Multiple Tunable Constant Multiplications: Algorithms and Applications

Levent Aksoy
INESC-ID
Lisboa, Portugal
levent@algos.inesc-id.pt

Eduardo Costa
Univ. Católica de Pelotas
Pelotas, Brazil
ecosta@ucpel.tche.br

Paulo Flores, José Monteiro
INESC-ID/IST TU Lisbon
Lisboa, Portugal
{pff, jcm}@inesc-id.pt

ABSTRACT

The multiple constant multiplications (MCM) problem, that is defined as finding the minimum number of addition and subtraction operations required for the multiplication of multiple constants by an input variable, has been the subject of great interest since the complexity of many digital signal processing (DSP) systems is dominated by an MCM operation. This paper introduces a variant of the MCM problem, called multiple tunable constant multiplications (MTCM) problem, where each constant is not fixed as in the MCM problem, but can be selected from a set of possible constants. We present an exact algorithm that formalizes the MTCM problem as a 0-1 integer linear programming (ILP) problem when constants are defined under a number representation. We also introduce a local search method for the MTCM problem that includes an efficient MCM algorithm. Furthermore, we show that these techniques can be used to solve various optimization problems in finite impulse response (FIR) filter design and we apply them to one of these problems. Experimental results clearly show the efficiency of the proposed methods when compared to prominent algorithms designed for the MCM problem.

1. INTRODUCTION

Multiplication of constant(s) by data input(s) is a ubiquitous operation and performance bottleneck in many DSP systems, such as fast Fourier transforms (FFTs), discrete cosine transforms (DCTs), and FIR filters, as shown in Figure 1. Since the implementation of a multiplication operation in hardware is expensive in terms of area, delay, and power dissipation and the filter coefficients are determined beforehand by the DSP algorithms, the multiplier block of an FIR filter is generally realized using addition/subtraction and shift operations. Note that shifts can be implemented using only wires that represent no hardware cost. Thus, the fundamental optimization problem, called the MCM problem [15], is defined as: given N constants to be multiplied by an input variable, find the minimum number of adders and subtracters that realize these constant multiplications.

IEEE/ACM International Conference on Computer-Aided Design (ICCAD) 2012, November 5-8, 2012, San Jose, California, USA
Copyright 2012 ACM 978-1-4503-1573-9/12/11 ...$15.00.

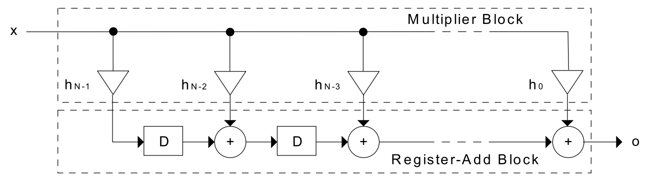


Figure 1: Transposed form FIR filter design.

Over the years many efficient algorithms have been introduced for the MCM problem [1, 2, 7, 11, 12, 13, 15]. Although they are equipped with different search techniques and look for a solution in different search spaces, their common feature is the partial product sharing which significantly reduces the number of operations. However, they assume that the constants are fixed and cannot be changed, which does not directly lend these algorithms to be applied to various optimization problems in FIR filter design [3, 9, 14]. In these problems, the aim is to find a set of filter coefficients that leads to a design including minimum number of adders/subtracters, satisfying constraint(s) related to the filter characteristics. The algorithms proposed for these problems [3, 9, 14] search a solution in intervals of floating point/integer numbers determined for each coefficient. In order to find a solution very close to the global minimum on these optimization problems, we introduce the MTCM problem where a constant to be multiplied by a variable is chosen from a set of possible constants. It is defined as: given N sets consisting of constants, $S_0, S_1, \ldots, S_{N-1}$, select one constant from each set S_i, with $0 \leq i \leq N-1$, such that the N constant multiplications require minimum number of adders/subtracters. In this paper, we present an exact common subexpression elimination (CSE) algorithm that formalizes the MTCM problem as a 0-1 ILP problem when constants are defined under a particular number representation. We also propose a local search method which incorporates an efficient graph-based (GB) MCM algorithm [15]. Furthermore, we describe how these algorithms can be adapted to handle an optimization problem in FIR filter design. Experimental results indicate that the proposed algorithms yield MCM and filter designs with significantly less number of operations compared to those obtained by MCM algorithms.

2. BACKGROUND

This section presents the background concepts, an overview on algorithms proposed for the shift-adds design of constant multiplications, and filter design optimization problems.

2.1 Number Representation

The binary representation decomposes a number in a set of additions of powers of two. The canonical signed digit (CSD) representation [12] is a signed digit system using the digit set $\{1, 0, \bar{1}\}$, where $\bar{1}$ denotes -1, and verifies the following properties: i) two nonzero digits are not adjacent; ii) the number of nonzero digits is minimum. The minimal signed digit (MSD) representation [13] is obtained by dropping the first property of CSD. Thus, a constant may have several representations under MSD, including its CSD representation, but all with a minimum number of nonzero digits.

As an example, consider 47 defined in seven bits. Its binary representation, 0101111, includes 5 nonzero digits. It is represented as $10\bar{1}000\bar{1}$ in CSD and both $10\bar{1}000\bar{1}$ and $011000\bar{1}$ denote 47 in MSD using 3 nonzero digits.

2.2 0-1 Integer Linear Programming

The 0-1 ILP problem is the minimization or the maximization of a linear cost function subject to a set of linear constraints and is generally defined as follows[1]:

$$\text{Minimize} \quad \mathbf{w}^T \cdot \mathbf{x} \tag{1}$$

$$\text{Subject to} \quad \mathbf{A} \cdot \mathbf{x} \geq \mathbf{b}, \quad \mathbf{x} \in \{0,1\}^n \tag{2}$$

In (1), w_j in $\mathbf{w}$ is an integer value associated with each of n variables x_j, $1 \leq j \leq n$, in the cost function, and in (2), $\mathbf{A} \cdot \mathbf{x} \geq \mathbf{b}$ denotes a set of m linear constraints where $\mathbf{b} \in \mathbb{Z}^m$ and $\mathbf{A} \in \mathbb{Z}^m \times \mathbb{Z}^n$.

2.3 Multiplierless Constant Multiplications

A straightforward way of realizing the constant multiplications under a shift-adds architecture, called digit-based recoding (DBR) [8], is first to define the constants under a particular number representation, and second, for the nonzero digits in the representation of the constant, is to shift the input variable according to the digit positions and add/subtract the shifted variable with respect to the digit values. As a simple example, consider the constant multiplications $39x$ and $83x$. Their decompositions in CSD are listed as follows:

$$39x = (10100\bar{1})_{CSD}x = x \ll 5 + x \ll 3 - x$$

$$83x = (101010\bar{1})_{CSD}x = x \ll 6 + x \ll 4 + x \ll 2 - x$$

which require 5 operations, as shown in Figure 2(a).

The complexity of an MCM design can be further reduced by sharing the common partial products among the constant multiplications. The algorithms designed for the MCM problem can be categorized in two classes as CSE methods [1, 11, 12, 13] and GB techniques [2, 7, 15]. The CSE algorithms generally find the most common subexpressions among the constant multiplications when all possible subexpressions are extracted from the representations of the constants. The GB methods are not limited to any particular number representation and aim to find intermediate constants that enable to realize the constant multiplications with the minimum number of operations. They consider a large number of alternative realizations of a constant and obtain significantly better solutions than the CSE algorithms.

Returning to our example, the exact CSE algorithm [1] under CSD finds $5x = (101)_{CSD}x$ as a common subexpression and obtains a solution with 4 operations (Figure 2b). The exact GB algorithm [2] finds a minimum solution with 3 operations (Figure 2c). Note that the CSE algorithm of [1] cannot realize $83x$ as the GB algorithm of [2].

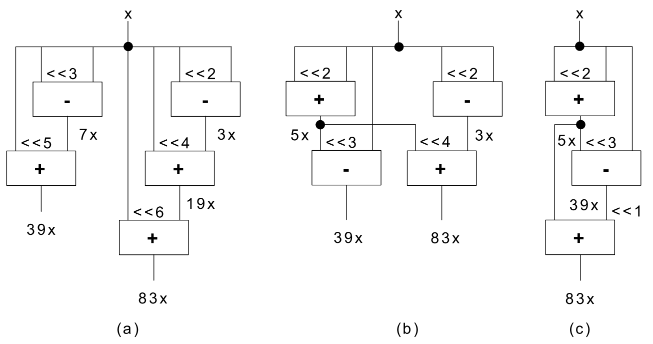


Figure 2: Shift-adds implementations of $39x$ and $83x$: (a) with the DBR technique [8]; (b) with the algorithm of [1]; (c) with the algorithm of [2].

Rather than the design of fixed constant multiplications $39x$ and $83x$, suppose that the constants can be chosen from $S_0 = \{38, 39, 40\}$ and $S_1 = \{82, 83, 84\}$, respectively. A straightforward way of finding a minimum solution requires the application of the exact GB algorithm [2] to all possible nine MCM instances[2]. In this case, a minimum solution consists of 2 operations where $40x = 5x \ll 3$ and $82x = 41x \ll 1$ are selected from S_0 and S_1, respectively and $5x$ and $41x$ are realized as $5x = x \ll 2 + x$ and $41x = 5x \ll 3 + x$, respectively. Note that the flexibility in selection of constants can reduce the required number of operations significantly.

2.4 Related Work

The exact CSE algorithms of [1, 11] formalize the MCM problem as a 0-1 ILP problem. They define the constants under a number representation and extract all possible implementations of constant multiplications from the representations of constants. The prominent CSE heuristics of [12, 13] iteratively find the most common subexpressions and replace them in constant multiplications.

The exact GB algorithms of [2] search a minimum solution in breadth-first and depth-first manners. The approximate GB algorithms of [7, 15] include two parts, optimal and heuristic. In their optimal parts, each constant, that can be implemented with a single operation, is synthesized. If there exist unimplemented elements left, then they switch to their heuristic parts where the required intermediate constants are found. The RAG-n algorithm [7] initially chooses a single unimplemented constant with the smallest single coefficient cost [6] and then, synthesizes it with a single operation including one(two) intermediate constant(s) that has(have) the smallest value. The Hcub algorithm [15] selects a single intermediate constant that yields the best cumulative benefit over all unimplemented constants for their implementation.

To the best of our knowledge, for the MTCM problem, there is only the algorithm of [9] that exhaustively considers all possible combinations of constants and applies Hcub [15] to each set of constants. However, it can only be applied to the MTCM instances consisting of small number of constants due to the exponential increase in the search space.

[1]The maximization objective can be easily converted to a minimization objective by negating the cost function. Less-than-or-equal and equality constraints are accommodated by $\mathbf{A} \cdot \mathbf{x} \leq \mathbf{b} \Leftrightarrow -\mathbf{A} \cdot \mathbf{x} \geq -\mathbf{b}$ and $\mathbf{A} \cdot \mathbf{x} = \mathbf{b} \Leftrightarrow (\mathbf{A} \cdot \mathbf{x} \geq \mathbf{b}) \wedge (\mathbf{A} \cdot \mathbf{x} \leq \mathbf{b})$, respectively.

[2]Given $S_0, S_1, \ldots, S_{N-1}$, the number of all possible MCM instances is $\prod_{i=0}^{N-1} |S_i|$. If each set has n constants, we have n^N combinations.

2.5 Filter Design Optimization Problems

The frequency response of an FIR filter is computed as:

$$H(w) = \sum_{i=0}^{N-1} h_i e^{-jwi}$$

where N is the filter length, $\mathbf{h} = \{h_0, \ldots, h_{N-1}\} \in \mathbb{R}^N$ is the actual vector of filter coefficients, and $w \in \mathbb{R}$ is the frequency in radians. As shown in [9], the absolute error in the frequency response is bounded by the 1-norm of the coefficient vector error regardless of the frequency, given as[3]:

$$||H(w)| - |H'(w)|| \leq |\sum_{i=0}^{N-1} (h_i - h'_i) e^{-jwi}|$$
$$\leq \sum_{i=0}^{N-1} |h_i - h'_i| = ||\mathbf{h} - \mathbf{h}'||_1$$

where $\mathbf{h}'$ denotes the ideal filter coefficients. Thus, the filter design optimization (FDO) problem under a tolerable error can be defined as: given the ideal set of filter coefficients $\mathbf{h}'$ and a tolerable error $\epsilon > 0$, find the actual set of filter coefficients $\mathbf{h}$ that yields an FIR filter design with minimum number of adders/subtracters, satisfying $||\mathbf{h} - \mathbf{h}'||_1 \leq \epsilon$.

In a variant of this problem [3], the zero-phase frequency response of the FIR filter and its specifications (the normalized pass-band (w_p) and stop-band (w_s) frequencies and the maximum allowed pass-band (δ_p) and stop-band (δ_s) ripples) are considered. Thus, it is defined as finding the actual set of filter coefficients that does not violate the filter characteristics and yields an FIR filter design including minimum number of adders/subtracters. Also, an important FDO problem [14] is to find the actual set of filter coefficients that minimizes the normalized peak ripple and the complexity of the FIR filter design.

In this paper, the FDO problem under a tolerable error is considered as an application for the algorithms proposed for the MTCM problem. Given the filter specifications (N, w_p, w_s, δ_p, and δ_s), we first find the ideal set of filter coefficients $\mathbf{h}'$ that respects the filter characteristics using a linear programming tool. Second, given the tolerable error $\epsilon > 0$ and taking into account the error constraint $||\mathbf{h} - \mathbf{h}'||_1 \leq \epsilon$, we compute the lower and upper bounds of each coefficient, h_i^l and h_i^u, as $h_i^l = h'_i - \epsilon$ and $h_i^u = h'_i + \epsilon$, respectively. Third, we compute the minimum quantization (Q) value that is used to convert the floating point coefficients to integers, respecting the error constraint. Thus, each set S_i associated with the coefficient h_i consists of integers between $round(h_i^l \cdot 2^Q)$ and $round(h_i^u \cdot 2^Q)$. The reason on finding the minimum Q value is that larger constants yield a greater number of operations in the multiplier block of the FIR filter and lead to a greater number of D flip-flops and larger adders in the register-add block with respect to smaller constants (Figure 1).

3. THE EXACT CSE ALGORITHM

The exact CSE algorithm consists of four main steps: i) generation of the implementations of constants, ii) construction of a Boolean network, iii) formalization of the 0-1 ILP problem, and iv) finding the minimum solution. These steps are described in detail next.

[3]1-norm of a vector $\mathbf{y}$ is computed as $||\mathbf{y}||_1 = \sum_{i=0}^{N-1} |y_i|$.

$$83 = (1000000)_{CSD} + (001010\bar{1})_{CSD} = 1 \ll 6 + 19$$
$$83 = (0010000)_{CSD} + (100010\bar{1})_{CSD} = 1 \ll 4 + 67$$
$$83 = (0000100)_{CSD} + (101000\bar{1})_{CSD} = 1 \ll 2 + 79$$
$$83 = (000000\bar{1})_{CSD} + (1010100)_{CSD} = -1 + 21 \ll 2$$
$$83 = (1010000)_{CSD} + (000010\bar{1})_{CSD} = 5 \ll 4 + 3$$
$$83 = (1000100)_{CSD} + (001000\bar{1})_{CSD} = 17 \ll 2 + 15$$
$$83 = (100000\bar{1})_{CSD} + (0010100)_{CSD} = 63 + 5 \ll 2$$

$$P_1 = \{(19), (67), (79), (21), (5, 3), (17, 15), (63, 5)\}$$

Figure 3: Implementations of 83 under CSD.

3.1 Generation of Constant Implementations

Given $S_0, S_1, \ldots, S_{N-1}$, consisting of integer constants, the algorithm first generates a set called C (initially including 0 and 1 which are labeled as implemented[4]) that will contain all the constants to be considered. Then, for each set S_i, it converts each of its elements to a positive and odd integer, adds it to C without a repetition, and labels it as unimplemented. An empty set of sets, called P, that will include all partial terms required for the realization of each element of C, is also generated. The part of the algorithm, where the partial terms are found, is as follows:

1. Take an unimplemented element from C, c_j.
2. Find an operation that implements c_j;
   (a) Convert the inputs of the operation to positive and odd integers, determine the non-repeated partial terms that are not equal to 1, and store them in an array called *Iarray*. Note that *Iarray* may be empty or it may contain a single partial term or a pair of partial terms.
   (b) If *Iarray* is empty, then make P_j empty and go to Step 5. In this case, c_j can be implemented with an operation whose inputs are 1.
   (c) If *Iarray* is not empty, check each element of P_j;
      i. If $P_j(k)$ dominates[5] *Iarray*, go to Step 3.
      ii. If *Iarray* dominates $P_j(k)$, delete $P_j(k)$.
   (d) Add *Iarray* to P_j.
3. Repeat Step 2 until all possible implementations of c_j are considered.
4. Add all the partial terms in P_j to C if they are not in C and label them as unimplemented.
5. Label c_j as implemented and repeat Step 1 until all elements in C are labeled as implemented.

Observe that the set C is augmented with partial terms that are required for the implementation of constants in later iterations of the algorithm. Note that in Step 2 of the algorithm, the implementations of a constant c_j are found by decomposing the nonzero digits in the representation of c_j into two partial terms. As an example, consider 83 defined under CSD, $(101010\bar{1})_{CSD}$. Figure 3 presents all of its possible implementations extracted from its CSD representation. Note that the duplications of implementations, such as $(5 \ll 4) + 3 = 3 + (5 \ll 4)$, are not listed in this figure.

Observe that the partial terms required for the implementation of 83 are 3, 5, 15, 17, 19, 21, 63, 67, and 79, and there are four single and three pairs of partial terms, as shown

[4]The variable that the constants are multiplied with is denoted by 1.
[5]A dominates B, if $A \cap B = A$.

in P_1. After the implementations of 83 are found, these partial terms are added to C without repetition and their implementations are found in a similar way.

Note that, in Step 2b of the algorithm, we determine that the implementation of a constant requires the minimum cost, *i.e.*, a single operation whose inputs are 1 or the shifted version of 1. In Step 2c, we avoid the repetition of a single partial term or a pair of partial terms, and remove the redundant partial terms, determined by the dominance rule [5].

The algorithm can handle the constants under binary, CSD, or MSD. In the case of MSD, we consider all the representations of the constant while finding its implementations.

3.2 The Boolean Network

The implementations of constants are represented in a Boolean network that includes only AND and OR gates. While an OR gate gathers all possible partial terms required for the implementation of a constant, an AND gate combines the elements of a pair of partial terms, indicating that both partial terms are required for the implementation of the constant. The Boolean network is constructed as follows:

1. For each constant in C, c_j, except 0 and 1, if its associated set P_j is not empty, then generate an OR gate corresponding to the constant c_j, OR_{c_j}. Otherwise, assign c_j as a primary input of the network.
2. For each P_j, that is not empty, if it includes a pair of partial terms, a and b, then generate an AND gate corresponding to this pair, $AND_{a\&b}$, and assign its output to the input of OR_{c_j}. If it includes a single partial term, *i.e.*, a primary input of the network or an OR gate output, then assign it to the input of OR_{c_j}.

The optimization variables of the 0-1 ILP problem are associated with the elements of C. In order to add them into the network, for each OR gate denoting a constant c_j, OR_{c_j}, we generate a 2-input AND gate, AND_{c_j}, where one of its inputs is the output of OR_{c_j} and the other is the optimization variable associated with c_j, OPT_{c_j}. Each primary input of the network denoting a constant is also represented with an optimization variable. Figure 4 presents the Boolean network constructed for the implementations of 83 under CSD.

Furthermore, for each set S_i, we generate an OR gate, OR_{S_i}, that gathers its elements s_{ik}, indicating that only one of them must be synthesized. For an MTCM problem without an error constraint, the inputs of this OR gate are the outputs of AND gates including an optimization variable or the primary input of the network, which are associated with the positive and odd version of s_{ik}. For an MTCM problem with an error constraint or an FDO problem under a tolerable error, the inputs of the OR gate OR_{S_i} are the outputs of 2-input AND gates generated for each element of S_i, $AND_{s_{ik}@S_i}$. One of the inputs of this AND gate is the output of an AND gate including an optimization variable or the primary input of the network, which are associated with the positive and odd version of s_{ik}. The other is a variable denoted by $VAR_{s_{ik}@S_i}$. The reason behind including this AND gate into the network is to differentiate the impact of each element in each set (s_{ik}) on the error constraint since the same constant can be in more than one set. Figure 5 depicts this part of the network for the set $S_1 = \{82, 83, 84\}$.

Observe from Figure 5 that when s_{ik} is 1 or the negative or shifted version of 1, rather than using a 2-input AND gate, the variable $VAR_{s_{ik}@S_i}$ can be assigned to $AND_{s_{ik}@S_i}$.

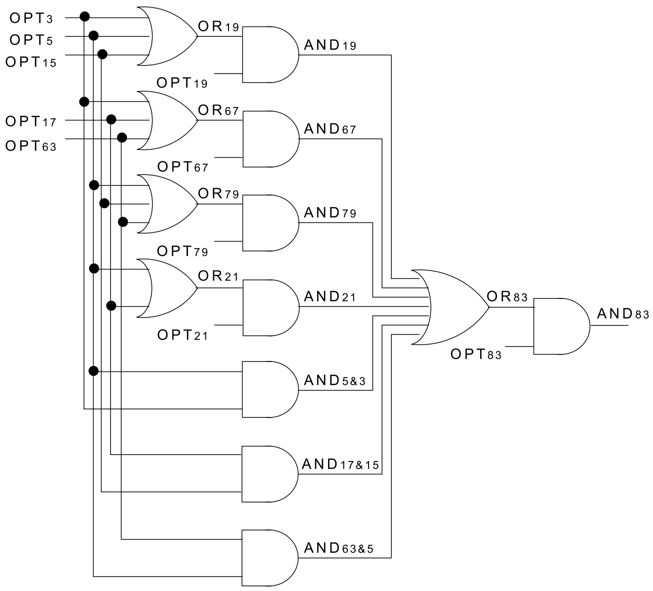


Figure 4: Implementations of 83 under CSD in a Boolean network including optimization variables.

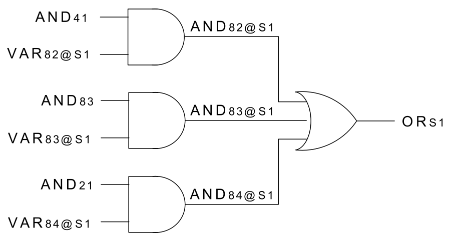


Figure 5: The part of the network for the elements of $S_1 = \{82, 83, 84\}$.

This is because the realization of the input, that the constants are multiplied with, does not require any hardware cost, implying that a variable related to the constant 1 in the 0-1 ILP problem is always assumed to be 1. The same simplification can be done when s_{ik} is equal to 0 due to the same reason. However, in an FDO problem under a tolerable error, for a filter coefficient equal to zero, no addition operation is required in the multiplier block and in the register-add block (Figure 1). In this case, we generate an optimization variable for an element of each set s_{ik} equal to 0, $OPT_{0@S_i}$, and assign it to the variable $VAR_{0@S_i}$.

3.3 0-1 ILP Formalization

The cost function of the 0-1 ILP problem is constructed as a linear function of optimization variables where the cost value of each optimization variable is 1. In an FDO problem under a tolerable error, the cost value of $OPT_{0@S_i}$, if exists, is set to -1 in order to model the savings of adders in the register-add block. Then, for each OR gate associated with a set of constants, OR_{S_i}, we generate an equality constraint which indicates that only one of its inputs must be set to 1. For our example in Figure 5, this equality constraint is $AND_{82@S_1} + AND_{83@S_1} + AND_{84@S_1} = 1$. For the rest of the logic gates in the Boolean network, we find the conjunctive normal form (CNF) formulas of each gate and express each clause of the CNF formulas as a linear inequality [4]. For example, a 2-input AND gate, $c = a \wedge b$, is translated to CNF as $(a+\bar{c})(b+\bar{c})(\bar{a}+\bar{b}+c)$ and converted to linear constraints as $a-c \geq 0$, $b-c \geq 0$, $-a-b+c \geq -1$.

For the error constraint in an FDO problem under a tolerable error, we generate a linear less-than-or-equal inequality, *i.e.*, $\sum_{i=0}^{N-1} \sum_{k=1}^{|S_i|} |s_{ik}/2^Q - h'_i| \cdot VAR_{s_{ik}@S_i} \leq \epsilon$, where Q is the quantization value and h'_i is the ideal filter coefficient in floating point, determined as described in Section 2.5. Also, $VAR_{s_{ik}@S_i}$ is the variable associated with a constant s_{ik} that can be selected from S_i (Figure 5).

3.4 Finding the Minimum Solution

A generic 0-1 ILP solver will search for the minimum value of the cost function by satisfying the constraints that represent how constants are implemented and by respecting the error constraint if available. The variables $AND_{s_{ik}@S_i}$ set to 1 will indicate which element is to be chosen from the set S_i. The optimization variables denoting constants, OPT_{c_j}, set to 1 will determine which constants are to be synthesized. Each of these constants are realized using a single operation whose inputs are 1 or one of these constants.

4. THE LOCAL SEARCH ALGORITHM

Since there are instances that the exact CSE algorithm cannot handle due to the NP-completeness of the MTCM problem and its solutions are limited to a number representation, we also introduce a local search algorithm, LS$_{\text{MTCM}}$, that is equipped with an efficient GB MCM algorithm, called Hcub [15]. Its pseudo-code is given in Figure 6.

The LS$_{\text{MTCM}}$ algorithm takes N sets consisting of integer constants as input and returns a set of operations with the fewest number of operations, denoted as the best solution (BS) in Figure 6. First, the *FixConstants* function is applied to determine an initial search point (ISP) by selecting a constant from each set. In this function, for each set S_i, we find the implementation cost of each constant of S_i in terms of the number of operations [6] and assign the one that has the minimum cost to $ISP(i)$. Then, we apply the *ComputeImpCost* function to ISP, find a solution with the fewest number of operations using Hcub, and store the set of operations to BS and its cost value in terms of the number of operations to *bcost*. In the infinite loop of LS$_{\text{MTCM}}$ (lines 3-25), since the order that we traverse on the N sets affects the solution of LS$_{\text{MTCM}}$, we first randomly generate a different ordering using the *DetermineOrder* function. Then, in an iterative loop (lines 5-18), we traverse each element of each set, $S_{O(j)}(k)$, and replace the related element of ISP with this constant. On this new set of constants, $NISP$, we first find the lower bound on the number of operations using the *ComputeLB* function [10]. If this lower bound is not greater than the cost of the best solution found so far (*bcost*), then we apply the *ComputeImpCost* function to $NISP$. If a solution with less number of operations than that of BS is obtained, we update BS and *bcost*, and replace the related element of ISP with this constant. The iterative loop terminates when no better solution is obtained during the traverse on each element of each set. In this case, we decide that a local minima is reached and we apply the *RandomFix* function to escape from this point by randomly generating a new ISP. The infinite loop terminates whenever the number of iterations in the infinite loop is 30 or the total number of Hcub runs is $30N$.[6]

[6]These values were determined empirically based on experiments.

```
LS_MTCM($S_0, S_1, \ldots, S_{N-1}$)
 1: $ISP$ = FixConstants($S_0, S_1, \ldots, S_{N-1}$)
 2: ($BS$, $bcost$) = ComputeImpCost($ISP$)
 3: while 1 do
 4:   $O$ = DetermineOrder($N$)
 5:   repeat
 6:     $w2loop = 0$
 7:     for $j = 0$ to $N-1$ do
 8:       for $k = 1$ to $|S_{O(j)}|$ do
 9:         if $S_{O(j)}(k) \neq ISP(O(j))$ then
10:           $NISP = ISP$
11:           $NISP(O(j)) = S_{O(j)}(k)$
12:           $lbcost$ = ComputeLB($NISP$)
13:           if $lbcost < bcost$ then
14:             ($AS$, $cost$) = ComputeImpCost($NISP$)
15:             if $cost < bcost$ then
16:               $w2loop = 1$, $BS = AS$, $bcost = cost$
17:               $ISP(O(j)) = S_{O(j)}(k)$
18:   until $w2loop = 0$
19:   if Terminating conditions are not met then
20:     ($ISP$) = RandomFix($ISP, S_0, S_1, \ldots, S_{N-1}$)
21:     ($AS$, $cost$) = ComputeImpCost($ISP$)
22:     if $cost < bcost$ then
23:       $BS = AS, bcost = cost$
24:   else
25:     return $BS$
```

Figure 6: The local search algorithm.

In an MTCM problem with an error constraint or an FDO problem under a tolerable error, the *FixConstants* and *RandomFix* functions are modified not to violate the error constraint while fixing and changing the value of a constant in an ISP, respectively. After a new set of constants $NISP$ is generated (line 11), we also check if its constants satisfy the error constraint. In a FDO problem under a tolerable error, the *ComputeImpCost* function is modified to compute the total number of adders/subtracters in the whole filter, *i.e.*, both in the multiplier and register-add blocks. While Hcub is applied to find a solution with the fewest number of operations in the multiplier block, the number of adders in the register-add block is computed as $N - ncz - 1$ where ncz is the number of filter coefficients equal to 0.

5. EXPERIMENTAL RESULTS

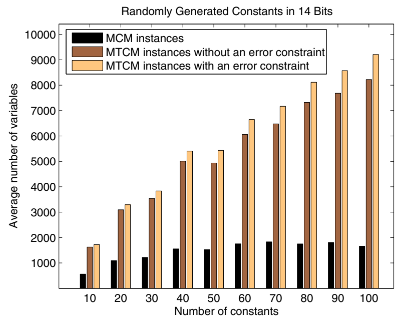
This section introduces the results of the proposed algorithms and compares them with those of prominent algorithms designed for the MCM problem.

As the first experiment set, we used randomly generated 14-bit constants. The number of constants (N) ranges from 10 to 100 and we generated 30 instances for each N, a total of 300 instances. The solutions of these MCM instances were obtained using the exact CSE [1] and Hcub [15] algorithms. Each MCM instance was converted to an MTCM instance, where, for each constant c_i, the set S_i consists of the integer values between $c_i - 2$ and $c_i + 2$. Also, from each MTCM instance, we generated an MTCM instance with an error constraint, determined as $||\mathbf{c} - \mathbf{c}'||_1 \leq 0.2N$ where $\mathbf{c}'$ is the set of constants in each original MCM instance. The proposed algorithms were applied to these MTCM instances.

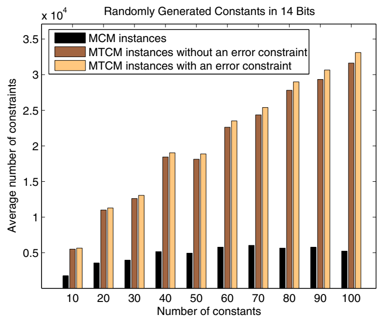
Table 1 presents the results of algorithms where *adder* denotes the average number of operations and *CPU* presents the average CPU time in seconds. Also, *gain* under the proposed exact CSE and LS$_{\text{MTCM}}$ algorithms stands for the

Table 1: **Summary of results of algorithms on randomly generated instances with 14-bit constants.**

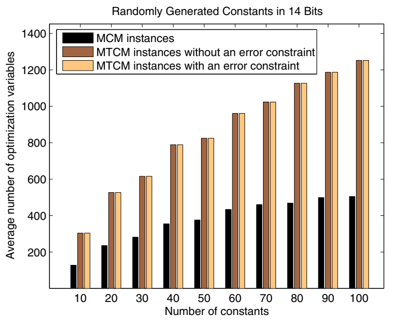
N	MCM instances				MTCM instances without an error constraint						MTCM instances with an error constraint					
	Exact CSE [1]		Hcub [15]		Exact CSE			LS_{MTCM}			Exact CSE			LS_{MTCM}		
	adder	CPU	adder	CPU	adder	gain	CPU	adder	gain	CPU	adder	gain	CPU	adder	gain	CPU
10	17.13	13.0	13.97	0.1	11.47	33.07	97.1	10.50	24.82	39.6	13.80	19.46	103.2	11.13	20.29	35.8
20	30.77	27.7	23.63	0.1	19.70	35.97	297.3	18.27	22.71	86.3	23.53	23.51	282.5	19.30	18.34	78.6
30	40.97	33.3	31.77	0.1	25.53	37.67	256.9	24.60	22.56	126.2	30.60	25.31	365.0	26.10	17.84	124.5
40	52.20	49.6	40.90	0.1	31.77	39.14	482.4	31.77	22.33	179.4	38.17	26.88	784.1	33.20	18.83	173.3
50	62.63	47.0	50.40	0.1	37.20	40.61	448.1	37.87	24.87	223.5	45.10	27.99	742.1	39.73	21.16	229.6
60	72.80	59.9	59.87	0.1	44.07	39.47	661.8	44.73	25.28	279.5	52.57	27.79	1217.3	47.00	21.49	282.6
70	83.07	65.5	69.80	0.1	48.23	41.93	590.3	51.13	26.74	339.5	58.27	29.86	1316.4	53.23	23.73	341.9
80	92.23	69.2	79.80	0.1	51.93	43.69	722.8	55.30	30.70	385.9	63.77	30.86	1566.2	58.70	26.44	390.7
90	102.23	75.3	89.67	0.1	57.57	43.69	961.7	61.93	30.93	461.2	70.03	31.50	1973.3	64.67	27.88	451.3
100	111.57	74.8	99.57	0.1	61.67	44.73	1312.1	66.80	32.91	548.6	75.97	31.91	1626.2	70.80	28.89	542.5



(a)



(b)



(c)

Figure 7: Sizes of 0-1 ILP problems generated by the exact CSE algorithms on 14-bit constants: (a) average number of variables; (b) average number of constraints; (c) average number of optimization variables.

gain in terms of the number of operations in percentage with respect to those of the exact CSE [1] and Hcub [15] algorithms, respectively. The algorithms were run on a PC with Intel Xeon at 2.33GHz and 4GB memory. In the exact CSE algorithms, the constants were defined under CSD and SCIP 2.0[7] was used as a 0-1 ILP solver.

Observe from Table 1 that the Hcub algorithm [15] obtains significantly better solutions on MCM instances in terms of the number of operations than the exact CSE algorithm [1] since it is not limited to any number representation. When each constant c_i to be multiplied by an input variable has the flexibility of being chosen from the set $\{c_i-2, c_i-1, c_i, c_i+1, c_i+2\}$, the proposed algorithms lead to significant reductions in the number of operations with respect to the algorithms of [1, 15] designed for the MCM problem. However, the runtime of the proposed algorithms is significantly increased with respect to these algorithms. For LS_{MTCM}, this is because Hcub [15] is run so many times in the local search technique. For the exact CSE algorithm, this is due to the increased size of the 0-1 ILP problem with respect to those generated in the exact CSE algorithm of [1], as shown in Figure 7. Also, observe from Figure 7(c) that the number of optimization variables in the 0-1 ILP problems generated for the MTCM problems with and without an error constraint is the same. However, Figures 7(a)-(b) show that the number of variables and constraints generated for an MTCM problem without an error constraint is slightly less than those generated for an MTCM problem with an error constraint. This is simply because when there is no error constraint, there is no need for the variables $VAR_{s_{ik}@S_i}$ and thus, the 2-input AND gates $AND_{s_{ik}@S_i}$ illustrated in Figure 5, as described in Section 3.2.

Also, when there is no error constraint in an MTCM instance, LS_{MTCM} obtains better solutions on instances with small N values than the exact CSE algorithm. This is due to the use of Hcub that considers more implementations of a constant. However, the exact CSE algorithm finds better solutions on instances with large N values than LS_{MTCM}, which is due to the greedy search technique used in LS_{MTCM}.

Moreover, when there is an error constraint in an MTCM instance, as expected, the number of operations obtained by the proposed algorithms is increased according to the results obtained on an MTCM instance without an error constraint. In this case, LS_{MTCM} finds the best solutions since the error constraint eliminates the promising subexpressions to be chosen in the exact CSE algorithm. Also, the exact CSE algorithm requires more CPU time when compared to its results on MTCM instances without an error constraint, which is due to the stringent error constraint. The LS_{MTCM} algorithm spends similar CPU time on both MTCM instances and is faster than the exact CSE algorithm.

Table 2: Filter specifications.

Filter	N	w_p	w_s	δ_p	δ_s
1	105	0.200	0.240	0.010	0.010
2	131	0.060	0.130	0.002	0.002
3	173	0.150	0.190	0.007	0.007
4	211	0.170	0.200	0.009	0.009
5	325	0.125	0.140	0.005	0.005

Table 2 presents the second experiment set, five low-pass FIR filters. The results of algorithms on these FIR filters are presented in Table 3 where Q presents the minimum quantization value and EWL denotes the maximum bitwidth of integer coefficients. Also, MA, SA, and TA stand respectively for the number of operations in the multiplier block, in the register-add block, and in the whole filter. We used three tolerable error values, namely 0.001, 0.002, and 0.005. The ideal filter coefficients ($\mathbf{h}'$), the minimum Q value, and each set of constants (S_i) were determined as described in Section 2.5. The exact CSE [1] and Hcub [15] algorithms were applied after the ideal filter coefficients were quantized

[7]The 0-1 ILP solver is available at http://scip.zib.de/

Table 3: **Summary of results of algorithms on FIR filter instances.**

Filter	ϵ	Q	EWL	MCM algorithms								MTCM algorithms							
				Exact CSE [1]				Hcub [15]				Exact CSE				LS$_{\text{MTCM}}$			
				MA	SA	TA	CPU	MA	SA	TA	CPU	MA	SA	TA	CPU	MA	SA	TA	CPU
1	0.001	15	13	50	104	154	6.7	50	104	154	0.2	44	104	148	241.2	42	104	146	175.6
	0.002	14	12	40	104	144	2.6	38	104	142	0.1	32	104	136	97.7	31	104	135	189.7
	0.005	13	11	32	104	136	0.7	32	104	136	0.1	19	104	123	51.0	19	104	123	330.9
2	0.001	15	12	46	128	174	5.0	43	128	171	0.2	44	128	172	157.5	40	128	168	113.0
	0.002	15	12	46	128	174	5.0	43	128	171	0.2	29	118	147	363.3	27	118	145	477.4
	0.005	13	10	27	118	145	0.4	27	118	145	0.1	23	112	135	25.7	25	108	133	341.8
3	0.001	16	14	58	170	228	4.7	57	170	227	0.2	43	170	213	2081.3	43	170	213	742.8
	0.002	15	13	48	170	218	1.9	47	170	217	0.2	34	168	202	453.6	35	168	203	669.3
	0.005	13	11	33	168	201	0.6	33	168	201	0.2	28	168	196	49.3	29	166	195	209.3
4	0.001	16	14	68	210	278	39.7	65	210	275	0.2	51	208	259	2857.0	53	204	257	852.1
	0.002	15	13	58	208	266	11.8	56	208	264	0.2	44	202	246	563.1	47	196	243	631.1
	0.005	14	12	44	202	246	1.3	44	202	246	0.2	31	186	217	298.2	32	186	218	763.9
5	0.001	17	15	111	322	433	19.0	109	322	431	0.2	75	320	395	2400.7	79	320	399	2313.1
	0.002	16	14	91	322	413	4.3	91	322	413	0.2	61	318	379	1430.4	65	318	383	1853.7
	0.005	14	12	56	318	374	1.0	56	318	374	0.2	51	316	367	127.5	51	316	367	283.0

to integers. In the exact CSE algorithms, the constants were defined under CSD and SCIP 2.0 was used as a 0-1 ILP solver.

Observe from Table 3 that the exact CSE [1] and Hcub [15] algorithms obtain similar results in terms of the total number of operations in the whole filter design. This also occurs on the exact CSE and LS$_{\text{MTCM}}$ algorithms designed for the MTCM problem. However, due to a large number of alternative constants in each set, they can find significantly better solutions than the algorithms of [1, 15] where the maximum gain on the total number of operations in the whole filter is 15% obtained on Filter 2 when ϵ is 0.002. Also, observe from the *SA* values that the proposed algorithms are capable of reducing the number of adders in the register-add block by assigning a 0 value to a filter coefficient, where the maximum gain is 7.9% obtained on Filter 4 when ϵ is 0.005. The maximum gain on the number of operations in the multiplier block is 40% obtained on Filter 1 when ϵ is 0.005.

However, the proposed techniques require more computational time than the MCM algorithms due to the larger search space. It is interesting to note that on some instances, such as every filter when ϵ is 0.005, the exact CSE algorithm requires less CPU time than LS$_{\text{MTCM}}$. This is because while the exact CSE algorithm solves a relatively easy problem due to the smaller Q value and less stringent error constraint, LS$_{\text{MTCM}}$ searches a solution until the terminating conditions are met although it finds the best solution earlier. We note that as the ϵ value is increased, as expected, the complexity of the filter design is decreased. This is primarily because the Q value is decreased, quantizing the coefficients to smaller integers. This is secondarily because the number of elements in each set S_i is increased, enabling the proposed techniques to consider more possible constants.

6. CONCLUSIONS

This paper introduced the MTCM problem and proposed an exact CSE algorithm and a local search method, incorporating an efficient GB MCM heuristic, for this problem. It also described the modifications required for the application of these algorithms to the FDO problem under a tolerable error. The experimental results indicated that their solutions lead to less complex MCM and filter designs with respect to those obtained by prominent MCM algorithms.

7. ACKNOWLEDGMENTS

This work was supported by FCT under INESC-ID multiannual funding through the PIDDAC Program and by the FCT project PTDC/EIA-EIA/103532/2008.

8. REFERENCES

[1] L. Aksoy, E. Costa, P. Flores, and J. Monteiro. Exact and Approximate Algorithms for the Optimization of Area and Delay in Multiple Constant Multiplications. *IEEE TCAD*, 27(6):1013–1026, 2008.

[2] L. Aksoy, E. Gunes, and P. Flores. Search Algorithms for the Multiple Constant Multiplications Problem: Exact and Approximate. *Elsevier MICPRO Journal*, 34(5):151–162, 2010.

[3] M. Aktan, A. Yurdakul, and G. Dündar. An Algorithm for the Design of Low-Power Hardware-Efficient FIR Filters. *IEEE TCAS*, 55(6):1536–1545, 2008.

[4] P. Barth. A Davis-Putnam Based Enumeration Algorithm for Linear Pseudo-Boolean Optimization. Technical report, MPII, 1995.

[5] O. Coudert. On Solving Covering Problems. In *DAC*, pages 197–202, 1996.

[6] A. Dempster and M. Macleod. Constant Integer Multiplication Using Minimum Adders. *IEE Proceedings - Circuits, Devices and Systems*, 141(5):407–413, 1994.

[7] A. Dempster and M. Macleod. Use of Minimum-Adder Multiplier Blocks in FIR Digital Filters. *IEEE TCAS-II*, 42(9):569–577, 1995.

[8] M. Ercegovac and T. Lang. *Digital Arithmetic.* Morgan Kaufmann, 2003.

[9] M. B. Gately, M. B. Yeary, and C. Y. Tang. Reduced-Hardware Digital Filter Design via Joint Quantization and Multiple Constant Multiplication Optimization. In *ICASSP*, pages 4368–4371, 2011.

[10] O. Gustafsson. Lower Bounds for Constant Multiplication Problems. *IEEE TCAS-II*, 54(11):974–978, 2007.

[11] O. Gustafsson and L. Wanhammar. ILP Modelling of the Common Subexpression Sharing Problem. In *ICECS*, pages 1171–1174, 2002.

[12] R. Hartley. Subexpression Sharing in Filters Using Canonic Signed Digit Multipliers. *IEEE TCAS-II*, 43(10):677–688, 1996.

[13] I.-C. Park and H.-J. Kang. Digital Filter Synthesis Based on Minimal Signed Digit Representation. In *DAC*, pages 468–473, 2001.

[14] D. Shi and Y. J. Yu. Design of Linear Phase FIR Filters With High Probability of Achieving Minimum Number of Adders. *IEEE TCAS*, 58(1):126–136, 2011.

[15] Y. Voronenko and M. Püschel. Multiplierless Multiple Constant Multiplication. *ACM Transactions on Algorithms*, 3(2), 2007.