An Exact Breadth-First Search Algorithm for the Multiple Constant Multiplications Problem

Levent Aksoy
Istanbul Technical University
Istanbul, Turkey
aksoyl@itu.edu.tr

Ece Olcay Gunes
Istanbul Technical University
Istanbul, Turkey
ece.gunes@itu.edu.tr

Paulo Flores
IST/INESC-ID, TULisbon
Lisbon, Portugal
pff@inesc-id.pt

***Abstract*—This paper addresses the multiplication of one data sample with multiple constants using addition/subtraction and shift operations, i.e., the multiple constant multiplications (MCM) problem. The MCM problem finds itself and its variants in many applications, such as digital finite impulse response (FIR) filters, linear signal transforms, and computer arithmetic. Although many efficient algorithms have been proposed to implement the MCM using the fewest number of operations, due to the NP-hardness of the problem, they have been heuristics, i.e., they cannot guarantee the minimum solution. In this work, we propose an exact algorithm based on the breadth-first search that finds the minimum number of operations solution of mid-size MCM instances in a reasonable time. The proposed exact algorithm has been tested on a set of instances including FIR filter and randomly generated instances, and compared with the previously proposed efficient heuristics. It is observed from the experimental results that, even though the previously proposed heuristics obtain similar results with the minimum number of operations solutions, there are instances for which the exact algorithm finds better solutions than the prominent heuristics.**

I. Introduction

In several computationally intensive operations, such as Finite Impulse Response (FIR) filters as illustrated in Figure 1 and fast Fourier transforms, the same input is multiplied by a set of coefficients, an operation known as Multiple Constant Multiplications (MCM). These operations are typical in digital signal processing applications and hardwired dedicated architectures are the best option for maximum performance and minimum power consumption.

Constant coefficients allow for a great simplification of the multipliers, which can be reduced to a set of shift-adds [1]. When the same input is to be multiplied by a set of constant coefficients, significant reductions in hardware, and consequently power, can be also obtained by sharing the partial products of the input among the set of multiplications. Since shifts are free in terms of hardware, the MCM problem is defined as finding the minimum number of addition/subtraction operations to implement the constant multiplications. The MCM problem has been proven to be NP-hard in [2].

As an example of an MCM problem, suppose the multiple constant multiplications of 11 and 13 by the input x as given in Figure 2(a). The shift-adds implementations of constant multiplications are presented in Figure 2(b)-(c). Observe that while the multiplierless implementation without partial product sharing requires 4 operations, Figure 2(b), the sharing of partial product $9x$ in both multiplications reduces the number of operations to 3, Figure 2(c).

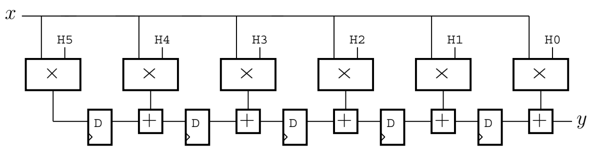


Fig. 1. Transposed form of a hardwired FIR filter implementation.

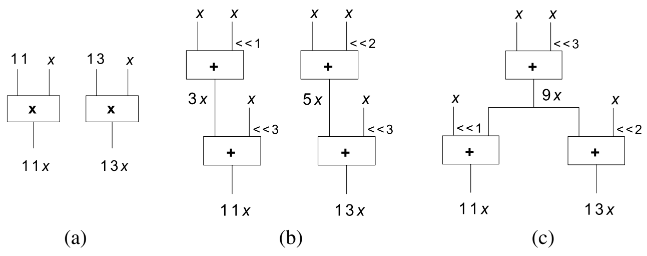


Fig. 2. (a) Multiple constant multiplications; Shift-adds implementations: (b) without partial product sharing; (c) with partial product sharing.

In the last decade, many efficient algorithms have been proposed for the optimization of the number of operations in MCM. These methods range from the Common Subexpression Elimination (CSE) algorithms [3], [4] to the graph-based coefficient synthesis techniques [5], [6]. The CSE algorithms basically find common non-zero digit patterns on the representations of the constants. For the example given in Figure 2, the sharing of partial product $9x$ is possible, when constants in multiplications $11x$ and $13x$ are defined in binary, i.e., $11x = (1011)_{bin}x$ and $13x = (1101)_{bin}x$ respectively, and the common partial product, i.e., $(1001)_{bin}x = x + 8x = 9x$, is identified in both multiplications. The exact CSE algorithms of [7], [8] formalize the MCM problem as a 0-1 integer linear programming problem and find the minimum number of operations solution of the MCM problem by maximizing the partial product sharing. However, the solution of the exact CSE algorithm depends on the number representation.

It is argued in [9] that being limited to a number representation does not yield the minimum number of operations solutions. The algorithm of [9] extends the exact CSE algorithm of [8] to handle the constants under general number representation increasing the search space and finds more promising solutions than those of the exact CSE algorithm. However, the algorithm of [9] does not consider the whole search space as graph-based algorithms, since it has limitations on the implementation of constants to guarantee the solution to be represented in a directed acyclic graph.

The graph-based algorithms are not restricted to any number representation and synthesize the constants iteratively by building a graph. Although the graph-based algorithms give better results than CSE algorithms as shown in [6], they are based on heuristics and provide no indication on how far from the minimum their solutions are. In a recent paper [10], the lower bound on the number of required operations to implement the MCM was investigated. The solution found by a heuristic algorithm is determined as minimum, if the number of operations in the found solution is equal to the lower bound. To the best of our knowledge, there is no exact graph-based algorithm proposed for the MCM problem.

In this paper, we introduce an exact graph-based algorithm, called BFS_{mcm}, that searches the minimum number of operations solution of the MCM problem in a breadth-first manner. We present the worst-case complexity of the exact algorithm and show that it can be applied on mid-size MCM instances. Although it is observed from the experimental results that the previously proposed graph-based algorithms find solutions very close to the exact solutions which shows for the first time the quality of heuristics used in these algorithms, we also note that there are instances for which BFS_{mcm} finds better solutions than the prominent graph-based heuristics.

The rest of the paper is organized as follows: Section II gives the background concepts and Section III describes the exact algorithm. Experimental results are presented in Section IV and finally, the conclusions and directions for the future work are given in Section V.

II. Background

In this section, initially, we give the main concepts and the problem definition, and then, we present an overview of the graph-based algorithms.

A. Definitions

In the MCM problem, the main operation, called *A-operation* in [6], is an operation with two integer inputs and one integer output that performs a single addition or a subtraction, and an arbitrary number of shifts. It is defined as

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

where $l_1, l_2 \geq 0$ are integers denoting left shifts, $r \geq 0$ is an integer indicating the right shift, and $s \in \{0,1\}$ is the sign that denotes the addition/subtraction operation to be performed. The operation that implements a constant can be represented in a graph where the vertices are labeled with constants and the edges are labeled with the sign and shifts as illustrated in Figure 3. In the MCM problem, the complexity of an adder and a subtracter is assumed to be equal in hardware. It is also assumed that the sign of the constant can be adjusted at some part of the design and the shifting operation has no cost, since shifts can be implemented with only wires in hardware. Thus, in the MCM problem, only positive and odd constants are considered. Observe from (1) that in the implementation of an odd constant, considering odd constants at the inputs, one of the left shifts, l_1 or l_2, is zero and r is zero, or l_1 and l_2 are zero and r is greater than zero. In finding an operation to implement a constant, it is also necessary to constrain the left

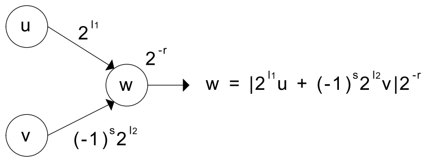


Fig. 3. The representation of the A-operation in a graph.

shifts, l_1 and l_2, otherwise a constant can be implemented in infinite ways. As shown in [5], it is sufficient to limit the shifts by the maximum bit-width of the constants to be implemented, i.e., bw, since allowing larger shifts than bw does not improve the solutions obtained with the former limits. In BFS_{mcm}, as in the algorithm of [6], the shifts are allowed to be at most $bw+1$. Thus, the MCM problem can be also defined as follows.

Definition 1: THE MCM PROBLEM - Given the target set, $T = \{t_1, \ldots, t_n\} \subset \mathbb{N}$, including the positive and odd unrepeated target constants to be implemented, find the smallest ready set $R = \{r_0, r_1, \ldots, r_m\}$ 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 operation $r_k = A(r_i, r_j)$.

Hence, the number of operations required to implement the MCM is $|R| - 1$ [6].

B. Related Work

For the single constant multiplication problem, an exact algorithm that finds the minimum number of required operations for a constant up to 12 bit-width is introduced in [11] and it is extended up to 19 bit-width in [12]. For the MCM problem, four algorithms, 'add-only', 'add/subtract', 'add/shift', and 'add/subtract/shift', are proposed in [13]. The latter algorithm, i.e., 'add/subtract/shift', is modified in [5], called BHM, by extending the possible implementations of a constant, considering only odd numbers, and processing constants in order of increasing single coefficient cost that is evaluated by the algorithm of [11]. A graph-based algorithm, called RAG-n, is also introduced in [5]. RAG-n has two parts: optimal and heuristic. In the optimal part, each target constant that can be implemented with a single operation, whose inputs are in the ready set, are synthesized. If there exist unimplemented element(s) left in the target set, the algorithm switches to the heuristic part where an intermediate constant is added to the target set. In the selection of the intermediate constant, RAG-n chooses an unimplemented target constant with the smallest single coefficient cost and synthesizes it with an intermediate constant that has the smallest value among the possible constants. The graph-based heuristic of [6], called Hcub, includes the same optimal part of RAG-n, but uses a better heuristic that considers the impact of each possible intermediate constant on all target constants to be implemented and chooses the one that yields the best cumulative benefit. Also, Hcub is not restricted to the lookup table that includes the single coefficient cost of constants as RAG-n, thus it is applicable to larger size constants. To the best of our knowledge, Hcub finds significantly better solutions than any of the previously published algorithms.

We make two simple observations on the algorithms Hcub and RAG-n. In these observations, $|T|$ denotes the number of elements of the target set to be implemented, i.e., the lowest bound on the number of required operations.

Lemma 1: If Hcub or RAG-n finds a solution with $|T|$ operations, then the found solution is minimum.

In this case, no intermediate constant is required to implement the target constants. Since the elements of the target set cannot be synthesized using less than $|T|$ operations as shown in [5] and the solution is obtained in the optimal part, then the found solution is the minimum solution. □

Lemma 2: If Hcub or RAG-n finds a solution with $|T| + 1$ operations, then the found solution is minimum.

In this case, only one intermediate constant is required to implement the target constants. If these heuristics cannot find a solution in the optimal part, then it is obvious that at least one intermediate constant is required to find the minimum solution. So, if the found solution includes $|T| + 1$ operations, then it is the minimum solution. □

Note that Hcub and RAG-n cannot determine their solutions as minimum if the obtained solutions include the number of operations more than the number of target constants to be implemented plus 1. Because, in this case, the target and intermediate constants are synthesized once at a time in the heuristic parts of these algorithms.

III. The Exact Breadth-First Search Algorithm

As defined in Section II-A, the MCM problem is to find the minimum number of intermediate constants such that each constant, target and intermediate, can be implemented with an operation as given in (1) where u and v are 1, target, or intermediate constants. It is obvious that the minimum number of intermediate constants, thus the minimum number of operations solution of the MCM problem, can be found using a breadth-first search.

In the preprocessing phase of $\mathrm{BFS_{mcm}}$, the target constants are made positive and odd, and added to the target set, T, without repetition. The maximum bit-width of the target constants, bw, is determined. In the main part of $\mathrm{BFS_{mcm}}$ given in Algorithm 1, the ready set, R, that includes the minimum number of elements is computed.

In *BFSearch* function, initially, the target constants that can be implemented with the elements of the ready set, $\{1\}$, are found iteratively and removed to the ready set using the *Synthesize* function (lines 1-2), as done in the optimal parts of RAG-n and Hcub. If there is no element left in the target set, then the minimum number of operations solution is obtained. Otherwise, the intermediate constants to be added to the ready set are considered exhaustively in the infinite loop, i.e., the line 7 of the algorithm, until all the target constants are synthesized. The infinite loop starts with the array of ready and target sets, W_{R_1} and W_{T_1}, i.e., the ready and target sets obtained on the line 2 of the algorithm. Note that the size of the array *W* including ready and target sets as a pair is denoted by *n*. Then, in the infinite loop, another array *X* is assigned to the array *W* and its size is represented with *m*. In an iteration of the infinite loop, for each ready set of the array *X*, the possible intermediate constants are found and added to the associated ready set forming new ready sets. The possible

Algorithm 1 $\mathrm{BFS_{mcm}}$. The algorithm takes the target set, T, including target constants to be implemented and returns the ready set, R, with the minimum number of elements including 1, target, and intermediate constants.

```
BFSearch(T, bw)
 1: $R \leftarrow \{1\}$
 2: $(R, T)$ = Synthesize($R$, $T$)
 3: if $T = \emptyset$ then
 4:    return $R$
 5: else
 6:    $n = 1$, $W_{R_1} \leftarrow R$, $W_{T_1} \leftarrow T$
 7:    while 1 do
 8:       $m = n$, $\mathrm{X_R} = \mathrm{W_R}$, $\mathrm{X_T} = \mathrm{W_T}$
 9:       $n = 0$, $\mathrm{W_R} = \mathrm{W_T} = [\,]$
10:       for $i = 1$ to $m$ do
11:          for $j = 1$ to $2^{bw+1} - 1$ step 2 do
12:             if $j \notin X_{R_i}$ and $j \notin X_{T_i}$ then
13:                $(A, B)$ = Synthesize($X_{R_i}$, $\{j\}$)
14:                if $B = \emptyset$ then
15:                   $X_{R_i} \leftarrow X_{R_i} \cup \{j\}$
16:                   $n = n + 1$
17:                   $(W_{R_n}, W_{T_n})$ = Synthesize($X_{R_i}$, $X_{T_i}$)
18:                   if $W_{T_n} = \emptyset$ then
19:                      return $W_{R_n}$

Synthesize(R, T)
 1: repeat
 2:    isadded = 0
 3:    for $k = 1$ to $|T|$ do
 4:       if $t_k$ can be synthesized with the elements of $R$ then
 5:          isadded = 1
 6:          $R \leftarrow R \cup \{t_k\}$
 7:          $T \leftarrow T \setminus \{t_k\}$
 8: until isadded = 0
 9: return $(R, T)$
```

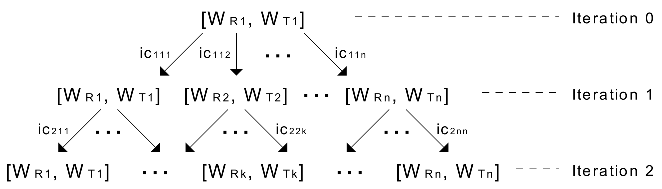


Fig. 4. The flow of $\mathrm{BFS_{mcm}}$ in two iterations.

intermediate constants are determined from odd constants that are not included in the current ready and target sets, X_{R_i} and X_{T_i}, and can be implemented with the elements of the current ready set (lines 11-14). Note that there is no need to consider the constants that cannot be implemented with the elements of the current ready set, since all these constants are considered in other ready sets due to the exhaustiveness of the algorithm. After the intermediate constant is added to the ready set X_{R_i}, its implications on the target set X_{T_i} are found by the *Synthesize* function and the modified ready and target sets are stored to the array *W* as a new pair (line 17).

The flow of the algorithm in two iterations is sketched in Figure 4 indicating the array *W* at the end of iterations. In this figure, the edges labeled with the intermediate constants represent the inclusions of constants to the ready set. An intermediate constant is denoted by ic_{abc} where a, b, and c denote the number of iteration it is included, the index of the ready set it is added, and its index in the iteration respectively.

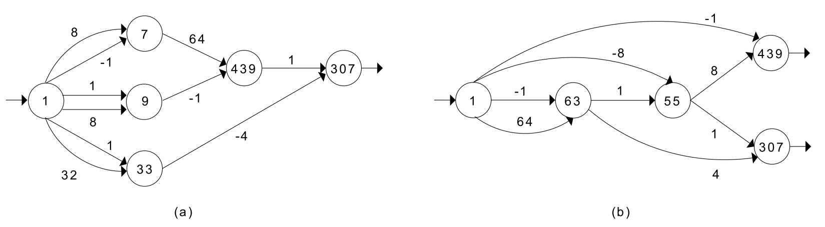


Fig. 5. The results of algorithms for the target constants 307 and 439: (a) 5 operations with Hcub; (b) 4 operations with $\mathrm{BFS_{mcm}}$.

Observe from Figure 4 that $\mathrm{BFS_{mcm}}$ explores the search space in a breadth-first manner. In each iteration, each ready set is augmented with a single intermediate constant. For example, while the ready set W_{R_1} at the end of the second iteration includes 1, the intermediate constants ic_{111}, ic_{211}, and the target constants that can be implemented with the elements of W_{R_1}, the associated target set W_{T_1} consists of the target constants have not been implemented by the elements of W_{R_1} so far. Hence, when there is no element left in a target set, the minimum number of operations solution is obtained with the associated ready set (lines 18-19).

We make an observation on $\mathrm{BFS_{mcm}}$ algorithm where $|T|$ denotes the number of target constants to be implemented.

Lemma 3: The solution obtained by $\mathrm{BFS_{mcm}}$ *algorithm yields the minimum number of operations solution.*

If a solution is returned on the line 4 of the *BFSearch* function, then no intermediate constant is required to implement the target constants. Hence, each target constant can be implemented using a single operation whose inputs are 1 or target constants as ensured by the *Synthesize* function. In this case, the number of required operations to implement the target constants is $|T|$. Because the target constants cannot be implemented using less than $|T|$ operations as shown in [5], the obtained ready set yields the minimum solution.

If a solution is returned on the line 19 of the *BFSearch* function, then intermediate constant(s) are required to implement the target constants. In this case, the number of required operations to implement the target constants is $|T|$ plus the number of intermediate constant(s). Because each element of the ready set, except 1, is guaranteed to be implemented using a single operation and all possible intermediate constants are considered in a breadth-first manner, the obtained ready set yields the minimum number of operations solution. $\square$

As can be observed from Lemma 3, after a ready set including the minimum number of intermediate constants is obtained by $\mathrm{BFS_{mcm}}$, the minimum number of operations implementation of the MCM problem can be realized by synthesizing the target and intermediate constants using a single operation whose inputs are 1, target, or intermediate constants as given in (1).

As a small example, suppose the target set including 307 and 439. Figure 5 presents the solutions obtained by Hcub and $\mathrm{BFS_{mcm}}$. As can be easily observed from Figure 5(a), since Hcub synthesizes each target constant in an iteration by including an intermediate constant, the intermediate constants included for the implementation of target constants in previous iterations may not be shared in the implementation of target constants in later iterations, although Hcub is particularly designed for this case. In $\mathrm{BFS_{mcm}}$, initially, it is observed that the target constants cannot be implemented using a single operation whose inputs are the elements of the ready set, i.e., $\{1\}$. Then, in the first iteration, the intermediate constants that can be implemented using a single operation with the elements of the ready set $\{1\}$, i.e., $3, 5, \ldots, 1023$, are found. However, all the possible ready sets including one intermediate constant, i.e., $\{1,3\}, \{1,5\}, \ldots, \{1,1023\}$, also cannot synthesize all the target constants. In the second iteration, for each ready set obtained in the first iteration, the intermediate constants that can be implemented with the elements of the associated ready set are found and added to the associated ready set. As can be observed from Figure 5(b), all the target constants are synthesized when the intermediate constant 55 is added to the ready set $\{1, 63\}$, i.e., one of the ready sets obtained in the first iteration of the exact algorithm.

TABLE I
UPPER BOUNDS ON THE NUMBER OF READY SETS EXPLOITED BY THE EXACT GRAPH-BASED ALGORITHM UNDER DIFFERENT BIT-WIDTHS.

bw	#ready sets considered in iterations				
	1	2	3	4	Total
8	15	378	12,398	1,668,403	1,681,194
9	17	504	20,118	5,897,424	5,918,063
10	19	648	30,428	19,000,657	19,031,752
11	21	810	43,761	57,559,925	57,604,517
12	23	990	60,435	165,546,959	165,608,407
13	25	1,188	80,907	458,873,308	458,955,428
14	27	1,404	105,462	1,230,677,125	1,230,784,018

The complexity of search space in $\mathrm{BFS_{mcm}}$ is dependent on both the number of considered ready sets and the maximum bit-width of the target constants, i.e. *bw*, since the number of considered ready sets increases as *bw* increases. Table I presents the number of ready sets exploited by $\mathrm{BFS_{mcm}}$ including up to 4 intermediate constants when *bw* is in between 8 and 14. The exponential growth of the search space can be clearly observed when the number of iterations increases. Because, the inclusion of an intermediate constant to a ready set in the current iteration increases the number of possible intermediate constants to be considered in the next iteration.

We note that the complexity of the search space also depends on the target constants to be implemented in an MCM instance. There are cases where multiple constants may reduce the complexity of the search space. For example, consider the single target constant 981 defined in 10 bit-width. The minimum number of operations implementation of 981 requires four operations, $3 = 1_{\ll 2} - 1$, $5 = 1_{\ll 2} + 1$, $43 = 5_{\ll 3} + 3$, and $981 = 1_{\ll 10} - 43$, thus three intermediate constants, 3, 5, and 43. To find this minimum solution, in the worst case, a total of $19 + 648 + 30428 = 31905$ ready sets must be considered. Now, suppose the multiple target constants 43 and 981. In this case, the minimum number of operations solution is found in two iterations with the ready set, $\{1, 3, 5\}$, including two intermediate constants, i.e., in the worst case, $19 + 648 = 667$ ready sets are exploited. As can be observed from this example, the number of ready sets exploited by the exact algorithm depends heavily on the target constants to be implemented. Hence, the exact algorithm can be efficiently applied on instances including large number of constants as shown in Section IV. Also, note that the minimum solution is generally obtained before the total number of ready sets are considered. Hence, Table I presents the upper bounds on the number of ready sets exploited by $\mathrm{BFS}_{\mathrm{mcm}}$. We note that the exact graph-based algorithm can obtain the minimum solutions of the MCM instances that require less than 5 intermediate constants in a reasonable time.

IV. Experimental Results

In this section, we present the results of $\mathrm{BFS}_{\mathrm{mcm}}$ on FIR filter and randomly generated instances and compare with those of the exact CSE algorithm [8] and those of the previously proposed graph-based heuristics [5] and [6]. The graph-based heuristics were obtained from [14].

As the first experiment set, we used uniformly distributed randomly generated instances where constants were defined under 14 bit-width. The number of constants ranges between 10 and 100, and we generated 30 instances for each of them. Thus, the experiment set includes 300 instances. Figure 6 presents the solutions obtained by the exact CSE algorithm of [8] when constants are defined under binary, canonical signed digit (CSD), and minimum signed digit (MSD) representations, and the minimum number of operations solutions obtained by $\mathrm{BFS}_{\mathrm{mcm}}$.

As can be observed from Figure 6, the solutions obtained by the exact CSE algorithm are far from the minimum number of operations solutions, since the implementations of constants in the exact CSE algorithm are restricted to the number representation. The average difference of the number of operations solutions between the exact CSE algorithm under binary, CSD, and MSD, and $\mathrm{BFS}_{\mathrm{mcm}}$ is 8.5, 10.8, and 8.6 respectively on overall 300 instances. Since both algorithms are exact according to the techniques they are based on, i.e., CSE and graph-based, we can clearly state that the graph-based algorithms obtain significantly better solutions than the CSE algorithms.

As the second experiment set, we used FIR filter instances where filter coefficients were computed with the *remez* algorithm in MATLAB. The specifications of filters are presented

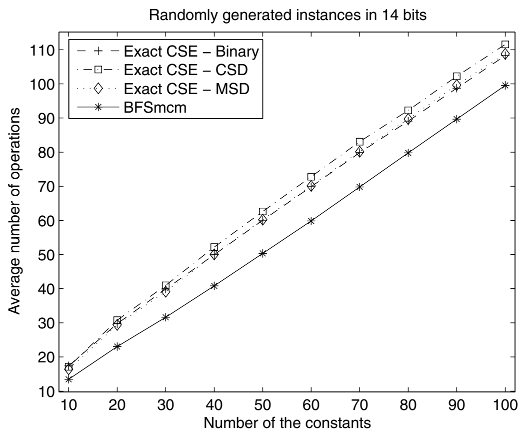


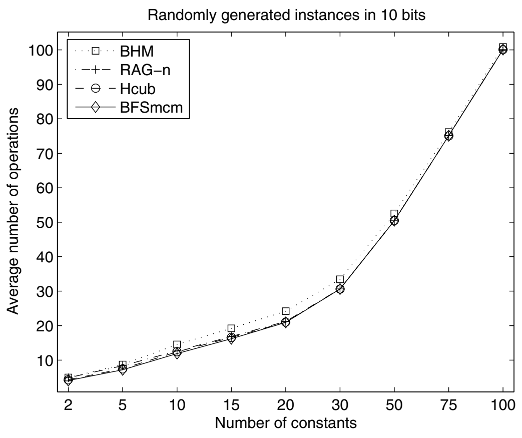
Fig. 6. Comparison of the solutions of the exact CSE algorithm with the minimum number of operations solutions.

in Table II where: *pass* and *stop* are normalized frequencies that define the passband and stopband respectively; *#tap* is the number of coefficients; and *width* is the bit-width of the coefficients. We note that the filter 11 was used as an example filter in [10]. In this table, $|T|$ denotes the number of positive and odd unrepeated filter coefficients, i.e., the lowest bound on the number of operations, and the *LBs* indicates the lower bounds on the number of operations and the number of operations in series, generally known as adder-step, obtained by the formulas given in [10]. The results of graph-based algorithms are also presented in Table II where *adder* denotes the number of operations, *step* indicates the number of adder-step, and *CPU* is the required CPU time in seconds of $\mathrm{BFS}_{\mathrm{mcm}}$ implemented in MATLAB to obtain the minimum solution on a PC with 2.4GHz Intel Core 2 Quad CPU. In this table, the required CPU times for the heuristics are not listed, since they obtained the solutions in a few seconds.

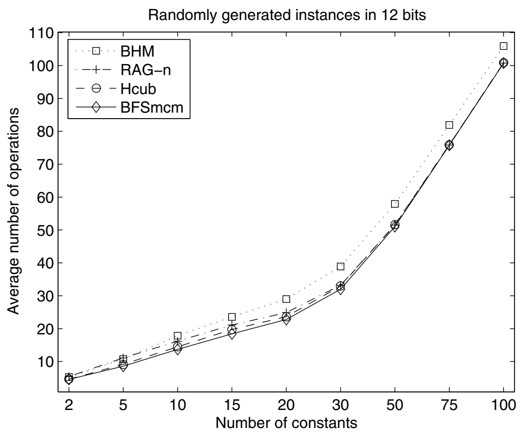
As can be easily observed from Table II, $\mathrm{BFS}_{\mathrm{mcm}}$ finds the minimum number of operations solutions with a little computational effort, since the minimum solutions require at most three extra intermediate constants. Observe that the CPU time required to find the minimum solution increases, as the minimum number of the required intermediate constants increases. We note that Hcub finds similar results with $\mathrm{BFS}_{\mathrm{mcm}}$, but it obtains worse solutions on filters 1, 2, and 6, and according to Lemma 2, Hcub determines only its solution on filter 4 as the minimum solution. On the other hand, BHM and RAG-n obtain suboptimal results on all filter instances that are far from the minimum solutions. Also, observe that

TABLE II
Filter specifications and summary of results of algorithms on FIR filter instances.

Filter	Filter Specifications				$\lvert T\rvert$	LBs [10]		BHM [5]		RAG-n [5]		Hcub [6]		$\mathrm{BFS}_{\mathrm{mcm}}$		
	pass	stop	#tap	width		adder	step	adder	step	adder	step	adder	step	adder	step	CPU
1	0.10	0.15	40	14	19	20	3	25	10	24	10	23	7	22	8	126.8
2	0.10	0.15	80	16	39	40	3	48	6	44	9	42	8	41	9	25.3
3	0.10	0.25	30	14	14	14	3	18	6	19	5	16	5	16	5	42.3
4	0.10	0.25	80	16	33	33	3	35	6	37	5	34	5	34	5	1.1
5	0.10	0.20	40	14	18	19	3	23	8	22	5	20	5	20	5	4.5
6	0.10	0.20	80	16	36	37	3	40	9	40	5	38	5	37	6	0.6
7	0.15	0.25	40	14	19	19	3	25	6	22	5	21	7	21	5	2.9
8	0.15	0.25	60	16	29	29	3	36	6	33	7	31	7	31	7	17.5
9	0.20	0.25	40	14	19	20	3	24	7	25	5	21	6	21	7	28.9
10	0.20	0.25	60	16	29	30	3	36	7	34	6	31	7	31	7	20.4
11	0.25	0.30	25	12	13	14	3	19	5	17	9	16	7	16	8	210.8
Total	—	—	—	—	268	275	33	329	76	317	71	293	69	290	72	481.1



(a)



(b)

Fig. 7. Comparison of algorithms on randomly generated hard instances: (a) with constants defined under 10 bits; (b) with constants defined under 12 bits.

the lower bounds on the number of operations [10] cannot be used to determine any of the solutions found by the heuristics as the minimum solution. This is because the formula of [10] computes a lower bound that is close to the lowest bound. Thus, this experiment clearly indicates that an exact algorithm is indispensable to compute the minimum solution.

As the third experiment set, we used randomly generated instances where the constants are defined in between 10 and 12 bit-width. We tried to generate hard instances to distinguish the algorithms clearly. Hence, under each bit-width, i.e., bw, the constants were generated randomly in $[2^{bw-2}+1, 2^{bw-1}-1]$. Also, the number of constants were determined as 2, 5, 10, 15, 20, 30, 50, 75, and 100, and we generated 30 instances for each of them. The experiment set includes 810 instances. Figure 7 presents the results of the algorithms on randomly generated hard instances defined under 10 and 12 bit-width.

As can be easily observed from Figure 7(a)-(b), the graph-based heuristics, except BHM, obtain competitive results with those of $\mathrm{BFS_{mcm}}$. However, on the instances with 30 constants defined in 12 bits, the difference of the average number of operations between Hcub and $\mathrm{BFS_{mcm}}$ is almost 1, and on the instances with 15 constants defined in 12 bits, this value between RAG-n and $\mathrm{BFS_{mcm}}$ is 2.7. We note that the number of instances that BHM, RAG-n, and Hcub found the same solutions with $\mathrm{BFS_{mcm}}$ is 65, 419, and 562 respectively on overall 810 instances. However, the number of instances that RAG-n and Hcub guaranteed the minimum solutions according to Lemma 1-2 is 343 and 354 respectively. This experiment indicates that although the heuristics obtain similar results with our exact graph-based algorithm, the number of instances proven to be minimum by the heuristics themselves is much less than the number of instances guaranteed to be minimum by the results of our exact algorithm.

V. Conclusions

In this work, we introduce an exact graph-based algorithm that uses a breadth-first search to find the minimum number of operations solution of the MCM problem. Unlike the exact CSE algorithms, the proposed exact algorithm is independent from the representation used for constants. The experimental results show that our exact algorithm can be applied on low complex instances of real size FIR filters. It is also observed that the previously proposed graph-based heuristic of [6] that finds significantly better solutions than any other previously published graph-based heuristics also obtains solutions close to the minimum number of operations solutions. This shows the quality achieved by the heuristic algorithm. However, it is shown that it may find suboptimal solutions on small size instances where the exact algorithm finds the minimum solutions in a reasonable time.

References

[1] H. Nguyen and A. Chatterjee, "Number-Splitting with Shift-and-Add Decomposition for Power and Hardware Optimization in Linear DSP Synthesis," *IEEE Transactions on VLSI*, vol. 8, pp. 419–424, 2000.

[2] P. Cappello and K. Steiglitz, "Some Complexity Issues in Digital Signal Processing," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 32, no. 5, pp. 1037–1041, October 1984.

[3] R. Hartley, "Subexpression Sharing in Filters using Canonic Signed Digit Multipliers," *IEEE Transactions on Circuits and Systems II*, vol. 43, no. 10, pp. 677–688, 1996.

[4] R. Pasko, P. Schaumont, V. Derudder, S. Vernalde, and D. Durackova, "A New Algorithm for Elimination of Common Subexpressions," *IEEE Transactions on Computer-Aided Design*, vol. 18, pp. 58–68, 1999.

[5] A. Dempster and M. Macleod, "Use of Minimum-Adder Multiplier Blocks in FIR Digital Filters," *IEEE Transactions on Circuits and Systems II*, vol. 42, no. 9, pp. 569–577, 1995.

[6] Y. Voronenko and M. Puschel, "Multiplierless Multiple Constant Multiplication," *ACM Transactions on Algorithms*, vol. 3, no. 2, 2007.

[7] O. Gustafsson and L. Wanhammar, "ILP Modelling of the Common Subexpression Sharing Problem," in *Proceedings of International Conference on Electronics, Circuits and Systems*, 2002, pp. 1171–1174.

[8] P. Flores, J. Monteiro, and E. Costa, "An Exact Algorithm for the Maximal Sharing of Partial Terms in Multiple Constant Multiplications," in *Proceedings of International Conference on Computer-Aided Design*, 2005, pp. 13–16.

[9] L. Aksoy, E. Costa, P. Flores, and J. Monteiro, "Minimum Number of Operations under a General Number Representation for Digital Filter Synthesis," in *Proceedings of European Conference on Circuit Theory and Design*, 2007, pp. 252–255.

[10] O. Gustafsson, "Lower Bounds for Constant Multiplication Problems," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 54, no. 11, pp. 974–978, 2007.

[11] A. Dempster and M. Macleod, "Constant Integer Multiplication using Minimum Adders," *IEE Proceedings - Circuits, Devices, and Systems*, vol. 141, no. 5, pp. 407–413, 1994.

[12] O. Gustafsson, A. Dempster, and L. Wanhammar, "Extended Results for Minimum-adder Constant Integer Multipliers," in *Proceedings of the International Symposium on Circuits and Systems*, 2002, pp. 73–76.

[13] D. Bull and D. Horrocks, "Primitive Operator Digital Filters," *IEE Proceedings G: Circuits, Devices and Systems*, vol. 138, no. 3, pp. 401–412, 1991.

[14] Spiral webpage. [Online]. Available: http://www.spiral.net