A new Constraint Programming model for the Multiple Constant Multiplication

Théo Cantaloube
Ensimag, Grenoble, France
Inria, INSA Lyon, CITI, UR3720, 69621 Villeurbanne, France

Xiao Peng
LAAS-CNRS, Université de Toulouse, Toulouse, France

Christine Solnon
INSA Lyon, Inria, CITI, UR3720, 69621 Villeurbanne, France

Anastasia Volkova
Inria, INSA Lyon, CITI, UR3720, 69621 Villeurbanne, France

Abstract

The Multiple Constant Multiplication (MCM) problem arises in many applications such as, for example, digital signal processing. Given a set T of target constants, the goal of MCM is to find the most efficient way for multiplying an input number with each constant in T, where multiplications are realized through bit-shifts and additions, and where intermediate results may be shared to produce different target constants. Different metrics may be considered for evaluating the cost of a solution, and a classical objective function is to minimize the number of adders. State-of-the-art methods, based on Integer Linear Programming (ILP), suffer from numerous performance and scalability bottlenecks. In this work, we propose for the first time a Constraint Programming (CP) model for minimizing the number of adders for the MCM.

Compared to the state-of-the-art ILP approach, CP does not suffer from the curse of linearization, hence permits significantly simpler formulations of the mathematical model. In order to evaluate our CP model, we focus on a widely used benchmark extracted from a collection of digital filter designs and compare ourselves with state-of-the-art ILP and SAT models. We show that our CP approach is less efficient on some easy instances, but more efficient on hard instances. We also introduce a pseudo-polynomial time algorithm which is able to solve some instances, and show that using this algorithm during a preprocessing step improves the solution process.

2012 ACM Subject Classification Theory of computation → Constraint and logic programming

Keywords and phrases Constraint Programming, Multiple Constant Multiplication, Hardware Optimization

Digital Object Identifier 10.4230/LIPIcs...

1 Introduction

1.1 Motivation

Electronic devices with embedded computations are everywhere: in hearing-aids, smart-watches, cars, or drones, for example. The strict resource and power constraints call for highly optimized implementations. Instead of using standard data formats and generic arithmetic units, hardware designers often use code generators to implement individual or compound operators tailored to the application's requirements.

In this context, a common type of arithmetic operator is Multiplication by Multiple Constants (MCM), where an integer number is multiplied by several target constants known at the design time. The idea is that by exploiting the a priori knowledge of the constants, one can avoid costly generic multiplier circuits and design a custom optimized computational implementation for a given target constant set. MCM is used in digital signal processing for

linear digital filter evaluation [1,13,19,22,26], Discrete Transforms (Discrete Cosine Transform, Fast Fourier Transform) [15, 32], as well as for inference of Deep Neural Networks [10, 17].

Multiplications by constants are commonly done using additions/subtractions and bit-shifts: shifting a number X by k bits on the left (or the right) is equivalent to multiplying X by 2^k (or 2^{-k}). This is called a *shift-and-add* approach. Shifts are basically wiring that is free in hardware, hence designers usually only minimize the number of additions/subtractions (which we simply refer to as adders). This is a good high-level metric of the final hardware cost, though other low-level ones that count logic gates exist.

Let us first illustrate our problem on a single constant multiplication (SCM) problem. Consider the multiplication of an integer variable X by 93, written as 1011101 in binary. Hence, $93X = 2^6X + 2^4X + 2^3X + 2^2X + X$ and a naive implementation of a shift-and-add approach for $93X$ is:

$$80X = 2^6X + 2^4X; \quad 88X = 80X + 2^3X; \quad 92X = 88X + 2^2X; \quad 93X = 92X + X$$

However, this idea does not take into account the possibility of making subtractions instead of additions. Thanks to the *Canonical Signed Digit* representation of a number [18,31], we obtain that $93X = 2^7X - 2^5X - 2^2X + X$ so a second implementation of a shift-and-add approach for $93X$ is:

$$96X = 2^7X - 2^5X; \quad 92X = 96X - 2^2X; \quad 93X = 92X + X$$

Unfortunately, neither of these approaches are guaranteed to minimize the number of additions. Indeed, the optimal implementation is:

$$31X = 2^5X - X; \quad 93X = 2^1(31X) + 31X$$

You can notice that previous approaches went wrong because they forced every addition to have X as one of the inputs, therefore forbidding additions between two previously computed multiplications.

In MCM, several constants must be multiplied by the same input X. For example, consider the target set of constants $\{7, 19, 31\}$.

A naive implementation of the MCM could consist in optimizing each constant individually:

$$\begin{aligned} 7X &= 2^3X - X \\ 15X &= 2^4X - X; \quad 19X = 15X + 2^2X \\ 31X &= 2^5X - X \end{aligned}$$

However, optimizing each multiplication separately does not ensure that the total number of additions is minimized, since intermediate terms can actually be shared. Indeed, the optimal shift-and-add implementation for this target set is:

$$7X = 2^3X - X; \quad 31X = 2^5X - X; \quad 19X = 2^{-1}(7X + 31X) \tag{1}$$

Note that here we used a right-shift to divide the even value $38X$ by 2 to obtain $19X$.

Given a set of target constants, the MCM Problem aims at finding a shift-and-add implementation that produces every target constant and minimizes a given metric.

There are a lot of MCM non-exclusive variations relying on different metrics used as objective functions to minimize [2, 3, 12, 23]. In this paper, we will focus on the simplest metric: minimizing the number of adders [11, 25]. Although it does not perfectly reflect the actual implemented hardware cost, this high-level metric is a good proxy for a cost function.

It is generally accepted that MCM is an $\mathcal{NP}$-hard problem from [6]. Historically, first papers to address the MCM problem were using heuristics, most of the time based on greedy algorithms [3, 4, 8, 24]. Then Integer Linear Programming (ILP) models were implemented, first by tabulating all possible factors that may occur in an optimal solution in preprocessing computations [3, 16, 20]. However, the size of these number tables grows exponentially with the word-length of target constants, then leading to models requiring highly demanding calculations.

More general new ILP models approached the MCM problem by initializing a lower bound k to the number of target constants, and by iteratively solving the decision problem "Can the MCM be solved with exactly k adders?" until reaching the smallest k for which the answer is yes [21]. This is called an "outer loop process". Instead of solving a sequence of decision problems, authors of [11, 21] introduced a model that directly minimizes the number of adders (while initializing k to an upper bound computed with a good heuristic). Due to the linearization necessity implied by the use of ILP models, these methods suffer from a lack of expressiveness, which translates into really complicated models and also undergo numerical instabilities when the target constants become large.

In the recent paper [9] was proposed a SAT model solving the MCM problem using an outer loop process. Finally, in [5] were introduced SAT models minimizing the number of full/half adders, another lower-level metric, to solve the Single Constant Multiplication Problem. Nevertheless, in this paper, we propose for the first time an approach for the MCM problem from the perspective of Constraint Programming.

1.2 Overview of the paper

In Section 2, we go deeper into some existing models and properties of the MCM problem. In Section 3, we introduce our CP model. In Section 4, we experimentally evaluate our CP model and show that it outperforms the ILP approach of [11] and is competitive with the SAT approach of [9]. In Section 5, we consider a simpler decision problem, that aims at deciding whether their exists a solution when the number of adders is fixed to the number of constant targets. We introduce a pseudo-polynomial time algorithm for this problem, and show that half of the benchmark instances can be solved with this algorithm because the answer is yes. We also show that using this algorithm during a preprocessing step significantly improves the solution process.

2 Problem statement and properties

The first input of an MCM instance is the set T of target constants. Intermediate values used to produce these target constants are called *fundamentals*. Constants that are not used to produce other constants are called *free constants*. For example, in Eq. 1 target constants are 7, 19, and 31. While 19 is a free constant, 7 and 31 are not free as they are used to produce 19.

The second input of an MCM instance is the word-length, *i.e.*, the number of bits w used to encode fundamentals. In this paper, we fix $w = \max_{j \in [\![1,|T|]\!]} \lceil \log_2(T_j) \rceil$.

We also assume, based on experimentation, that intermediate values of fundamentals can be represented with $w + 1$ bits. Indeed, optimal solutions can often pass by a fundamental that is larger than the largest target constant. For example, in Eq. 1, the target set is representable with 5 bits but the last adder requires 6 to represent the fundamental $7 + 31 = 38 > 31 = 2^5 - 1$.

2.1 Adder modeling

For each adder a, we note $c_{a,l}$, $c_{a,r}$, and c_a the left operand, the right operand, and the result. As each operand is shifted before being added, a first possibility is to use two shift variables for representing the number of bit-shifts that must be performed on the left and right operands, respectively. As shifts may either be to the left (to multiply by positive powers of 2) or to the right (to multiply by negative powers of 2, i.e. divide), these variables are assigned to positive values when the shift is to the left, and to negative values when it is to the right.

We can reduce the domain of shift variables by considering a theorem of [8] according to which we can impose that either the shift on the left operand is positive and there is no shift on the right operand, or both shifts are negative and equal, so that they can be applied after the addition. Then we obtain [11]:

$$c_a = 2^{-s_a^r}[(-1)^{\sigma_{a,l}} 2^{s_a^l} c_{a,l} + (-1)^{\sigma_{a,r}} c_{a,r}] \tag{2}$$

where s_a^l is the left shift applied to the left input, s_a^r is the right shift applied to the sum of the left and right inputs, and $\sigma_{a,l}$ and $\sigma_{a,r}$ are signs of the left and right inputs. Eq. 2 is equivalent to:

$$2^{s_a^r} c_a = (-1)^{\sigma_{a,l}} 2^{s_a^l} c_{a,l} + (-1)^{\sigma_{a,r}} c_{a,r} \tag{3}$$

Note that $s_a^l, s_a^r \in \mathbb{N}$ instead of $\mathbb{Z}$. These notations are summed up in Fig. 1.

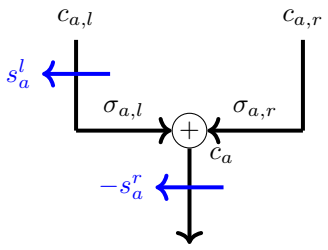


Figure 1 Optimized modeling of adders

▶ **Example 2.1.** Let us consider the shift-and-add solution Eq. 1. For the adder producing $7 = 2^3 - 1$ we have $c_a = 7$, $c_{a,l} = c_{a,r} = 1$, $s_a^l = 3$, $s_a^r = 0$, $\sigma_{a,l} = 0$ and $\sigma_{a,r} = 1$. For the adder producing $19 = 2^{-1}(31 + 7)$, we have $c_a = 19$, $c_{a,l} = 7$, $c_{a,r} = 31$, $s_a^l = 0$, $s_a^r = 1$, $\sigma_{a,l} = \sigma_{a,r} = 0$.

2.2 Properties of optimal solutions

Here is a basic property of optimal adders:

▶ **Property 1.** *In optimal implementations, every free constant is a target constant.*

Proof. A free constant is not used to produce other constants. If it is not a target constant, then we can remove it from the circuit. But the implementation is supposed to be optimal, so this is a contradiction. ◀

Keep in mind that the converse is false: a target constant may be used to compute other target constants (for example, in Eq. 1, 7 and 31 are not free constants).

Also, if we denote MCM-Adders(T, w) the minimal number of adders required to produce the target constants set T on w bits, we have:

▶ **Property 2.** *MCM-Adders*$(T, w) \geq |T|$

Proof. This is simply because we need an adder output producing each target constant. ◀

2.3 Preprocessing of target constants

The target constant set T is preprocessed before launching the solving process, as proposed in [8]. First, we replace every target constant by its absolute value, as the sign may be adjusted later. Second, we divide every target constant by 2 until it becomes odd, as a shift can be applied to retrieve the original even-valued constant. Finally, we remove 1 from the resulting set of target constants, as it can be obtained for free from the initial input. More formally, T is replaced with $\{odd(|T_j|), T_j \in T\} \setminus \{1\}$ where the operator odd is defined as:

$$odd(x) = \frac{x}{\max\limits_{n \geq 0}(gcd(x, 2^n))}$$

As a consequence, every preprocessed target constant is odd and greater than 1.

3 Constraint Programming model

In this section, we first introduce a CP model for the decision problem where the number of adders is given. Then, we show how to use this model to solve the optimization problem.

3.1 Fixed Number of adders Decision Problem

In this section, we assume that the number of adders is fixed to N.

Variables

For each adder $a \in [\![1, N]\!]$, we introduce variables to represent its operands and output:

- $c_a \in [\![1, 2^w - 1]\!]$ is the output constant;
- $c_{a,l} \in [\![1, 2^w - 1]\!]$ and $c_{a,r} \in [\![1, 2^w - 1]\!]$ are the left and right inputs.

We also introduce a variable c_0 which models the initial input and which is constrained to be equal to 1.

Variables $c_a, c_{a,l}$, and $c_{a,r}$ are related with sign values ($\sigma_{a,l}$ and $\sigma_{a,r}$) and shift values (s^l_a and s^r_a) as defined in Eq. 3. Instead of introducing sign and shift variables, as done in ILP models, we gather them in k_a, $k_{a,l}$, and $k_{a,r}$ variables, which are further gathered in c^{nsh}_a, $c^{sh,sg}_{a,l}$ and $c^{sh,sg}_{a,r}$ variables as displayed below:

$$\underbrace{\underbrace{2^{s^r_a}}_{k_a} c_a}_{c^{nsh}_a} = \underbrace{\underbrace{(-1)^{\sigma_{a,l}} 2^{s^l_a}}_{k_{a,l}} c_{a,l}}_{c^{sh,sg}_{a,l}} + \underbrace{\underbrace{(-1)^{\sigma_{a,r}}}_{k_{a,r}} c_{a,r}}_{c^{sh,sg}_{a,r}} \tag{4}$$

Hence, for each adder $a \in [\![1, N]\!]$, we introduce the following variables to store intermediate results:

- $k_a \in \{2^s \mid s \in [\![0, w]\!]\}$;
- $k_{a,l} \in \{-2^s, 2^s \mid s \in [\![0, w]\!]\}$;
- $k_{a,r} \in \{-1, 1\}$;
- $c^{nsh}_a \in [\![0, 2^{w+1}]\!]$;
- $c^{sh,sg}_{a,l} \in [\![-2^{w+1}, 2^{w+1}]\!]$ and $c^{sh,sg}_{a,r} \in [\![-2^w + 1, 2^w - 1]\!]$.

C1: $\forall\ a \in [\![1, N]\!],\ k_a \times c_a = c_a^{nsh}$
C2: $\forall\ a \in [\![1, N]\!],\ k_{a,l} \times c_{a,l} = c_{a,l}^{sh,sg}$ and $k_{a,r} \times c_{a,r} = c_{a,r}^{sh,sg}$
C3: $\forall\ a \in [\![1, N]\!],\ c_{a,l}^{sh,sg} + c_{a,r}^{sh,sg} = c_a^{nsh}$
C4: $\forall\ a \in [\![1, N]\!],\ k_{a,l} > 0$ or $k_{a,r} > 0$
C5: $\forall\ a \in [\![1, N]\!],\ k_a + k_{a,l} \neq 2$ and $k_a + k_{a,l} \neq 0$
C6: $T \subseteq \{c_a \mid a \in [\![1, N]\!]\}$
C7: $c_0 = 1$
C8: $\forall\ a \in [\![1, N-1]\!],\ \{a' \in [\![k+1, N]\!] \mid ind_{a',l} = a\ \vee\ ind_{a',r} = a\} = \emptyset \implies c_a \in T$
C9: $\forall\ a \in [\![1, N]\!],\ c_{a,l} = c_{ind_{a,l}}$ and $c_{a,r} = c_{ind_{a,r}}$
C10: allDifferent$((c_a)_{a \in [\![0,N]\!]})$
C11: $\forall\ a \in [\![1, N]\!],\ c_a \equiv 1[2]$

Figure 2 Constraints for the Decision problem, when the number of adders is fixed to N

Finally, for each adder $a \in [\![1, N]\!]$, we introduce variables for linking the left and right inputs of a with the output of another adder. To ensure that there is no cycle in this dependency relation, the domain of these variables only contains values smaller than a:

- $ind_{a,l} \in [\![0, a-1]\!]$ and $ind_{a,r} \in [\![0, a-1]\!]$

Constraints

Constraints are listed in Fig. 2.

Constraints C1 to C3 are straightforward consequences of Eq. 4.

As fundamentals cannot have negative values, we know that $k_{a,l}$ and $k_{a,r}$ cannot be both negative. This is expressed by Constraint C4.

As stated in Section 2.1, whenever there is no right shift, *i.e.*, $k_a = 1$, then there is a left shift, *i.e.*, $k_{a,l} \notin \{-1, 1\}$. This is ensured by Constraint C5.

To be a valid MCM implementation for the given target set T, adders must produce every target constant. This is realized by Constraint C6.

Constraint C7 initializes the origin input.

According to the Property P. 1, free constants are target constants. First member of the implication of Constraint C8 is equivalent to: "If adder a produces a free constant..." and second member is equivalent to "...then it is a target constant.".

Constraint C9 links the left and right inputs of each adder a with the output of adders $ind_{a,l}$ and $ind_{a,r}$, respectively.

Finally, constraints C10 and C11 ensure that the generated values are all different and odd, respectively.

Search strategy

Random exploration of the search space in the MCM problem rarely produces valid or meaningful solutions. Fixing the fundamental of the second adder before the first adder generally results in failure. In contrast, constructing the adder graph incrementally, by adding one adder at a time in a well-defined order, is more likely to yield a valid solution. Thus, we impose a Lower Bound First strategy, and the variable ordering heuristic is detailed in Fig. 3.

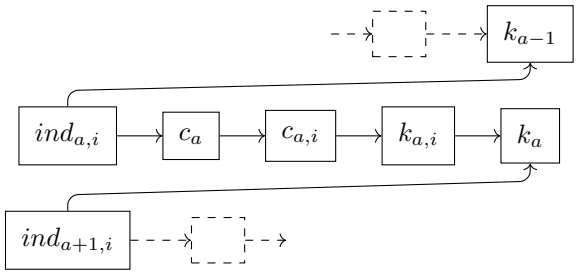


Figure 3 Branching priority for Constraint Programming modeling

3.2 Minimization Problem

To solve the MCM minimization problem, we solve a sequence of decision problems: we initialize N to $|T|$, and then we iteratively solve the model defined in Section 3.1 and increment N until we find a solution.

4 Experimental Evaluation

Our CP model has been implemented in Java with Choco-solver [30]. Experiments were conducted on a machine equipped with an Intel® Xeon® Gold 6444Y processor (16 cores, 32 threads, 45 MB L3 cache) with 256 GB RAM, running on an x86_64 architecture with support for AVX-512 and related vector instruction sets. We call this approach CP – Choco.

In addition to CP – Choco, we propose one more model using Minizinc [28]. As we heard how efficient SAT solvers (directly written in SAT) were for the MCM problem resolution, we used Minizinc with CP models in front-end, PicatSAT [33] in middle-end (to translate CP into SAT) and KisSAT in back-end. We denote as CP – PicatSAT our CP model with this approach.

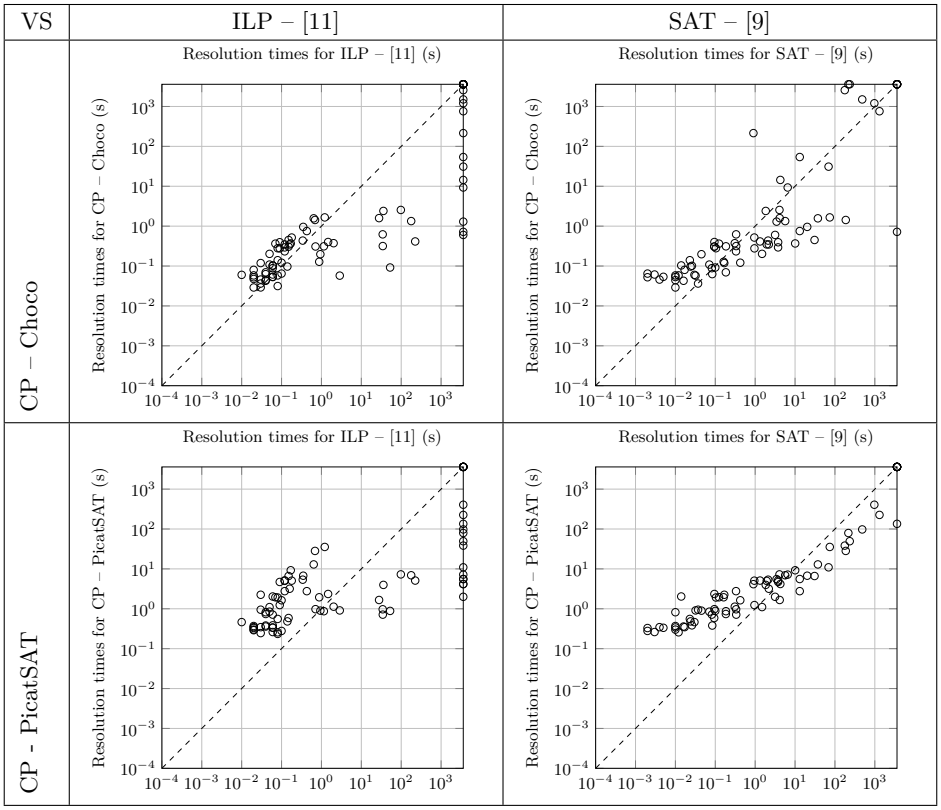
The code written for these models is available as open source[1].

The benchmark we have worked on is extracted from a collection of linear digital filters [27] that can be implemented with the MCM problem. It has been widely used in the literature and for the ILP-based approaches in particular [11, 21, 24]. It contains 83 instances with $|T|$ ranging from 1 to 51 constants and w from 4 to 19 bits.

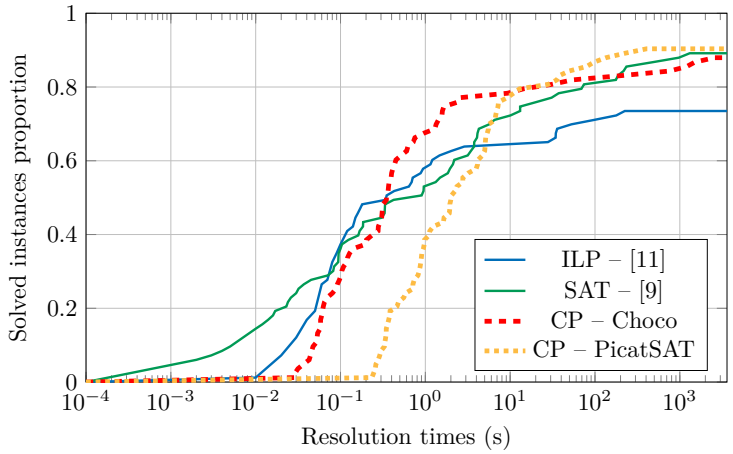
In Fig. 4(a), we compare our CP-based approach with the ILP approach of [11] and the SAT approach of [9] on a per instance basis. ILP – [11] and SAT – [9] are faster on "easy" instances, solved in less than 1s by both approaches. However, our approaches are faster on harder instances. While, in a run time limit of 3600s, CP – Choco and CP – PicatSAT are respectively able to solve 73 and 75 out of 83 instances, ILP – [11] and SAT – [9] respectively solved 61 and 74 out of 83 instances. To give more details, whenever CP – Choco or CP – PicatSAT resulted with a Time-Out, ILP – [11] resulted with a Time-out and whenever CP – PicatSAT resulted with a Time-Out, SAT – [9] resulted with a Time-out. When looking at Fig. 4(b), that displays the evolution of the cumulative percentage of solved instances with respect to time, we note that ILP – [11] and SAT – [9] are more successful than CP – Choco and CP – PicatSAT when the time limit is smaller than 0.1s, whereas SAT – [9], CP – Choco and CP – PicatSAT is more successful than ILP – [11] for longer time limits.

In addition, we can see how much better suited CP – Choco is to the combinatorial

[1] `https://gitlab.inria.fr/emeraude/mcm-cp`



(a) Comparing CP – Choco and CP – PicatSAT to ILP – [11] and SAT – [9]



(b) Distribution of the solved instances proportion through time

Figure 4 Comparative analysis of ILP and CP resolution times.

aspects of the MCM problem than ILP – [11] by comparing the number of variables in the respective models. ILP – [11] uses $2 + N(10 + |T| + 2w + (N+1)/2)$ variables whereas CP – Choco uses $11N$ variables. We can notice that unlike ILP – [11], there is no dependence on the number of target constants or on the word-length and the dependence on N is linear.

5 Preprocessing step

In this section, we study the decision problem that aims at deciding whether there exists a solution with $|T|$ adders. We show that this problem may be solved in pseudo-polynomial time, and that the solution process is improved when solving this problem in a preprocessing step.

Let us first formally define this decision problem:

Problem: T-OPT
Input: Target constant set T and word-length w
Question: Do we have MCM-Adders$(T, w) = |T|$?

Before delving into the algorithm, we first need to show how to efficiently determine if z can be produced with x and y and inputs (we denote this as $\mathsf{can}(x, y, z)$). We could do $\mathsf{can}(x, y, z) = (z \in \mathcal{A}_w(x, y))$ where $\mathcal{A}_w(x, y)$ represents the set of fundamentals written on w bits "reachable" from inputs x and y but computing $\mathcal{A}_w(x, y)$ is not optimal (the table is too large and it takes too much time to generate). Instead, we can use the operator *odd* and do:

$$\mathsf{can}(x, y, z) = \left(\exists\, (\sigma_1, \sigma_2) \in \{0, 1\}^2 \setminus (1, 1) \;\middle|\; \begin{array}{rl} & z = odd((-1)^{\sigma_1} x + (-1)^{\sigma_2} y) \\ \text{or} & (-1)^{\sigma_1} x = odd(z - (-1)^{\sigma_2} y) \\ \text{or} & (-1)^{\sigma_2} y = odd(z - (-1)^{\sigma_1} x) \end{array} \right)$$

▶ **Example 5.1.** Let us use the operator can on $T = \{7, 19, 31\}$ (cf Eq. 1). $\mathsf{can}(1, 1, 7) = true$ because $1 \times 2^3 - 1 = 7$ so $odd(7 + 1) = 1$. $\mathsf{can}(1, 1, 31) = true$ because $1 \times 2^5 - 1 = 31$ so $odd(31 + 1) = 1$. $\mathsf{can}(7, 31, 19) = true$ because $2^{-1} \times (7 + 31) = 19$ so $odd(7 + 31) = 19$.

The time complexity for computing can is logarithmic in the number of bits because of the operator *odd* [7, chapter 10].

▶ **Theorem 5.2.** *T-OPT can be solved in pseudo-polynomial time* $\mathcal{O}(log(w) \times |T|^3)$.

Proof. The pseudo-polynomial algorithm is displayed in Alg. 1. We define the depth of an adder as the length of the longest path from the initial input 1, as illustrated in Fig. 5. The idea is to process depth by depth and build R, the set of reached target constants, while Q is the queue of target constants waiting to be processed. We start from 1 and we compute $\mathsf{can}(1, 1, T_j)$, $\forall T_j \in T$. We can repeat the process until we have found all target constants, as illustrated in Fig 5. The algorithm will necessarily terminate because $|T| =$ MCM-Adders(T, w). The while and for loops on lines 3, 5 and 6 of Alg. 1 run at most $|T|$ times. Hence, the time complexity of Alg. 1 is $\mathcal{O}(log(w) \times |T|^3)$. ◀

A straightforward consequence of Th. 5.2 is that any instance (T, w) such that MCM-Adders$(T, w) = |T|$ may be solved in pseudo-polynomial time. Hence, we propose to run Alg. 1 in a preprocessing step. If the algorithm returns true, then *prec* is a solution and the instance is solved; otherwise we can set the lower bound on the number of adders to $|T| + 1$, and then solve the instance with CP. We call these new approaches CP T-OPT – Choco and CP T-OPT – PicatSAT.

Algorithm 1 Pseudo-polynomial implementation of MCM on T-OPT instances

Input: Target constant set T and word-length w
Output: $(A, prec)$ such that A is the answer for T-OPT(T, w) and if A=true, then $prec$ is a predecessor function corresponding to an MCM implementation

```
function MCM-Adders_{pseudo-poly}(T)
    Let R = {1} and W = {1} be sets of constants
    while W ≠ ∅ do
        Pop x from W
        forall z ∈ T \ R do
            forall y ∈ R \ W do
                if can(x, y, z) then
                    prec(z) ← (x, y)
                    R ← R ∪ {z}
                    W ← W ∪ {z}
                    Break
        if R = T then
            return (True, prec)
    return (False, ∅)
```

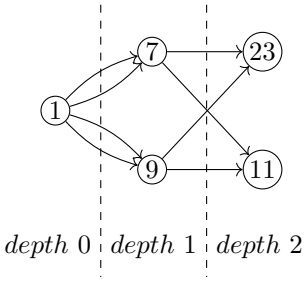
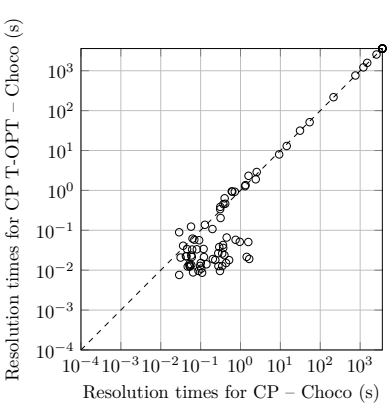


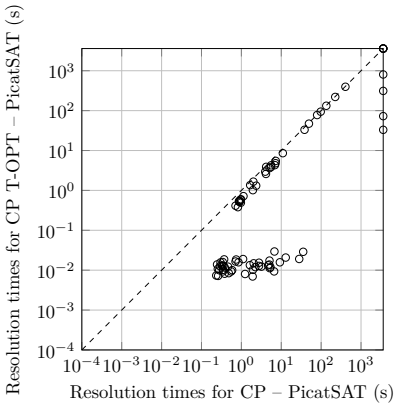
Figure 5 Illustration of Alg. 1 on the toy instance ($T = \{1, 7, 9, 11, 23\}$, $w = 5$). We have $\mathsf{can}(1, 1, 7) = \mathsf{can}(1, 1, 9) = true$ because $7 = 2^3 - 1$ and $9 = 2^3 + 1$. Hence, at the end of the first iteration of the while loop, we have $R = \{1, 7, 9\}$ and $W = \{7, 9\}$. We have $\mathsf{can}(7, 9, 11) = \mathsf{can}(7, 9, 23) = true$ because $11 = 2 \times 9 - 7$ and $23 = 2 \times 7 + 9$. Hence, at the end of the second iteration of the while loop, we have $R = \{1, 7, 9, 11, 23\}$ and $W = \{23, 11\}$. Then $R = T$ so the algorithm will return $prec = \{7 \leftarrow (1, 1), 9 \leftarrow (1, 1), 11 \leftarrow (7, 9), 23 \leftarrow (7, 9)\}$

When running Alg. 1 on our benchmark, we noticed that the answer was true for 44 out of the 83 MCM instances. In Fig. 6(a) and (b), we compare CP T-OPT – Choco with CP – Choco and CP T-OPT – PicatSAT with CP – PicatSAT. It shows us that the preprocessing step allows us to significantly reduce the solution time for many instances, those for which MCM-Adders$(T, w) = |T|$, whereas solving times are not significantly changed for the other instances (as the running time of Algo. 1 is largely negligible compared to the running time of CP). When looking at Fig. 6(c), that displays the evolution of the cumulative percentage of solved instances with respect to time, we note that CP T-OPT – Choco and CP T-OPT – PicatSAT are always better than ILP – [11] and SAT – [9]. CP T-OPT – PicatSAT is now able to solve 79 out of 83 instances of the benchmark. This improvement is a direct consequence of the preprocessing step: by avoiding unnecessary resolution of the decision

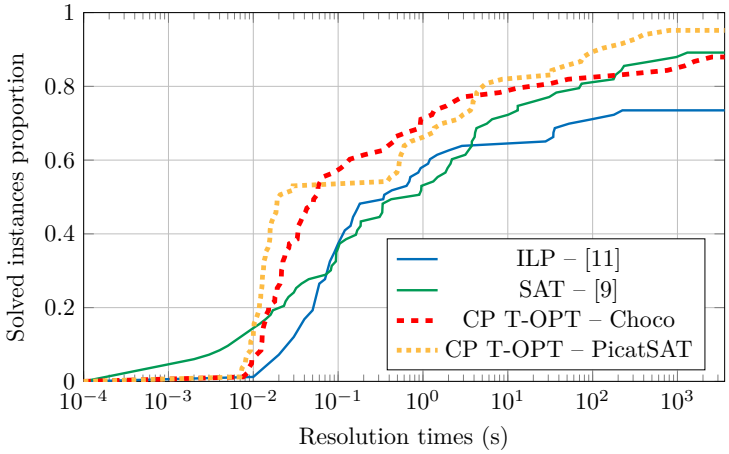
problem when the number of adders equals the number of target constants, more time remains available to solve harder instances before reaching the timeout.



(a) CP – Choco vs CP T-OPT – Choco



(b) CP – PicatSAT vs CP T-OPT – PicatSAT



(c) Distribution of the solved instances proportion through time

Figure 6 Comparative analysis of ILP – [11], SAT – [9], CP T-OPT – Choco and CP T-OPT – PicatSAT resolution times.

6 Conclusion and perspectives

In this paper, we introduced a new Constraint Programming model to solve the Multiple Constant Multiplication problem. We demonstrated that it outperforms previous ILP-based approaches and is very competitive in comparison to SAT models. We have also shown that half of the usual benchmark MCM instances do not require modeling and optimization paradigms but only a pseudo-polynomial algorithm.

Further works will focus on the improvement of the scalability with respect to the word-length (with $w > 20$ bits). Moreover, it seems that the demonstration of $\mathcal{NP}$-hardness of the MCM problem from [6] is not correct because it assumes that the MCM problem is the same as the Ensemble Computation problem proven $\mathcal{NP}$-hard in [14], which is false. Thus, a proper demonstration of the $\mathcal{NP}$-hardness of the MCM problem would be useful.

We also plan to extend our model so that it directly solves the minimization problem, instead of solving a sequence of decision problems with fixed numbers of adders. The key point is to model the fact that some adders are not used. This should allow us to improve performance of CP.

Finally, we plan to approach the MCM problem through the prism of fixed topology and decision problems. The idea is that fixing the topology of the shift-and-add implementation (the adder nodes and the arcs) facilitates the search for the corresponding shifts, signs and fundamentals. Hence, we plan to enumerate every graph topology up to a reasonable size to facilitate the optimization. In order to eliminate symmetries we plan to investigate the interest of using canonical codes based on Breadth-First Search, as proposed in [29].

References

1. Levent Aksoy, Eduardo Costa, Paulo Flores, and Jose Monteiro. Optimization of area in digital fir filters using gate-level metrics. In *2007 44th ACM/IEEE Design Automation Conference*, pages 420–423, 2007. doi:10.1109/DAC.2007.375200.
2. Levent Aksoy, Eduardo Costa, Paulo Flores, and José Monteiro. Optimization of gate-level area in high throughput multiple constant multiplications. In *2011 20th European Conference on Circuit Theory and Design (ECCTD)*, pages 588–591, 2011. doi:10.1109/ECCTD.2011.6043602.
3. Levent Aksoy, Eduardo da Costa, Paulo Flores, and JosÉ Monteiro. Exact and approximate algorithms for the optimization of area and delay in multiple constant multiplications. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(6):1013–1026, 2008. doi:10.1109/TCAD.2008.923242.
4. Robert Bernstein. Multiplication by integer constants. *Software: Practice and Experience*, 16(7):641–652, July 1986. doi:10.1002/spe.4380160704.
5. Hendrik Bierlee, Jip J. Dekker, Vitaly Lagoon, Peter J. Stuckey, and Guido Tack. Single constant multiplication for sat. In Bistra Dilkina, editor, *Integration of Constraint Programming, Artificial Intelligence, and Operations Research – 21st International Conference, CPAIOR 2024, Proceedings*, volume 14742 of *Lecture Notes in Computer Science*, pages 84–98, Cham, Switzerland, 2024. Springer. doi:10.1007/978-3-031-60597-0_6.
6. Peter Cappello and Kenneth Steiglitz. Some complexity issues in digital signal processing. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 32(5):1037–1041, October 1984. doi:10.1109/TASSP.1984.1164457.
7. Florent de Dinechin and Martin Kumm. *Application-Specific Arithmetic*. Springer, 2024. URL: https://link.springer.com/book/10.1007/978-3-031-42808-1.
8. Andrew G. Dempster and Malcolm D. Macleod. Constant integer multiplication using minimum adders. *IEE Proceedings - Circuits, Devices and Systems*, 141(5):407–413, 10 1994.
9. Nicolai Fiege, Martin Kumm, and Peter Zipf. Bit-level optimized constant multiplication using boolean satisfiability. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 71(1):249–261, 2024. doi:10.1109/TCSI.2023.3327814.
10. Rémi Garcia, Léo Pradels, Silviu-Ioan Filip, and Olivier Sentieys. Hardware-Aware Training for Multiplierless Convolutional Neural Networks. In *ARITH 2025 - 32nd IEEE International Symposium on Computer Arithmetic*, pages 1–8, El Paso, United States, May 2025. IEEE.
11. Rémi Garcia and Anastasia Volkova. Toward the multiple constant multiplication at minimal hardware cost. *IEEE Transactions on Circuits and Systems I: Regular Papers*, pages 1–13, 2023. arXiv:hal-03784625v3, doi:10.1109/TCSI.2023.3241859.
12. Rémi Garcia, Anastasia Volkova, and Martin Kumm. Truncated multiple constant multiplication with minimal number of full adders. In *2022 IEEE International Symposium on Circuits and Systems (ISCAS)*, Austin, Texas, United States, May 2022. IEEE. doi:10.1109/ISCAS48785.2022.9937350.

13 Rémi Garcia, Anastasia Volkova, Martin Kumm, Alexandre Goldsztejn, and Jonas Kühle. Hardware-aware design of multiplierless second-order iir filters with minimum adders. *IEEE Transactions on Signal Processing*, 70:1673–1686, 2022. doi:10.1109/TSP.2022.3161158.

14 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. A Series of Books in the Mathematical Sciences. W. H. Freeman and Company, New York, NY, USA, January 1979.

15 Sidinei Ghissoni, Eduardo Costa, Cristiano Lazzari, José Monteiro, Levent Aksoy, and Ricardo Reis. Radix-2 decimation in time (dit) fft implementation based on a matrix-multiple constant multiplication approach. In *2010 17th IEEE International Conference on Electronics, Circuits and Systems*, pages 859–862, 2010. doi:10.1109/ICECS.2010.5724648.

16 Oskar Gustafsson. Towards optimal multiple constant multiplication: A hypergraph approach. In *Proceedings of the IEEE Asilomar Conference on Signals, Systems and Computers (ACSSC)*, pages 1805–1809, Pacific Grove, CA, USA, October 2008. IEEE.

17 Tobias Habermann, Jonas Kühle, Martin Kumm, and Anastasia Volkova. Hardware-aware quantization for multiplierless neural network controllers. In *2022 IEEE Asia Pacific Conference on Circuits and Systems (APCCAS)*, pages 541–545, 2022. doi:10.1109/APCCAS55924.2022.10090271.

18 Reid M. Hewlitt and Earl S. Swartzlander. Canonical signed digit representation for fir digital filters. In *2000 IEEE Workshop on Signal Processing Systems. SiPS 2000. Design and Implementation*. IEEE, 2000. Cat. No.00TH8528. doi:10.1109/SIPS.2000.886740.

19 Charles D. Howard, Linda S. DeBrunner, and Victor DeBrunner. Hybrid mcm implementation for fir filters in fpga devices. In *2014 IEEE International Conference on Electro/Information Technology (EIT)*, pages 1–6. IEEE, June 2014. doi:10.1109/EIT.2014.6871750.

20 Martin Kumm. *Multiple Constant Multiplication Optimizations for Field Programmable Gate Arrays*. Springer, Wiesbaden, Germany, October 2015. doi:10.1007/978-3-658-10419-7.

21 Martin Kumm. Optimal constant multiplication using integer linear programming. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 65(5):567–571, May 2018. doi:10.1109/TCSII.2017.2772922.

22 Martin Kumm, Anastasia Volkova, and Silviu-Ioan Filip. Design of optimal multiplierless fir filters with minimal number of adders. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 42(2):658–671, 2023. doi:10.1109/TCAD.2022.3179221.

23 Martin Kumm and Peter Zipf. High speed low complexity fpga-based fir filters using pipelined adder graphs. In *2011 International Conference on Field-Programmable Technology (FPT)*. IEEE, December 2011. doi:10.1109/FPT.2011.6132663.

24 Martin Kumm, Peter Zipf, Mathias Faust, and Chip-Hong Chang. Pipelined adder graph optimization for high speed multiple constant multiplication. In *2012 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 49–52. IEEE, May 2012. doi:10.1109/ISCAS.2012.6271996.

25 Vitaly Lagoon and Amit Metodi. Deriving optimal multiplication-by-constant circuits with a sat-based constraint engine. In *ModRef 2020 – The 19th Workshop on Constraint Modelling and Reformulation*, September 2020. September 2020.

26 Pramod Kumar Meher and Yu Pan. Mcm-based implementation of block fir filters for high-speed and low-power applications. In *Proceedings of the IEEE/IFIP 19th International Conference on Very Large Scale Integration and System-on-Chip (VLSI-SoC)*, pages 118–121. IEEE, October 2011. doi:10.1109/VLSISoC.2011.6081653.

27 Singapore Nanyang Technological University. FIRsuite: Suite of Constant Coefficient FIR Filters. https://www.firsuite.net/FIR/FromPublication, November 2023.

28 Nicholas Nethercote, Peter J. Stuckey, Ralph Becket, Sebastian Brand, Gregory J. Duck, and Guido Tack. MiniZinc: Towards a standard cp modelling language. In Christian Bessiere, editor, *Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming (CP 2007)*, volume 4741 of *Lecture Notes in Computer Science*, pages 529–543. Springer, 2007. doi:10.1007/978-3-540-74970-7_38.

29 Xiao Peng and Christine Solnon. BFS-Based Canonical Codes for Generating Graphs with Constraint Programming. In *31st International Conference on Principles and Practice of Constraint Programming (CP 2025)*, volume 340, Glasgow, United Kingdom, August 2025. URL: https://hal.science/hal-05104512, doi:10.4230/LIPIcs.CP.2025.41.

30 Charles Prud'homme and Jean-Guillaume Fages. Choco-solver: A java library for constraint programming. *Journal of Open Source Software*, 7(78):4708, 2022. doi:10.21105/joss.04708.

31 George W. Reitwiesner. Binary arithmetic. In Franz L. Alt, editor, *Advances in Computers*, volume 1, pages 231–308. Academic Press, 1960.

32 James B. Wendt, Nathaniel A. Conos, and Miodrag Potkonjak. Multiple constant multiplication implementations in near-threshold computing systems. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1071–1075. IEEE, April 2015. doi:10.1109/ICASSP.2015.7178134.

33 Neng-Fa Zhou and Håkan Kjellerstrand. The picat-sat compiler. In Marco Gavanelli and John H. Reppy, editors, *Practical Aspects of Declarative Languages (PADL 2016)*, volume 9585 of *Lecture Notes in Computer Science*, pages 48–62. Springer, 2016. doi:10.1007/978-3-319-28228-2_4.