therefore, as the final value for the estimate of  $\text{dist}(R + s, t)$  we take the minimum of the four overestimates and  $\text{dist}(R, t)$ :

$$\text{dist}(R + s, t) \simeq \min(\text{dist}(R, t), E_1, E_2, E_3, E_4). \quad (23)$$

Here and further on we will use  $\simeq$  to denote that the right hand size is an estimate.

Table IV shows for each case the set  $Z$  (of  $z$  values) to compute, the worst case size of the set, and the runtime for computing  $\text{Est}(Z)$  (“Est( $Z$ ) runtime”) for each successor-target pair assuming that computing  $\text{Est}(z)$  takes  $O(b)$  time.

In our actual implementation of Algorithm 2, the fourth estimator is not used, since it is the most expensive and improves the results only insignificantly. Further, the sets required for the first three estimators do not change between iterations, and thus  $\text{Est}(Z)$  can be computed once for each successor-target pair.

### 5.7 Estimation Admissibility

We mentioned before already that the exact distance function is admissible.

The distance estimate using (23) obviously satisfies the admissibility conditions 1–4 in Definition 4.1. However, it is not as obvious that condition 5 in this definition is satisfied. We prove this in the following theorem.

Recall, that the estimate for  $\text{dist}(R, t)$  is obtained from the estimate for  $\text{dist}(R + s, t)$ , when  $s$  is chosen to be synthesized. Initially, all cached estimates are assumed to be infinite. Therefore, at the first iteration we are guaranteed that the obtained estimate  $d$  satisfies  $d < \text{dist}(R, t)$ . The theorem below proves that this will continue to be the case for the following iterations, as long as the given estimators are used.

**THEOREM 5.6.** *Let  $R, S, T$  be the ready set, the successor set, and the target set, respectively. Let  $s \in S$ , and  $t \in T$ , and let  $\text{dist}(R + s, t) \simeq d$  be the distance estimate obtained from (23). Further, assume that  $d < \text{dist}(R, t)$ , and thus  $B(R, s, t) > 0$ . Then, at the next iteration, there exists  $\bar{s} \in S_{\text{new}}$  (the new successor set of  $R_{\text{new}} = R + s$ ), such that  $\text{dist}(R + s + \bar{s}, t) = d - 1$ , i.e.,  $B(R + s, \bar{s}, t) > 0$ .*

PROOF. There are 4 cases to consider:  $d = E_1$ ,  $d = E_2$ ,  $d = E_3$ , and  $d = E_4$ . We will show only the case  $d = E_1$ ; the proofs for the other cases are analogous.

According to the case 1 estimator  $\mathcal{A}$ -equation,

$$t \in \mathcal{A}_*(s, z).$$

Since  $d \geq 3$ ,  $Est(z) \geq 2$ , i.e. the CSD cost of  $z$  is at least 2. If one non-zero CSD bit is removed from  $z$ , we get a new constant  $\bar{z}$  with a CSD cost that is reduced by 1, and  $z$  can be written as

$$z = \mathcal{A}_p(1, \bar{z}) \quad \text{so that} \quad Est(\bar{z}) = Est(z) - 1.$$

If this expression for  $z$  is substituted into the original  $\mathcal{A}$ -equation, we obtain

$$t \in \mathcal{A}_*(s, \mathcal{A}_p(1, \bar{z})).$$

Using the definition of  $\mathcal{A}^{\text{odd}}$  it can be easily shown that there exists an  $\mathcal{A}$ -configuration  $p'$  such that

$$t \in \mathcal{A}_*(\mathcal{A}_{p'}(s, 1), \bar{z}).$$

However  $\mathcal{A}_{p'}(s, 1) \subset S_{\text{new}}$ , and hence

$$t \in \mathcal{A}_*(S_{\text{new}}, \bar{z}).$$

Therefore there exists  $\bar{s} \in S_{\text{new}}$  with  $t \in \mathcal{A}_*(\bar{s}, \bar{z})$ , and using the case 1 overestimate  $\text{dist}((R + s) + \bar{s}, t) \leq 1 + Est(\bar{z}) = 1 + Est(z) - 1 = d - 1$ .  $\square$

In the proof, we assumed that an estimate  $d$  was obtained using  $E_1$ , and found that the value of again  $E_1$  will necessarily decrease. For the other estimates  $E_2$ – $E_4$  of  $d$ , however, at the next iteration only a different estimate might decrease.

We have experimented with more expensive estimators. Often, the proof does not go through without an additional and even more expensive estimator that guarantees admissibility.

## 6. RUNTIME ANALYSIS AND EXPERIMENTAL EVALUATION

In this section we analyze the runtime of the new algorithm (Algorithm 2) and its performance in terms of the average number of  $\mathcal{A}$ -operations (add/subtract operations) in the synthesized solutions. We compare to the best performing algorithms from the literature. We also provide some measured sample runtimes of the new algorithm. As before, we will use  $b$  to denote the constant bitwidth, and  $n = |T|$  to denote the number of constants in the target set.

### 6.1 Algorithms Evaluated

We provide some details on the actual implementations of the algorithms used in our benchmarks. In some cases, we reimplemented and improved the published algorithms for fair comparison.

**New algorithm with heuristic  $H_{\text{cub}}$ .** We have implemented our Algorithm 2 in C++, and tried both the  $H_{\text{cub}}$  and the  $H_{\text{maxb}}$  heuristic from Section 4.3. Both heuristics have the same computation cost, but  $H_{\text{cub}}$  consistently found better solutions. Thus, we present the results of  $H_{\text{cub}}$  only, and for convenience the algorithm is simply abbreviated as  $H_{\text{cub}}$ .

Table V. Asymptotic runtime summary.

Algorithm	Runtime	$b$	$n =  T $
Optimal [Gustafsson et al. 2002]	$\Omega(2^b)$	$\leq 19$	1
RAG-n [Dempster and Macleod 1995]	$O(n^2 b^3 \log(nb))$	$\leq 19$	$\geq 2$
BHM [Dempster and Macleod 1995]	$O(n^3 b^4)$	any	any
Lefèvre [Lefèvre 2001]	$O(n^3 b^3)$	any	any
$H_{\text{cub}}$	$O(n^4 b^5 \log(nb) + n^3 b^6)$	any	any
$H_{\text{cub}}$ (distance-2 tests only)	$O(n^3 b^5)$	any	any

**Optimal SCM.** This method performs exhaustive search over all possible graph topologies to find optimal single constant decompositions. It was originally described in [Dempster and Macleod 1994], and later improved in [Gustafsson et al. 2002] to handle constants up to 19 bits. We have reimplemented the algorithm in C++ and cross-checked the generated constant complexities with the authors.

**RAG-n.** Described in [Dempster and Macleod 1995], and discussed in Section 3, RAG-n is currently the best published MCM algorithm that we are aware of. The authors have kindly provided us with their MATLAB implementation, which, however, only handled constants up to 12 bits. For a fair comparison, we have reimplemented the algorithm in C++, and generated the lookup table up to 19 bits (using [Gustafsson et al. 2002]). Further, we improved the algorithms by inserting our complete  $\mathcal{A}$ -distance test for distance 2, in which case the original implementation used a heuristic only. All RAG-n results shown were produced using this improved version.

**BHM.** Also described in [Dempster and Macleod 1995], BHM is an improved version of BHA (i.e., the “add/subtract/shift” algorithm from [Bull and Horrocks 1991]). Both BHA and BHM are described in Section 3. We have implemented this algorithm in C++ using the BHA pseudo-code given in [Bull and Horrocks 1991] and the BHM improvements from [Dempster and Macleod 1995].

**Lefèvre.** Described in [Lefèvre 2001], this is one of the newer common subexpression elimination based MCM algorithms. We did not discuss the details of the algorithm in this paper, since it is not graph-based. The algorithm uses more sophisticated methods for identifying common subexpressions, but otherwise is similar to [Pasko et al. 1999]. The author has kindly provided us with his implementation in Perl.

Note, that we implemented every algorithm except the CSE-based from Lefèvre. This was facilitated by the general framework presented in this paper, which enables considerable code reuse. All four algorithms (ours, optimal SCM, RAG-n, and BHM) require only about 1900 lines of C++ code. Our package implementing these algorithms is available at [Spiral website 2005].

## 6.2 Asymptotic Runtime Analysis

Table V summarizes asymptotic bounds for the worst case runtimes of the different MCM algorithms. Below we describe how they were obtained.

**$H_{\text{cub}}$ .** To derive the worst case runtime of the new algorithm, we use the worst case bounds from Sections 4.5 and 5.

The algorithm executes three conceptually separate parts at each iteration: the

Table VI. Runtime breakdown for the new algorithm.

	Per-iteration runtime	Total runtime
$S$ computation	-	$O(n^2b^3 \log(nb))$
Optimal part	$O(n \log(nb))$	$O(n^2b \log(nb))$
Heuristic part, exact distance tests	$O(n^3b^4 \log(nb))$	$O(n^4b^5 \log(nb))$
Heuristic part, distance estimation	-	$O(n^3b^6)$
<b>Total</b>	-	<b><math>O(n^4b^5 \log(nb) + n^3b^6)</math></b>

incremental successor set  $S$  construction, the optimal part, and the heuristic part. Below we show the runtime of each of these steps, and then compute the total runtime, using the  $O(nb)$  bound for the number of iterations.

*Successor set construction.* The computation of  $S$  is done in increments and is distributed across iterations. The total number of successors when the algorithm terminates is  $O(n^2b^3)$ . We assume that  $S$  is kept sorted in order to do quick set intersections, and thus the total runtime is  $O(n^2b^3 \log |S|) = O(n^2b^3 \log(nb))$ .

*Optimal part.* The only overhead of the optimal part over the computation of  $S$  is checking for targets in the new successors, i.e., the computation of  $S' \cap T$ . For  $n$  targets, the runtime per iteration is  $O(|T| \log |S|) = O(n \log(n^2b^3)) = O(n \log(nb))$ .

*Heuristic part.* At each iteration we perform a series of exact tests and, if applicable, distance estimators. Table III shows the per target runtime for the exact tests, which have to be evaluated at every iteration. Table IV shows the distance estimator runtime per successor-target pair, which do not have to be recomputed between iterations.

The most expensive exact distance test is case 5 of Table III with the runtime of  $O(n^2b^4 \log(nb))$  per target, for  $n$  targets we obtain the per iteration runtime of  $O(n^3b^4 \log(nb))$  and since the worst case number of iterations is  $O(nb)$  the total runtime is  $O(n^4b^5 \log(nb))$ .

The most expensive distance estimator that we use are cases 2 and 3 of Table IV with the runtime of  $O(b^3)$  per each successor-target pair. As discussed earlier we do not use the case 4 estimator. The sets  $Z$  computed for case 1–3 estimators do not change between iterations, and therefore the estimator values have to be computed only once per each successor-target pair. There are  $O(|T| \cdot |S|) = O(n^3b^3)$  such pairs, and thus the total runtime is  $O(n^3b^3 \cdot b^3) = O(n^3b^6)$ .

Table VI summarizes the per iteration and total runtimes for the new algorithm. As can be seen from the table, the total runtime of the algorithm is dominated by the exact distance tests and distance estimators in the heuristic part and is  $O(n^4b^5 \log(nb) + n^3b^6)$ .

If only distance-2 tests and estimators are used, the most expensive distance test is the distance-2 case 1 of Table III with the runtime of  $O(nb^2 \log(nb))$  per target, which for  $n$  targets yields the runtime of  $O(n^2b^2 \log(nb))$  per iteration.

The only available distance-2 based estimator is the case 1 of Table IV with the runtime of  $O(b^2)$  per successor-target pair, yielding the total runtime of  $O(n^3b^3 \cdot b^2) = O(n^3b^5)$ .

Table VII summarizes the per iteration and total runtimes for the new algorithm with distance-2 tests only. Note that the total runtime is now dominated by the

Table VII. Runtime breakdown for the new algorithm with distance-2 tests only.

	Per-iteration runtime	Total runtime
$S$ computation	-	$O(n^2b^3 \log(nb))$
Optimal part	$O(n \log(nb))$	$O(n^2b \log(nb))$
Heuristic part, exact distance tests	$O(n^2b^2 \log(nb))$	$O(n^3b^3 \log(nb))$
Heuristic part, distance estimation	-	$O(n^3b^5)$
<b>Total</b>	-	<b><math>O(n^3b^5)</math></b>

distance estimation and is decreased to  $O(n^3b^5)$ .

**Optimal SCM.** The optimal method for SCM performs an exhaustive search over all possible decompositions, and thus has the highest runtime among all methods. The method has to look at every constant of given bitwidth, so the runtime is  $\Omega(2^b)$ . The exact analysis was not provided in the original paper. The size of the generated lookup table is  $O(2^b)$ .

**RAG-n.** The authors of RAG-n did not provide a runtime analysis for the algorithm, so we perform the analysis here.

Since at each iteration at least one target is synthesized, the total number of iterations in the worst case is  $O(|T|) = O(n)$ . The optimal part is equivalent to  $H_{\text{cub}}$  with a runtime of  $O(n^2b^3 \log(nb))$ . When the optimal part is not applicable, the same distance-2 test as in our algorithm is invoked (which is our improvement of the original as stated in the beginning of this section). According to Section 5.1, the per iteration, per target runtime is  $O(nb^2 \log(nb))$ . At each iteration the number of targets is decreased by at least 1, thus the total runtime is

$$O((n + (n - 1) + (n - 2) + \dots) \cdot b^2 \log(nb)) = O(n^2b^2 \log(nb)).$$

When the distance-2 test fails, RAG-n uses an  $O(1)$  table lookup. Thus, the total runtime is dominated by the successor set construction in the optimal part and is  $O(n^2b^3 \log(nb))$ . This is different from the original RAG-n paper [Dempster and Macleod 1995], in which the authors observed the heuristic part to be slower. The reason is that our distance-2 test is more efficient.

**BHM.** For BHM the runtime analysis was also not available. We do it below, first obtaining a bound on the number of iterations.

The error  $\epsilon$  starts as a  $b$  bit number equal to one of the targets, and at each iteration is reduced by at least a factor of 4, i.e., by 2 bits, until it reaches 0 and a new target is selected. This gives a total of  $O(\frac{b}{2} \cdot |T|) = O(nb)$  iterations.

At each iteration the heuristic makes a pass through the entire set  $S$  according to (13). Thus, the total runtime is  $O(|S| \cdot b \cdot nb) = O(n^2b^3 \cdot nb) = O(n^3b^4)$ . It can be verified that the bound  $|S| = O(n^2b^3)$  derived for our algorithm still holds (BHM uses an almost equivalent  $\mathcal{A}$ -operation constraint, and  $|R|$  is still  $O(nb)$ ).

BHM has a higher runtime than RAG-n, while [Dempster and Macleod 1995] stated otherwise, and created a hybrid RAG-n + BHM algorithm to make it faster. The authors' implementation of the distance-2 test was suboptimal, and with the  $\mathcal{A}$ -distance computation presented in this framework RAG-n runs much faster.

**Lefèvre.** The asymptotic runtime of Lefèvre's algorithm was provided to us by the author; he also noted that the average runtime is lower.

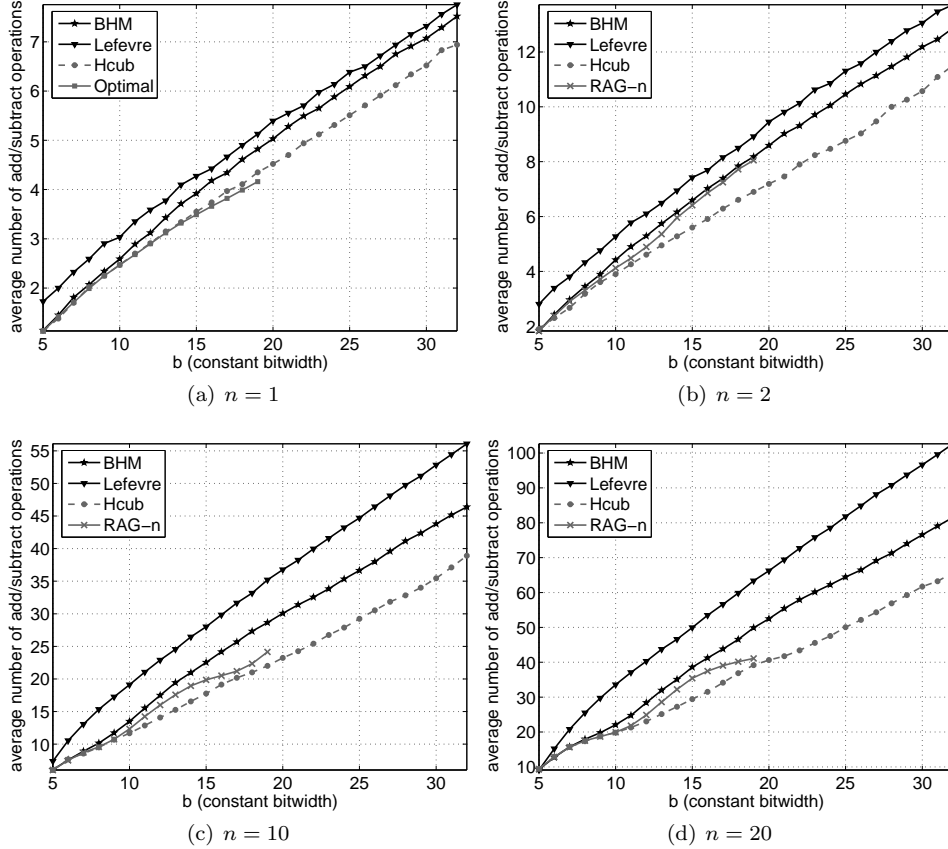


Fig. 11. Average number of  $\mathcal{A}$ -operations (adds/subtracts) vs. constant bitwidth  $b$  for a fixed number of constants  $n$ . The average is taken over 100 uniformly drawn random target sets.

### 6.3 Experimental Evaluation.

To evaluate the performance of different algorithms we ran a series of experiments on a large random sample of uniformly distributed target sets and measured the average number of adds/subtracts in MCM decompositions.

In the first experiment, we fix  $n$  (the number of constants in the target set) and vary  $b$  (the bitwidth of constants). In the second experiment, we fix  $b$  and vary  $n$ . In both experiments, we consider BHM, Lefevre, RAG- $n$ , and  $H_{\text{cub}}$ . In the third experiment, we investigate the improvement of our algorithm over RAG- $n$ . The fourth experiment shows the gain obtained by using distance-3 tests compared to only distance-2 tests in the algorithm. Finally, we show the actual runtime of our algorithm on a 3.4 GHz Pentium 4 workstation.

**Fixed number of constants.** This experiment investigates the effect of changing the constant bitwidth  $b$  on the number of  $\mathcal{A}$ -operations for  $n = 1, 2, 10$  and  $20$ . Fig. 11 shows the average number of operations (y-axis) versus  $b$  (x-axis) for a 100 uniformly drawn random target sets of size  $n$ .

For a single constant RAG- $n$  uses an optimal SCM decomposition, and therefore is not shown separately.  $H_{\text{cub}}$  is within 4% of an optimal decomposition at  $b = 19$  bits, which is the largest bitwidth handled by the optimal algorithm. As the number of bits increases,  $H_{\text{cub}}$  approaches BHM slightly, because the  $\mathcal{A}$ -distance estimation used in  $H_{\text{cub}}$  gives increasingly less accurate results. This indicates that for large bitwidths a hybrid  $H_{\text{cub}} + \text{BHM}$  based method might be beneficial. Refer to Section 3.5 for a discussion of how hybrid algorithms can be implemented.

As the number of constants increases ( $n = 2, 10, 20$ ), the performance deterioration effect of  $\mathcal{A}$ -distance estimation at large bitwidths is delayed, since the joint optimization nature of  $H_{\text{cub}}$  outweighs the drawbacks of less accurate  $\mathcal{A}$ -distance estimations.

In particular,  $H_{\text{cub}}$  performs well for 2 constants. For bitwidths larger than 14 bits it requires 10-15% fewer adds/subtracts than RAG- $n$ , which is the best of all other algorithms. Beyond 19 bits, where RAG- $n$  is not applicable,  $H_{\text{cub}}$  uses up to 17% fewer operations. Sets with 2 constants are an important case in linear signal transforms, such as the discrete Fourier transform and the various discrete cosine transforms.

For 20 constants  $H_{\text{cub}}$  produces solutions with up to 17% fewer operations than RAG- $n$ , and 25% fewer operations than BHM, where RAG- $n$  is not applicable.

**Fixed constant bitwidth.** This experiment investigates the effect of changing  $n$ , the number of constants in the target set, for different fixed bitwidths  $b = 12, 16, 19, 22, 28$  and  $32$ . Fig. 12 shows the plots of the average number of operations ( $y$ -axis) versus  $n$  ( $x$ -axis) for 200 uniformly drawn random target sets of size  $n$  for  $b \leq 22$ , and 50 sets for  $b \geq 28$ .

Again, in all cases  $H_{\text{cub}}$  outperforms all the other algorithms. For 12 bits, both RAG- $n$  and  $H_{\text{cub}}$  quickly converge to the optimal lower bound of  $n$  specified in Theorem 4.4. For 16 bits more constants are needed to converge, and starting with 19 bits, we no longer see this behavior within the considered range of  $n$ . Note, that this lower bound holds for  $n$  distinct odd constants (i.e., after right shifting), however on average the randomly drawn constant set contains slightly less than  $n$  unique odd constants after right shifting.

For 16 bits and more,  $H_{\text{cub}}$  performs clearly best, improving up to 20% over RAG- $n$  for 16 and 19 bits. Beyond that, RAG- $n$  is not applicable anymore, and the gap between  $H_{\text{cub}}$  and the next best algorithm (BHM) widens. For example, at 28 bits  $H_{\text{cub}}$  requires up to 26% less operations than BHM.

**Comparison with RAG- $n$ .** To evaluate the performance improvement relative to BHM and RAG- $n$ , we generated 100 randomly distributed constant sets, computed their MCM decompositions using RAG- $n$ , BHM and  $H_{\text{cub}}$ , and then plotted the ratio of the average  $\mathcal{A}$ -operation counts of the solutions produced by  $H_{\text{cub}}$  and BHM over those produced by RAG- $n$  versus  $n$  (Fig. 13) and versus  $b$  (Fig. 14). In the latter case, an optimal SCM decomposition was used for  $n = 1$ .

Fig. 13 fixes the bitwidth  $b = 12, 16, 19$ , and plots the ratio versus  $n$ , the number of constants. The largest improvements of  $H_{\text{cub}}$  are observed for  $b = 19$ , with up to 20% lower operations counts than RAG- $n$  at  $n = 80$ . BHM performs consistently worse than RAG- $n$ .

Fig. 14 fixes  $n = 1, 2, 10, 20$ , and plots the ratio versus  $b$ . Since, for  $n = 1$

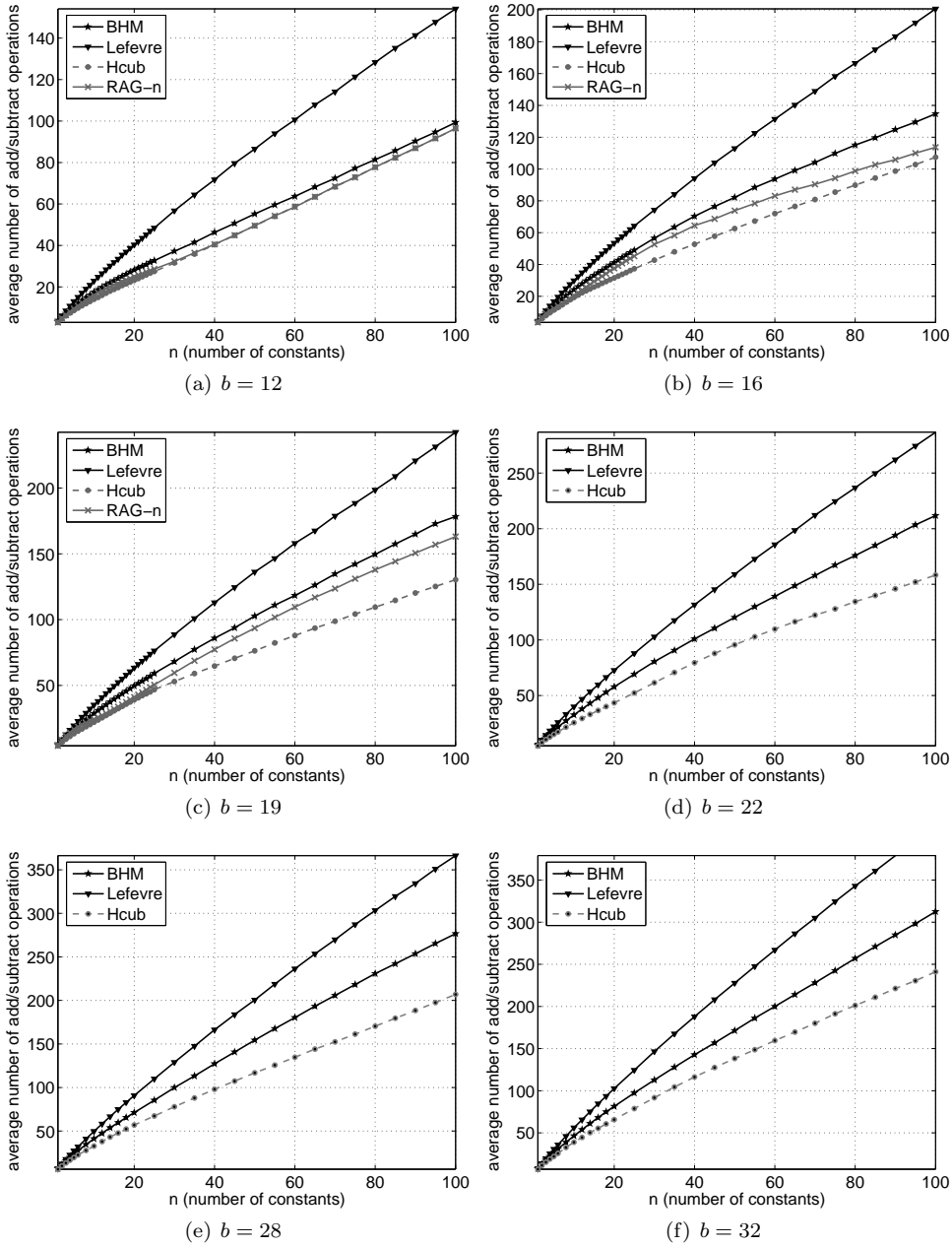


Fig. 12. Average number of  $\mathcal{A}$ -operations (adds/subtracts) vs. number of constants  $n$  for a fixed constant bitwidth  $b$ . Average is taken over 200 uniformly distributed random constant sets for  $b \leq 22$ , and over 50 sets for  $b \geq 28$ .

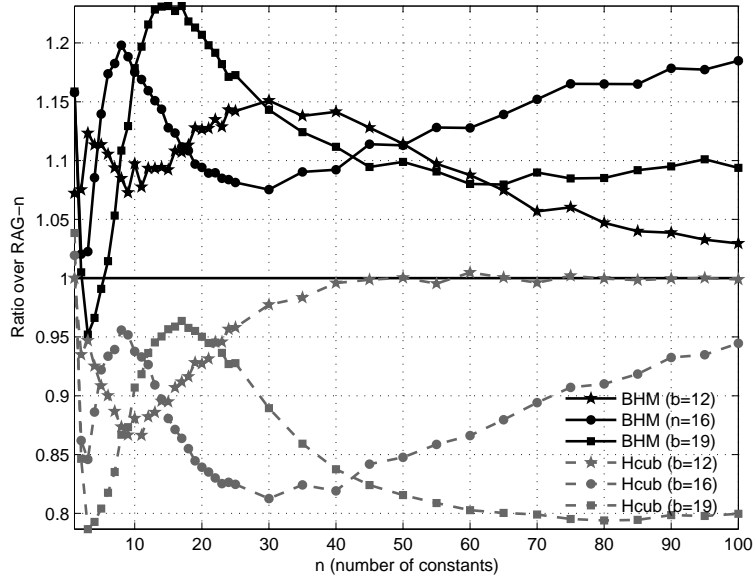


Fig. 13. Ratio of the average number of  $\mathcal{A}$ -operations in the solution produced by H<sub>cub</sub> and BHM over the average for RAG-n versus n for a fixed constant bitwidth b. The average is taken over 100 uniformly drawn random target sets.

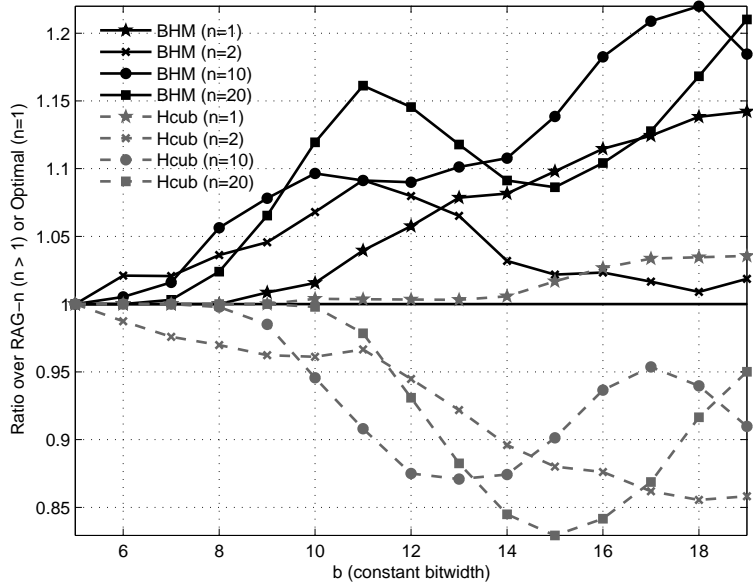


Fig. 14. Ratio of the average number of  $\mathcal{A}$ -operations in the solution produced by H<sub>cub</sub> and BHM over the average for RAG-n versus b for a fixed constant set size n. The average is taken over 100 uniformly drawn random target sets.

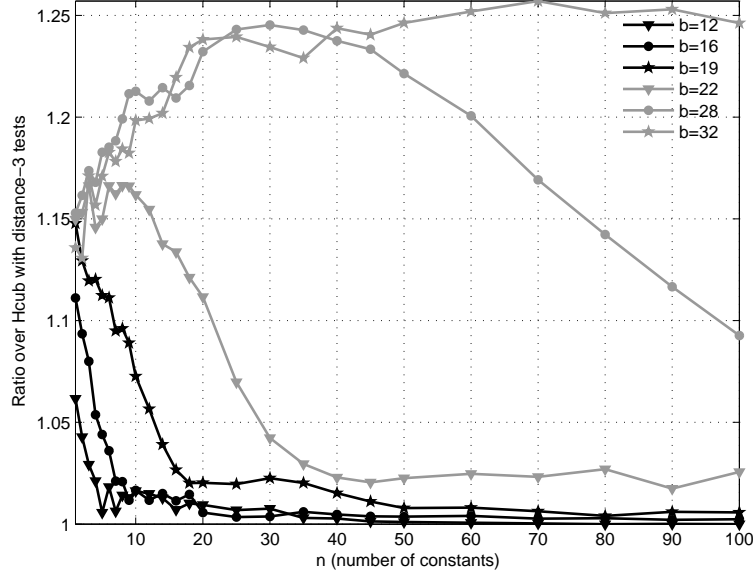


Fig. 15. Ratio of the average number of  $\mathcal{A}$ -operations in the solution produced by  $H_{\text{cub}}$  with distance-2 tests only over the average for  $H_{\text{cub}}$  with distance-3 and distance-2 tests versus  $n$  for several fixed constant bitwidths  $b$ . The averages are taken over 100 uniformly drawn random target sets.

an optimal decomposition is used, the ratio for  $n = 1$  is always greater than 1. Otherwise, the average improvement over RAG- $n$  tends to increase with  $b$ , but the ratio does not increase monotonically.

**Effect of distance-3 tests.** The runtime of  $H_{\text{cub}}$  can be reduced by removing the exact distance-3 tests and using only the estimator based on the distance-2 topology (i.e., case 1 of Table IV). As discussed earlier, this reduces the asymptotic runtime to  $O(n^3b^5)$ .

Fig. 15 shows the ratio of average  $\mathcal{A}$ -operation counts of the solutions produced by our algorithm with distance-2 tests only over our original variant with distance-2 and distance-3 tests. The averages were computed from 100 uniformly drawn random target sets.

For a single constant ( $n = 1$ )  $H_{\text{cub}}$  with distance-2 produces solutions with 6% to 15% higher operation count. For  $b \leq 19$  the largest difference occurs at  $n = 1$  and decreases to eventually less than 3% for larger constant sets. For  $b \geq 22$ , however, the ratio initially rises and drops much slower. Since the average complexity of constants goes up with  $b$ , the importance of more precise distance tests should also increase, which is confirmed by the plot. Further the plot shows that for increasing  $n$ , the difference between both tests eventually vanishes. The reason is that, intuitively, as the number of constants is increased, the precise distance value to a single target becomes less important, since the main objective is to optimize jointly for all targets.

**Runtime.** In Table VIII, we give a few average runtime examples of  $H_{\text{cub}}$  for one target set of varying size and bitwidth on a 3.4 GHz Pentium 4 EM64T Xeon  
ACM Journal Name, Vol. V, No. N, Month 20YY.

Table VIII. Average runtimes for  $H_{\text{cub}}$  in seconds. Average runtime for  $H_{\text{cub}}$  with distance-2 tests only is given in parentheses. The averages are taken over 100 experiments with random constant sets. Values less than .01 are rounded to a single decimal digit.

$b$	$n = 1$	$n = 5$	$n = 10$	$n = 20$	$n = 50$	$n = 100$
12	.002 (.0004)	.003 (.002)	.007 (.006)	.012 (.012)	.020 (.020)	.063 (.063)
16	.002 (.001)	.023 (.013)	.060 (.038)	.16 (.11)	.64 (.60)	1.1 (1.0)
19	.007 (.002)	.092 (.035)	.29 (.14)	.80 (.50)	3.6 (2.9)	13 (13)
22	.018 (.004)	.28 (.071)	1.6 (.31)	3.9 (1.4)	16 (10)	60 (45)
28	.081 (.010)	1.9 (.20)	7.0 (1.0)	32 (6.0)	540 (60)	1800 (360)
32	.20 (.018)	4.2 (.35)	20 (1.8)	95 (11)	910 (150)	8500 (1040)

workstation.

The runtimes are averages of 100 experiments, where each experiment was performed with a different random constant set. The runtimes show that the scope of parameters that should be sufficient for most applications is handled efficiently (even 8500 seconds, i.e., about 2.5 hours for 100 constants of bitwidth 32 would be acceptable within a specialized hardware design for a digital filter).

**Summary.** In all performed experiments  $H_{\text{cub}}$  outperforms all other algorithms in terms of the number of  $\mathcal{A}$ -operations of the produced solutions. We achieve up to 20% improvement over RAG- $n$ , the previously best available algorithm, while not being limited to 19 bit constants. The improvement comes at the expense of an increased runtime.

Although not shown in this paper, we measured the standard deviation of the number of add/subtract operations for different algorithms on uniformly drawn random constant sets. Interestingly, the  $H_{\text{cub}}$  algorithm had the smallest standard deviation followed by RAG- $n$  and BHM, and CSD had the highest standard deviation.

## 7. CONCLUSIONS

The main contribution of this paper is a new MCM algorithm that achieves significantly better results than previous methods as we demonstrated for the cases most relevant in practice: bitwidth  $b \leq 32$  and  $n \leq 100$  constants. However, asymptotically the new algorithm produces solutions with no known better complexity than  $O(nb)$  add/subtract operations, just like CSD and all the other algorithms.

The  $\mathcal{A}$ -distance computation and estimation framework developed in this paper should be useful for further research in this area. One direction could be to improve the heuristic, which currently combines  $\mathcal{A}$ -distances in a trivial way. Another direction would be to use our framework to optimize MCM blocks with respect to other criteria such as critical path or to also minimize for the number of shifts required.

The big question that remains unanswered is the actual asymptotic worst-case cost of SCM and MCM decompositions. However, the precise bounds remain unknown even for the simpler problem of addition chains.

## ACKNOWLEDGMENTS

The authors acknowledge the support of NSF, which funded this work through awards 0234293, 0310941, and 0325687. Further the authors would like to thank

one anonymous reviewer whose detailed comments helped to improve the quality of the paper.

## REFERENCES

- AVIZIENIS, A. 1961. Signed-digit number representation for fast parallel arithmetic. *IRE Transactions on Electronic Computers EC-10*, 389–400.
- BERNSTEIN, R. L. 1986. Multiplication by integer constants. *Software – Practice and Experience* 16, 7, 641–652.
- BULL, D. R. AND HORROCKS, D. H. 1991. Primitive operator digital filters. *IEE Proceedings G* 138, 3, 401–412.
- CAPPELLO, P. R. AND STEIGLITZ, K. 1984. Some complexity issues in digital signal processing. *IEEE Transactions on Acoustics, Speech, and Signal Processing ASSP-32*, 5, 1037–1041.
- CHEN, Y.-J., ORAINTARA, S., TRAN, T. D., AMARATUNGA, K., AND NGUYEN, T. Q. 2002. Multiplierless approximation of transforms with adder constraint. *IEEE Signal Processing Letters* 9, 11, 344–347.
- CHOO, H., MUHAMMAD, K., AND ROY, K. 2004. Complexity reduction of digital filters using shift inclusive differential coefficients. *IEEE Transactions on Signal Processing* 52, 6, 1760–1772.
- COLEMAN, J. O. 2001. Cascaded coefficient number systems lead to FIR filters of striking computational efficiency. In *Proc. International IEEE Conference in Electronics, Circuits, and Systems*.
- DEMPSTER, A. G., DEMIRSOY, S. S., AND KALE, I. 2002. Designing multiplier blocks with low logic depth. In *Proc. IEEE International Symposium on Circuits and Systems*. Vol. 5. 773–776.
- DEMPSTER, A. G. AND MACLEOD, M. D. 1994. Constant integer multiplication using minimum adders. *IEE Proceedings - Circuits, Devices and Systems* 141, 5, 407–413.
- DEMPSTER, A. G. AND MACLEOD, M. D. 1995. Use of minimum-adder multiplier blocks in FIR digital filters. *IEEE Transactions in Circuits and Systems-II: Analog and Digital Signal Processing* 42, 9, 569–577.
- DEMPSTER, A. G. AND MACLEOD, M. D. 2004. Using all signed-digit representations to design single integer multipliers using subexpression elimination. In *Proc. IEEE International Symposium on Circuits and Systems*.
- DOWNNEY, P. J., SETHI, R., AND TARJAN, R. E. 1980. Variations on the common subexpressions problem. *J. ACM* 27, 4, 758–771.
- GAREY, M. R. AND JOHNSON, D. S. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman And Company, New York.
- GUSTAFSSON, O., DEMPSTER, A. G., AND WANHAMMAR, L. 2002. Extended results for minimum-adder constant integer multipliers. In *Proc. IEEE International Symposium on Circuits and Systems*.
- HARTLEY, R. I. 1996. Subexpression sharing in filters using canonic signed digit multipliers. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing* 43, 10, 677–688.
- KANG, H.-J., KIM, H., AND PARK, I.-C. 2001. FIR filter synthesis algorithms for minimizing the delay and the number of adders. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing* 48, 8, 770–777.
- KNUTH, D. 1969. *The Art of Computer Programming: Seminumerical Algorithms*. Vol. 2. Addison-Wesley.
- LEFÈVRE, V. 2001. Multiplication by an integer constant. Tech. rep., INRIA.
- LEFÈVRE, V. 2003. Multiplication by an integer constant: Lower bounds on the code length. In *Proc. 5th Conference on Real Numbers and Computers*.
- LIANG, J. AND TRAN, T. 2001. Fast multiplierless approximations of the DCT with the lifting scheme. *IEEE Transactions on Signal Processing* 49, 12, 3032–3044.
- PASKO, R., SCHAUMONT, P., DERUDDER, V., VERNALDE, S., AND DURACKOVA, D. 1999. A new algorithm for elimination of common subexpressions. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 18, 1, 58–68.

- PÜSCHEL, M., ZELINSKI, A., AND HOE, J. C. 2004. Custom-optimized multiplierless implementations of DSP algorithms. In *Proc. Int'l Conf. Computer Aided Design (ICCAD)*. 175–182.
- SPIRAL WEBSITE. 2005. <http://www.spiral.net>.
- TUMMELTSHAMMER, P., HOE, J. C., AND PÜSCHEL, M. 2004. Multiple constant multiplication by time-multiplexed mapping of addition chains. In *Proc. Design Automation Conference*. 826–829.
- WU, H. AND HASAN, M. A. 1999. Closed-form expression for the average weight of signed-digit representations. *IEEE Transactions on Computers* 48, 848–851.

Received Month 2005; revised Month 2005; accepted Month 2005