International Journal of Engineering and Technology, 4 (4) (2015) 472-488
www.sciencepubco.com/index.php/IJET
©Science Publishing Corporation
doi: 10.14419/ijet.v4i4.5132
Research Paper

Developing a combinatorial model for the multiple constant multiplication

Firas Al-Hasani

TracMap company, 21B Gladstone Road South, Mosgiel 9053, New Zealand
E-mail: firasissa75@yahoo.com

Abstract

A combinatoric model for the multiple constant multiplication (MCM) operation is developed. The model is found by decomposing each coefficient using the $\mathcal{A}-$operation into two subexpressions. The constituted subexpressions are, in turn, decomposed using the $\mathcal{A}-$operation. Connecting all of the decompositions results the decomposition graph which represents the solution space. The decomposition graph itself is not feasible for routing to find the minimum solutions. Therefore, a transformation on the $\mathcal{A}-$operation is proposed to make the decomposition graph routable. In this case, the $\mathcal{A}-$operation is transformed into a subexpression operation by replacing the shift information attached to the arcs by the other subexpression information which is called the demand. A demand that attached to an arc will represent its cost. The resulting transformed graph is called the demand graph. The demand graph is augmented with deadheading arcs to make it routable. Deadheading arcs are with zero demand. Similarly, traversing an arc with synthesized demand is of zero cost. Enumerating all of the routes that start from the signal vertex and visit all the coefficients gives all the solutions of the MCM problem. The routing technique requires redirecting the route when encountering an unsynthesized demand. The route in this case backtrack to the first encountered synthesized ancestors for this demand. This routing style analogous to the dynamic capacitated arc routing. To prevent exhaust routing, ant colony optimization (ACO) meta-heuristics is proposed to traverse the augmented demand graph. The solution space contains all the possible solutions that can be obtained from using both of the common subexpression elimination (CSE) and graph dependent heuristics that traditionally used to solve the MCM operation.

Keywords: *Multiple Constant Multiplication (MCM) Operation, Demand Graph, Dynamic Capacitated Arc Routing Problem, Ant Colony Optimization (ACO) Metaheuristics, Common Subexpression Elimination (CSE) Heuristics, $\mathcal{A}-$operation Decomposition.*

1. Introduction

Multiple constant multiplication (MCM) operation is a core operation in DSP systems. Examples of DSP building blocks that implement MCM are found in finite impulse response (FIR) filters, infinite impulse response (IIR) filters, and the Constant Matrix-Vector Multiplication (CMVM) operation such as the Fast Fourier Transform (FFT) and Discrete Cosine Transform (DCT) [2]. Therefore, optimizing MCM operation optimizes the performance of the underling DSP system. This is why the problem of optimizing the MCM operation has been an active research area in the last three decades [3] [5] [9] [12] [13] [20] [26] [28] [29] [37] [45] [47]. Optimizing the MCM operation is achieved by minimizing the number of logic operators and logic depth in it. To explain the concept, consider the 2-tap FIR filter shown in Fig. 1. In this figure, 37 and 47 are the coefficients, x_n is the input variable at discrete

time n, and Z^{-1} is the delay operator. The direct realization of Fig. 1 requires using 2 hardware multipliers and 1 structure adder. If the signal x_n is of 8-bit wordlength, then two 8×6 digital multipliers are required to implement the multiplier block (MB) shown in Fig. 1. The total number of full adders (FA) in the two multiplier arrays will be $2 \times 8 \times 6 = 96$. This example shows the reason of considering the multiplier as a bottleneck element in DSP systems because it consumes a large area, high power, and has a long delay (critical path) [1]. To solve the multiplier problem, the MCM is designed to be free of the explicit multiplication (multiplierless) [5] [47] by using simpler logic operations of shift and add/subtract as shown in Fig. 2. The shift operation can be realized by redirecting the signal path making it a hardware free operation, while the hardware cost of addition and subtraction is assumed to be the same [20] [29] [44] [47]. In Fig. 2, the number of MB adders equals 4, if 8-bit length adders are used, then the total number of FA becomes $4 \times 8 = 32$ FA. This number is less than that implemented by the two multipliers shown in Fig. 1. In addition, the number of MB adders can be reduced if common factors are found and eliminated inside the block. For example, the multiplier $15x_n$ in Fig. 2 is common between the two multipliers 37 and 47. Sharing this block between the multipliers 37 and 47 results Fig. 3 with a number of FA equals to $3 \times 8 = 24$, which is about quarter of the cost of realizing Fig. 1. This method of finding and eliminating the common factors in the MB is called common subexpression elimination (CSE) [28].

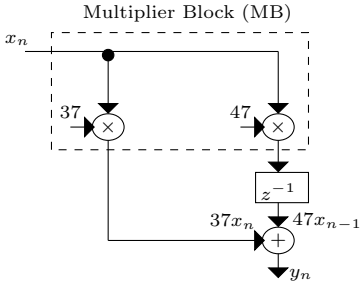


Figure 1: The transposed structure of the FIR filter $y_n = 47x_{n-1} + 37x_n$.

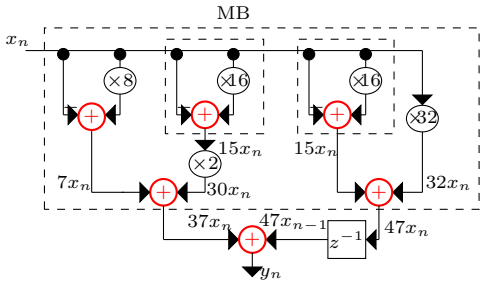


Figure 2: Multiplication free realization of the filter $y_n = 47x_{n-1} + 37x_n$.

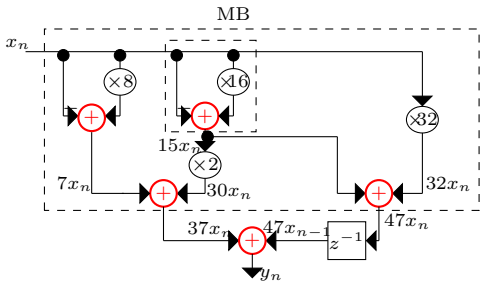


Figure 3: Eliminating the common factor $5x_n$ in Fig. 2 reduces the number of MB adders to 3.

The hardware complexity of the multiplierless MCM block is measured by the number of adders or logic operators (LO) required to implement it and by the logic depth (LD), which is the longest path of consecutive adders. The problem of minimizing the hardware complexity of MCM is now achieved by minimizing its LO and LD. The minimum LD of MCM is specified by the coefficient w if its canonical signed digit (CSD) representation is

of maximum adder step δ_w among other coefficients' adder steps, where the CSD representation is the same of binary but with no two adjacent digits both of them are of non-zero value. The coefficients are assumed to be synthesized from their CSD representations in order to use minimum number of adder steps. For example, the CSD representation of 47 is $10\bar{1}000\bar{1}$ with a minimum Hamming weight of $\mathcal{H} = 3$, where $\mathcal{H}$ is the number of non-zero digits in the representation. The resulting minimum adder step realization of 47 is shown in Fig. 4 which is with $\delta_{47} = 2$. The relationship between the adder step δ_w of coefficient w and the Hamming weight $\mathcal{H}_w$ is given by [21] [24] [47]

$$\delta_w = \lceil \log_2(\mathcal{H}_w) \rceil. \tag{1}$$

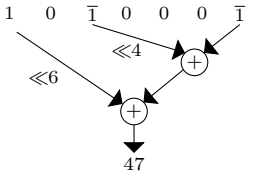


Figure 4: Synthesis of 47 from its CSD representation $10\bar{1}000\bar{1}$ requires 2 adders.

The problem of minimizing the hardware complexity of MCM is classified as NP-hard because there is no guarantee to find the minimum solutions in polynomial time [25] [29] [44] [46]. Therefore, heuristic algorithms are needed to solve the problem in polynomial time. Examples of heuristics that traditionally used to tackle the MCM problem are the CSE [8] [12] [33] [43] [47] and graph dependent (GD) methods [3] [9] [13] [25] [46]. The CSE method (described previously) searches a space of subexpressions [47] to find minimum realization of the MCM coefficients. On the other hand, the GD method searches for the minimum number of fundamentals [13] required to synthesis the MCM coefficients.

Finding more efficient search algorithms than heuristics requires developing a combinatorial model for the MCM problem. Once this model is found, metaheuristic algorithms can be used instead of heuristics to search a space of candidate solutions. Examples of metaheuristics include evolutionary computation and ant colony optimization [11] [15]. One definition of metaheuristic is given by Dorigo [15] as "a metaheuristic can be seen as a general-purpose heuristic method designed to guide an underlying problem-specific heuristic toward promising regions of the search space containing high-quality solutions". To the best of our knowledge, there is no combinatorial model in the literature that describes the MCM problem. In this work we develop such model using the subexpression space concept proposed in [5]. The individual subexpression spaces of coefficients and subexpressions are combined into one large space resulting in an acyclic directed graph called the decomposition graph. In this way, subexpressions are shared by common paths which makes the problem of minimizing of the MCM as a graph traversal. However, to be able to traverse the decomposition graph, it must be transformed into a demand graph by transforming the shift operation over arcs into demand of subexpressions. Traversing an arc will cost a number of adders determined by the complexity of the demand subexpression attached to this arc. Each traversed arc becomes a deadheading which means there is no cost when visiting this arc in the subsequent time steps [19]. Additional deadheading arcs can be added to the demand graph to make it a directed cyclic graph, which is called the augmented demand graph. Solutions to the MCM problem are found by constructing tours on the augmented demand graph. Each possible tour starts from the signal vertex with value 1 and should passes by all unsynthesized coefficients. If one tour passed by an arc with unsynthesized demand, this tour should be redirected toward the closest synthesized ancestor of this demand. This behavior of redirecting the tour belongs to the class of arc routing problem that called Dynamic Capacitated Arc Routing Problems (DCARP) in which the rout is re-planned during servicing [6] [18] [27] [35]. An example of the DCARP is the dynamic winter gritting problem [27]. Ant Colony Optimization (ACO) technique is found as an efficient metaheuristic method that can be used to solve such problems [7] [34] and to search the graph in parallel [36]. In this case, each ant constructs one possible complete tour by selecting arcs with minimum complexity and maximum sharing with other ants. Sharing of an arc between different ant tours is measured by the accumulated pheromone amount on this arc. This is similar to the real life ants behavior in which ants deposit pheromone on their path from the nest to the food source. This pheromone is accumulated by time over shortest paths and evaporated from the longer paths. This natural behavior is simulated in the ACO by letting each ant deposit on its tour a pheromone quantity that is proportional with the quality of its solution. Pheromone evaporation is applied equally likely over all the arcs.

The paper is structured as follows: The derivation of the MCM combinatorial model is described in Section 2. Arc routing problems are discussed in Section 3. Constructing tours over the demand graph is shown in Section 4. Ant colony optimization metaheuristic is described in Section 5. Finding the heuristic parameters for MCM problem is described in Section 6. The results of implementing of the ACO are shown in Section 7. Conclusions are given in Section 8.

2. Decomposition, demand, and augmented demand graphs

To develop the combinatorial model of the MCM operation, consider the coefficient set $W = \{5, 37, 47\}$. The decomposition graph is started with the vertices 1, 5, 37, and 47 as shown in Fig. 6 (a). The vertex with value 1 is the signal vertex. The binary signed digit (BSD) subexpression tree of 47 is generated as shown Fig. 5. The tree grows from the root node of value 47, which is at depth 0, towards leaves at depth $L = 6$ (from the least significant digit (LSD) toward the most significant digit (MSD)). Tree growth can be expressed by the recursion [5] [30]

$$v_c = \frac{v_p - d}{r}, \tag{2}$$

where v_p is the parent node value, v_c is the child node value, and d is the edge value between them. The modulo operation $v_p \equiv d \bmod r$ partitions the digit set $\mathbb{D}$ into r disjoint sets [5]. Digits will be released from their disjoint sets when the modulo operation is applied at nodes. For BSD, the digit set $\mathbb{D} = \{\bar{1}, 0, 1\}$ is partitioned into two disjoint sets, $\mathbb{D}_0 = \{0\}$ and $\mathbb{D}_1 = \{\bar{1}, 1\}$ [5]. Starting from 47, we have $47 \equiv d_0 \bmod 2$, where d_0 is the LSD and $47 \equiv d_0 \bmod 2$ means 47 is congruent to d_0 modulo 2. The digits $\bar{1}$ and 1 are released from the set $\mathbb{D}_1 = \{\bar{1}, 1\}$ to satisfy Equation 2. This is represented in Fig. 5 by two edges emitted from 47 with values -1 and 1 respectively. Left and right children are calculated using Equation 2 to give nodes $v_c = \frac{47-(-1)}{2} = 24$ and $v_c = \frac{47-1}{2} = 23$ respectively. Consider the right branch of the tree of 47 shown in Fig. 5. The digit $d_0 = 1$ is considered as one co-prime part (subexpression) named by s_1 of the two-part decomposition of the value 47. The other part of decomposition is called s_2. The general $\mathcal{A}-$operation for decomposing w into two parts s_1 and s_2 is given by

$$\mathcal{A}_w(s_1, s_2) = |s_1 \pm s_2 \ll i|, \tag{3}$$

$s_2 \ll i$ represents the shift of s_2 to the left by i positions. Applying the $\mathcal{A}-$operation to the value 47 with $s_1 = 1$ (known) results in $47 = 1 + s_2 \ll i$. Therefore, the subexpression s_2 is calculated as $s_2 \ll i = 47 - 1 = 46$, or alternatively $s_2 = 46 \ll -i$. The value i represents the required right shift for 46 to become a co-prime number. In this case, $i = 1$ resulting in $s_2 = 23$. Therefore, the decomposition set of 47 at node 23 is given by $P_{47}^{23} = \{(1, 23)\}$ as shown in Fig. 5. The subexpression set at node 23 is given by $S_{47}^{23} = \{1\}$. Adding the vertex 23 to the decomposition graph and connecting it with the vertex 1 to compose the coefficient 47 results in Fig. 6 (b). In this figure, the coefficient 5 is of adder step $\delta_5 = 1$ so it can be synthesized from the vertex 1 using one adder. The next level released digit of the same branch is 1 which is of weight 2. The new subexpression set is found as [5]

$$S_w^c = S_w^p \bigcup \{r^l \times d\} \bigcup \{s_1 + r^l \times d, \forall s_1 \in S_w^p\}, \tag{4}$$

where w is the coefficient being decomposed, l is the tree depth, d is the edge connecting child node with its parent, and S_w^p and S_w^c are the parent and child subexpression sets, respectively. In this case, the subexpression set is given by $S_{47}^{11} = S_{47}^{23} \bigcup \{2^1 \times 1\} \bigcup \{1 + 2^1 \times 1\} = \{1, 2, 3\}$ and the decomposition set is given by $P_{47}^{11} = \{(1, 23), (3, 11)\}$

Adding the vertices 3 and 11 to the decomposition graph shown in Fig. 6 (b) gives Fig. 6 (c). The subexpression 3 is of adder step $\delta_3 = 1$ and can be synthesized from the vertex 1 using a minimum adder cost of 1 adder. Unsynthesized subexpressions with adder step $\delta > 1$ are moved to the coefficient set. In this case, the coefficient set becomes $W = \{37, 11, 23\}$. The procedure continues until the coefficient set W becomes empty. The resulting decomposition graph consists of 24 nodes and 1174 edges. A subgraph of the resulting decomposition graph is shown in Fig. 7 (a). In this figure, arcs with the same color represent the inputs of the $\mathcal{A}-$operation given in Equation 3. For example, the green pair that enters vertex 47 represents the decomposition $47 = 31 \ll 1 - 15 = 62 - 15$.

To make the decomposition graph shown in Fig. 7 (a) unicursal (routable) [19], a transformation is applied to the $\mathcal{A}-$operation. The transformation steps are shown in Fig. 9. The graph representation for the $\mathcal{A}-$operation is shown in Fig. 9 (a). In this case, arcs' attributes are the shift information. Fig. 9 (b) combines the addition vertex (+) with the output of the $\mathcal{A}-$operation (w). The final step of the transformation is shown in Fig. 9 (c) which includes replacing the shift information with the subexpression information. The subexpression information is considered as a demand on the traversed arc. Thus, traversing an arc from vertex s_1 to w requires synthesizing (servicing) vertex s_2. Similarly, traversing the arc from vertex s_2 to w requires servicing vertex s_1.

Applying the transformation shown in Fig. 9 to the decomposition graph shown in Fig. 7 (a) results in another graph called the demand graph as shown in Fig. 7 (b). Crossing unsynthesized arc in this graph will not synthesize the target vertex completely. It is required to return back to the nearest synthesized ancestors of the demand to synthesize it and then complete synthesizing the target vertex. To make the demand graph routable requires augmenting it with deadheading [19] arcs of zero cost as shown by the dashed arcs in Fig. 7 (c), resulting in the augmented demand graph. To find the optimal solutions of MCM, it requires to construct tours that start from

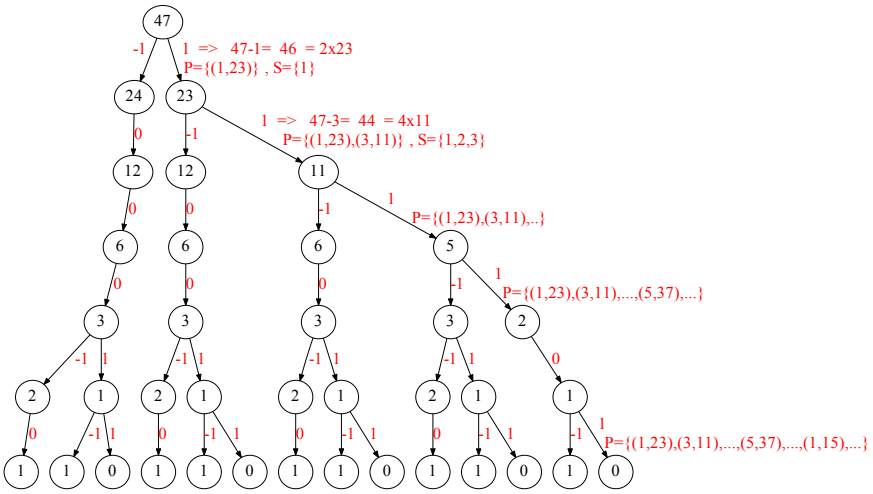


Figure 5: The subexpression tree of the coefficient 47.

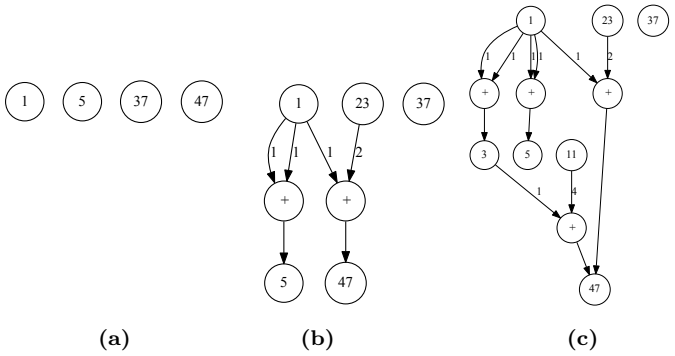


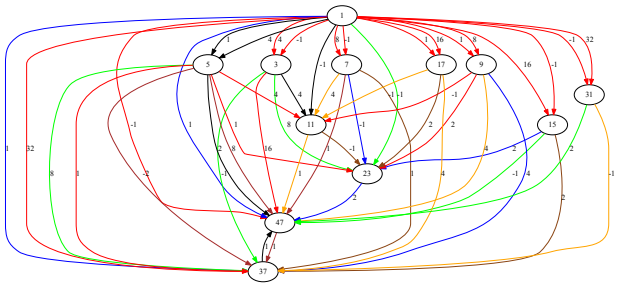
Figure 6: Steps show growing the decomposition graph of the coefficients 37 and 47. (a) Initial graph. (b) Composing 47 from the co-prime pair $(1, 23)$. (c) Add the decomposition pair $(3, 11)$ of 47. All the subexpressions of $\delta = 1$ are composed from the pair $(1, 1)$.

vertex 1 and synthesizes all the coefficients as mentioned in Section 1. Each constructed tour represents one solution of the MCM problem. This sort of graph routing is called arc routing because the routing problem regards the arcs [18] [17], not the nodes. Therefore MCM belongs to the arc routing category (this will be shown in Section 4). A literature review about some of common arc routing problems is given in the following section.

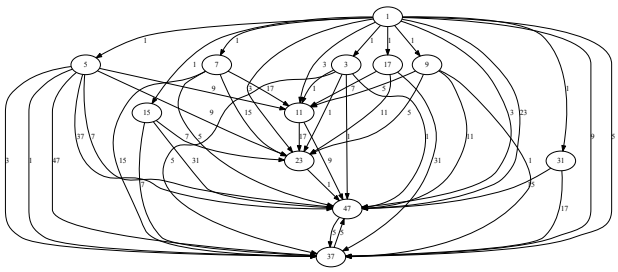
3. Arc routing problems

Graph routing problems are classified into two main categories, which are vertex (node) routing and arc (edge) routing problems [19]. Vertex routing includes problems where service (demand) occurs at the vertices of the graph. Examples include traveling salesman, traveling tourist, etc [19]. Arc routing includes problems where the service occurs on the graph arcs. Examples include street maintenance, garbage collection, milk or mail delivery, school bus tour, electric meter reading, and winter gritting [17] [18].

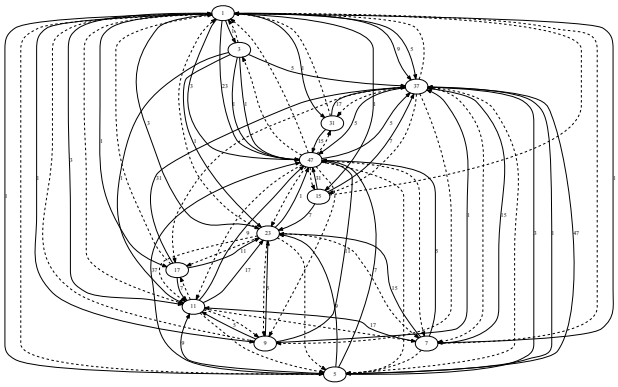
Arc routing problem is classified as Capacitated Arc Routing Problem (CARP) when some of the graph arcs have zero demand. In this case, only a subset $R \subseteq A$ of the graph arcs require servicing, where A is the arc set in the graph G. Other arcs that do not require servicing are used to deadhead between arcs that require service. In this case, deadheading corresponds to traveling on roads that do not require service. Usually deadheading is minimized. The CARP is NP-hard [19]. An example of the CARP is the rural postman problem (RPP) [18]. In the rural postman problem not all street segments require service. Here, the postman traverses deadheading streets to arrive at the street segments to be serviced. The undirected and the directed rural postman problems are NP-hard so heuristics are required to solve the problem. Another example of the CARP is the winter gritting or salting



(a)



(b)



(c)

Figure 7: (a) A subgraph of the decomposition graph of the coefficients 37 and 47. A pair of similar color arcs entering a vertex represents a decomposition. (b) Applying the transformation in Fig. 9 on the decomposition subgraph transforms it to the demand subgraph. (c) Augmenting the demand graph with deadheading arcs.

problem described in the next section.

3.1. Dynamic Winter Gritting (Salting) Problem

When roads become hazardous due to ice or snow, a de-icing agent (usually salt) is spread on them for safety reasons. The problem of winter gritting is how to design routes for a fleet of vehicles that depart from a depot to minimize the costs. A depot here is where the vehicles are loaded with salt. This problem belongs to the CARP, therefore it is an NP-hard problem [31] [17] [18]. Many constraints may be imposed on the driver such as time, different priorities of roads, and limited vehicle capacity. However, in snowy weather, the most significant problem is in the priority of which roads to grit [27]. A wrong decision of spreading salt on roads that do not require service is a financial loss. On the other hand, untreated roads are a major hazard. Therefore the decision should be taken on dynamic basis. This means that the driver may re-plan the route according to newly received weather forecast information. The route in this case become dynamic and using static heuristics may be insufficient to find a solution.

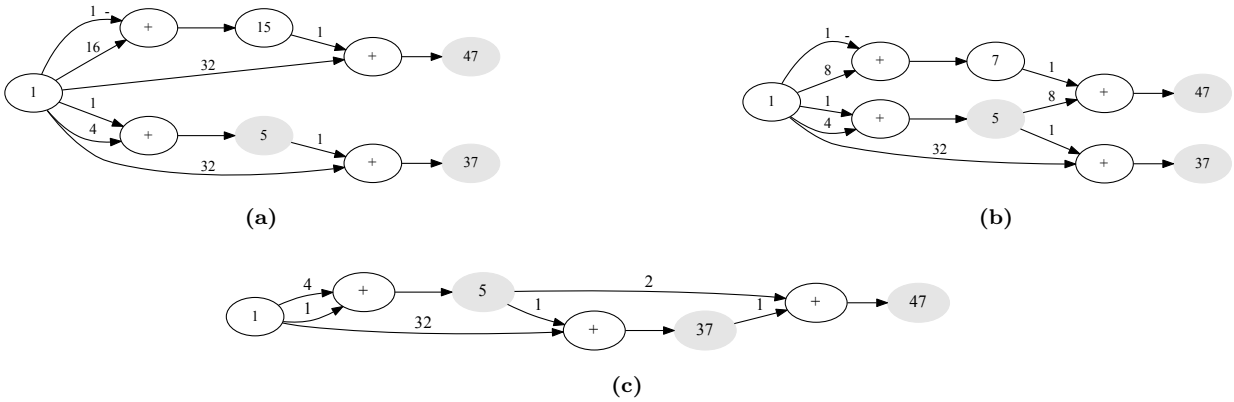


Figure 8: Three possible realizations for the coefficients $\{5, 37, 47\}$.

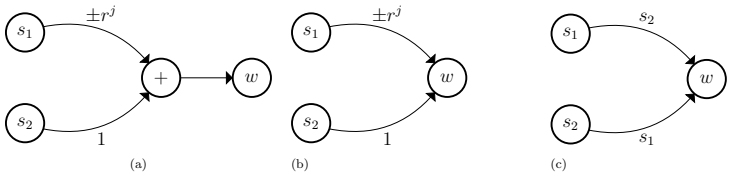
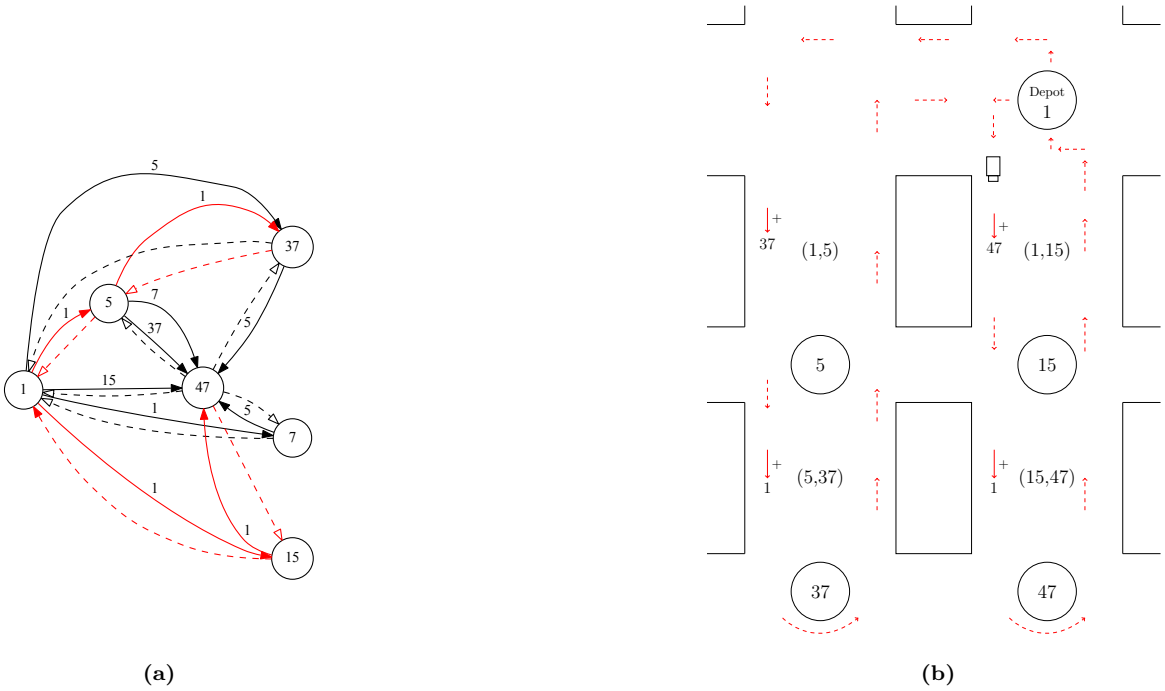


Figure 9: Transforming the shift information to a subexpression information. (a) Representing the $\mathcal{A}-$operation where the arcs carry shift information. (b) Combining the addition vertex with the output vertex. (c) A new representation for the two part decomposition.

4. Dynamic winter gritting analogy

We have shown in Section 2 that the MCM problem can be modeled by a graph called the demand graph. The existence of demands on the demand graphs' arcs can be analogized to the dynamic arc routing problems [18] [27] [35] such as the dynamic winter gritting problem [27]. Consider the MCM example shown in Fig. 8. One of the possible tours on the augmented demand graph is shown in Fig. 10 (a). This tour results in the realization shown in Fig. 8 (a) with logic operator LO = 4 and logic depth constraint of LD = 2. The analogy of the tour in Fig. 10 (a) is the winter gritting route shown in Fig. 10 (b). In this case, a vehicle departs from the depot to service the roads $5, 37$, and 47. The blocks are just for illustration and not necessary represent city blocks. To make the things simple, the capacity of the vehicle is assumed to be 20 tons and each road consumes 10 tons of salt. Assuming that the cost correspondence between the salt weight and adder cost is that each 10 tons $\equiv$ 1 adder. The correspondence between the adder depth constraint and vehicle capacity is that violating the constraint LD = 2 is equivalent to violating the vehicle capacity which is 20 tons. The depot in Fig. 10 (b) corresponds to the signal vertex shown in Fig. 10 (a) which labeled by the number 1. Assume that due to a change in the weather forecast, road 15 becomes more hazardous than other roads and requires urgent servicing. Therefore, the station would signal to the driver to service road 15 first. The driver in this case should re-plan his/her route and go to service road 15. Again for simplicity and to illustrate the concept, road 15 is draw to be next to the depot and in the reality the deriver could drive a long distance from the depot when he/she received the redirection. The corresponding movement on the graph shown in Fig. 10 (a) is to go from vertex 1 to synthesize coefficient 15 as shown by the red arc $(1, 15)$. The red arrow in Fig. 10 (b) illustrates this service which is labeled by $(1, 15)$. The cost of servicing road 15 equals 10 tons which is indicated by the plus sign on the direction (arrow) of the vehicle. The plus sign here is introduced to keep the corresponding with the augmented demand graph. After servicing road 15, the driver returns to the original plan and finds that road 47 is the closest one to his/her position as shown in Fig. 10 (b). Servicing road 47 from the end of 15 is labeled by $(15, 47)$ which consumes another 10 tons. The corresponding movement on Fig. 10 (a) is to synthesize 47 from 15 resulting LD = 2. At this point the vehicle is empty and the deriver cannot continue to serve other roads. Therefore, the driver decides to return to the depot through the deadheading roads 47 and 15 indicated by the red dashed lines shown in Fig. 10 (b). These roads are considered deadheading because they have been serviced. A similar situation of breaching logic depth constraint arises when try synthesizing the coefficients 5 and 37 from 47. Therefore, the corresponding movement on the graph shown in Fig. 10 (a) is to return over the dashed red arcs $(47, 15)$ and $(15, 1)$ respectively. These arcs are also deadheading because their attached subexpressions are already synthesized. The winter gritting route is now (depot $\rightarrow$ 15 $\rightarrow$ 47 $\rightarrow$ 15 $\rightarrow$ depot) which

is corresponding to the tour $(1,15),(15,47),(47,15),(15,1)$ on the demand graph in Fig. 10 (a). After reloading the vehicle at the depot, road 5 is serviced with 10 tons of salt followed by servicing road 37 by another 10 tons of salt. This is correspond to synthesize 5 from 1 in Fig. 10 (a), then synthesize 37 from 5. The resulting logic depth after synthesizing 37 equals 2 adders which will not violate the logic depth constraint. On the other hand, the winter gritting route ends after finishing road 37 and the driver returns to the depot via the deadheading arcs. The vehicle route is now (depot $\rightarrow$ 15 $\rightarrow$ 47 $\rightarrow$ 15 $\rightarrow$ depot $\rightarrow$ 5 $\rightarrow$ 37 $\rightarrow$ 5 $\rightarrow$ depot). The corresponding tour on the graph in Fig. 10 (a) is $((1,15),(15,47),(47,15),(15,1),(1,5),(5,37),(37,5),(5,1))$. This is a feasible tour on the demand graph because it will not violate the adder depth constraint. The resulting LO of this tour equals 4 adders. Fig. 11 (a) shows another possible tour on the augmented demand graph which results in the realization shown in Fig. 8 (b) with LO = 4 under the constraint LD = 2. Obtaining realizations with smaller LO is possible if the LD constraint is relaxed (increased). This can be illustrated by using the same example of the coefficients $5, 37$, and 47 when LD is allowed to be of value of 3 adders. The corresponding tour on the demand graph is illustrated in Fig. 11 (b) which results in the realization shown in Fig. 8 (c) with LO = 3 and LD = 3. The same realization would be obtained from using the graph dependent method.



(a) **(b)**

Figure 10: (a) A possible tour on the augmented demand graph results in the realization shown in Fig. 8 (a). (b) The corresponding winter gritting route. A solid arrow with plus sign indicates the salting action and the label preceding each arrow is the next destination after this road. A dashed arrow means traversing a deadheading road. The roundabouts are labeled by the road number and the road is labeled by the path between two roundabouts.

5. Ant colony optimization

Ant colony optimization (ACO) metaheuristics proposed by Dorigo [10] to solve large combinatorial optimization problems. The algorithm is inspired by the swarm intelligence of a real life ant colony when searching for food. Each ant excretes a chemical factor called pheromone and deposits it on its trail. The quantity of pheromone on the shortest paths to the food is reinforced by the successive passing of other ants. On the other hand, the pheromone on the longer paths evaporates during the time. Ants choose the path with high pheromone concentrations, increasing the pheromone concentration and encouraging most of subsequent ants select that path too. A minority of ants selects alternative paths. The behavior of this minority is important because these ants continue searching for better solutions [34] [38].

ACO algorithms use artificial ant agents that search the graph concurrently. Each artificial ant builds its solution by constructing a tour on the graph making ACO algorithms to be easily amendable to parallel computation [36] [48]. Each ant tour starts from the source vertex and ends when all the coefficients have been synthesized. In their touring, ants use the information of pheromone quantity and heuristic values that are stored in the arcs' attributes to decide

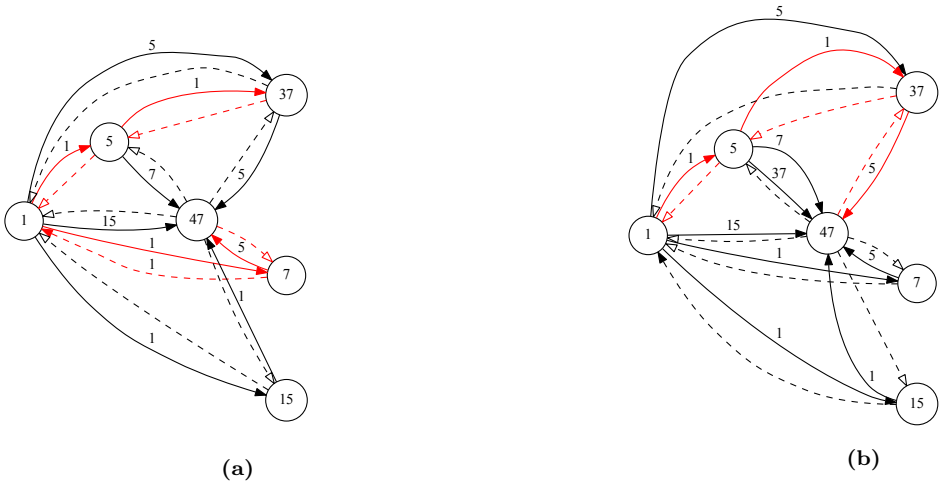


Figure 11: (a) A possible tour on the augmented demand graph results in the realization shown in Fig. 8 (b). (b) A possible tour on the augmented demand graph results in the realization shown in Fig. 8 (c).

the next arc to be traversed. Heuristic values could be the length of an arc as in CARP problems. Pheromone levels are initialized to the same value. Through successive iterations arcs with low heuristic values and stronger pheromone quantities become more attractive. After all the ants finish their tours, the pheromone levels are reduced according to some evaporation formula. Then each ant deposits a pheromone value on the arcs of its tour. The deposited pheromone value is proportional to the quality of the solution found by the corresponding ant. Pheromone evaporation and depositing continue until some ants find an optimum solution.

There are four basic ACO algorithms and other algorithms are derived from them. These algorithms are the Ant System (AS), MAX-MIN Ant System (MMAS), Elitist Ant System (EAS), and Rank based Ant System [15] [14]. The MMAS is chosen as the most suitable for the MCM as it prevents early stagnation (all the ants follow a particular tour). However, the MMAS itself is derived from the AS algorithm so the latter is discussed first in the next section.

5.1. Ant System

In the AS, m artificial ants are initially distributed randomly on chosen graph vertices. Before the ants start their first tour, the pheromone's level are initialized. The initial value of the pheromone, τ_0, should not be too high, otherwise the search is quickly biased by the first tours generated by the ants [10]. On the other hand, choosing τ_0 too low may slow convergence of the algorithm. After depositing τ_0 on all the graph arcs, ants start their movement on the graph. The ants construct their solution such that each one determines its next move according to a probabilistic formula. An ant k at vertex v_i moves to the adjacent vertex v_j with probability $p^{(k)}_{(v_i,v_j)}$ given by

$$p^{(k)}_{(v_i,v_j)} = \frac{\tau^{\alpha}_{(v_i,v_j)}\eta^{\beta}_{(v_i,v_j)}}{\sum_{l\in V^k_{v_i}}\tau^{\alpha}_{(v_i,v_l)}\eta^{\beta}_{(v_i,v_l)}} \quad \text{if} \quad v_j \in V^{(k)}_{v_i}, \tag{5}$$

$$= \quad 0 \quad \text{if} \quad v_j \notin V^{(k)}_{v_i}, \tag{6}$$

where $\tau_{(v_i,v_j)}$ is the strength of the pheromone on the arc (v_i, v_j), $\eta_{(v_i,v_j)}$ is a heuristic related to the cost of traversing arc (v_i, v_j), α and β are two parameters which determine the relative influence of the pheromone trail and the heuristic information, and $V^{(k)}_{v_i}$ is the set of vertices that ant k has not yet visited from vertex v_i. Equation 5 is interpreted as follows. The probability of ant k crossing from vertex v_i to vertex v_j increases with the value of pheromone trail $\tau_{(v_i,v_j)}$ on this arc and with the heuristic value $\eta_{(v_i,v_j)}$ associated with this arc [15].

After each ant has constructed (finished) its tour, a pheromone update cycle starts with the pheromone evaporation

$$\hat{\tau}^{(n)}_{(v_i,v_j)} = (1-\rho)\tau^{(n-1)}_{(v_i,v_j)}, \ \forall (v_i, v_j) \in A, \tag{7}$$

where $\hat{\tau}^{(n)}_{(v_i,v_j)}$ is the pheromone strength after evaporation, $\tau^{(n-1)}_{(v_i,v_j)}$ is the pheromone strength at the previous iteration, $0 < \rho \leq 1$ is the pheromone evaporation rate and A is the arc set. After evaporation, all the ants deposit

pheromone on the arcs they have crossed in their tour according to

$$\tau^{(n)}_{(v_i,v_j)} = \hat{\tau}^{(n)}_{(v_i,v_j)} + \sum_{k=1}^{m} \Delta\tau^{(k)}_{(v_i,v_j)}, \ \forall (v_i, v_j) \in A, \tag{8}$$

where $\tau^{(n)}_{(v_i,v_j)}$ is the new pheromone strength, and $\Delta\tau^{(k)}_{(v_i,v_j)}$ is the pheromone quantity deposited by ant k on the arcs visited by this ant. This value is chosen to reflect the solution quality of ant k. For example, if ant k found a solution with cost R_k, then a possible selection of $\Delta\tau^{(k)}_{(v_i,v_j)}$ is

$$\Delta\tau^{(k)}_{(v_i,v_j)} = \begin{cases} \frac{1}{R_k}, & \text{if arc } (v_i, v_j) \text{ belongs to the tour of ant } k; \\ 0, & \text{otherwise.} \end{cases} \tag{9}$$

5.2. Max-Min Ant System

The Max-Min Ant System (MMAS) is a modification to AS introduced to prevent early stagnation of the algorithm in which all the ants follow the same tour [42]. The algorithm allows either the best-so-far ant (the ant that produced the best tour since starting the iterations) or the best-iteration ant (the ant that produced the best tour in the current iteration) to deposit pheromone. The pheromone level is limited to the range $[\tau_{\min}, \tau_{\max}]$. The pheromone levels are initialized to $\tau_{\max}$. If the algorithm reaches a stagnation state or there is no improvement in the solution, the pheromone level is reinitialized. The value of $\tau_{\max}$ is estimated from the best-so-far (bs) tour solution, i.e. $\tau_{\max} = \frac{1}{R_{\text{bs}}}$, where R_{bs} is the cost of the best-so-far tour. In other words, the value of $\tau_{\max}$ is updated each time a new best-so-far solution is found. The lower pheromone levels is estimated to $\tau_{\min} = \frac{\tau_{\max}}{b}$, where b is a parameter that can be tuned to prevent the stagnation [15]. The equation of pheromone evaporation is the same as Equation 7, while pheromone updating is given by

$$\tau^{(n)}_{(v_i,v_j)} = \tau^{(n-1)}_{(v_i,v_j)} + \Delta^{\text{best}}_{(v_i,v_j)}, \tag{10}$$

where

$$\Delta^{\text{best}}_{(v_i,v_j)} = \begin{cases} \frac{1}{R_{\text{bi}}}, & \text{if best-iteration tour is chosen} \\ \frac{1}{R_{\text{bs}}}, & \text{if best-so-far tour is chosen,} \end{cases} \tag{11}$$

where R_{bi} is the cost of the best-iteration tour. The algorithm alternates using the best-so-far and the best-iteration updating rules. The frequency of alternating between these rules is related to the problem size. It has been shown that for small combinatorial problems the algorithm gives better results when using only best-iteration tours. With larger combinatorial problems, the best-so-far tour gives better results [15]. When some of pheromone level reach the value of minimum pheromone level $\tau_{\min}$ or maximum levels $\tau_{\max}$ [15], the algorithm re-initialize these levels to τ_0 which is calculated according to

$$\tau_0 = \frac{\tau_{max}}{c}, \tag{12}$$

where $\tau_{max} = \frac{d}{(\rho R_{\text{bs}})}$ and $\tau_{min} = e\frac{\tau_{max}}{d}$. And where c, d, and e are tunable parameters.

6. Heuristic parameters for the MCM problem

To apply ACO to the MCM problem it is necessary to modify Equation 5 to take into account of the routing over the demand graph. In the demand graph each vertex is a subexpression, so that the vertex notation in Equation 5 is changed from v_i to s_i. The other important modification is that Equation 5 considers the case of searching a graph with no parallel arcs while the demand graph is a multigraph with parallel arcs. In this case, each ant choses to cross one of the arcs $A_{(s_i,s_j)} = \{(s_i, s_j)_1, (s_i, s_j)_2, \cdots, (s_i, s_l)_1, (s_i, s_l)_2, \cdots\}$ that depart from subexpression s_i. Therefore the probability that an ant k crosses from s_i to s_j using arc $(s_i, s_j)_h \in A_{(s_i,s_j)}$ is given by

$$p^{(k)}_{(s_i,s_j)_h} = \frac{\tau^{\alpha}_{(s_i,s_j)_h} \chi_{(s_i,s_j)_h}{}^{-\beta}}{\sum_{s_l \in V^{(k)}_{s_i}, (s_i,s_l)_n \in A_{(s_i,s_l)}} \tau^{\alpha}_{(s_i,s_l)_n} \chi_{(s_i,s_l)_n}{}^{-\beta}}, \tag{13}$$

$$\text{if } s_j \in V^{(k)}_{s_i} \text{ and } (s_i, s_l)_n \in A_{(s_i,s_l)}$$

$$\text{for } n = 1, 2, \cdots$$

where $p^{(k)}_{(s_i,s_j)_h}$ is the probability of ant k crossing arc number h denoted by $(s_i, s_j)_h$, $\tau_{(s_i,s_j)_h}$ is the pheromone strength on arc $(s_i, s_j)_h$, $\chi_{(s_i,s_j)_h}$ is the complexity (heuristics value) of arc $(s_i, s_j)_h$, α and β are two positive parameters that determine the relative influence of the pheromone trail and the heuristic information, $A^{(k)}_{(s_i,s_l)}$ is feasible arc set which does not violate the logic depth constraint, and $V^k_{s_i}$ is the set of unvisited vertices for ant k at vertex s_i.

Assume an ant wants to cross the arc $(s_i, s_j)_h$. The complexity of the arc is given by

$$\chi_{(s_i,s_j)_h} = \pi + L_{\max}\mathcal{H}_{s_h} + L_{s_h} \tag{14}$$

where $L_{\max}$ is the maximum wordlength, L_{s_h} and $\mathcal{H}_{s_h}$ are respectively the wordlength and Hamming weight of the demand s_h, and π is the priority of crossing arc $(s_i, s_j)_h$ which is calculated according to Table 1. In this table, W is the set of coefficients waiting for synthesis, T is the set of traversed vertices (synthesized subexpressions), and a and b are constants used to determine the priority parameter π in Equation 14. The parameters in Equation 14 are chosen to favor the priority π over the Hamming weight $\mathcal{H}_{s_h}$ and the latter over the wordlength L_{s_h}. A low priority number is more desirable. If an ant compares two arcs with the same priority, it choses to cross the one with a demand of smallest Hamming weight. If the Hamming weights are the same, the arc with short wordlength is preferred. The priority values are chosen to encourage the ants to search for the coefficients (food). The minimum value of π should be such that

$$\frac{a}{b^5} > L_{\max}\mathcal{H}_{s_h} + L_{s_h}, \tag{15}$$

to dominate the second and third terms in Equation 14. The maximum possible Hamming weight is $\mathcal{H}_{\max} = L_{\max}$. Therefore, Equation 15 becomes

$$\frac{a}{b^5} > L^2_{\max} + L_{\max}. \tag{16}$$

Table 1: Priority values of the demand s_h associated with arc $(s_i, s_j)_h$. W is the set of subexpressions waiting for synthesis, T is the set of synthesized subexpressions, a and b are constants. π is the priority of crossing an arc. A low priority number is more desirable.

s_h	s_j	π
T	W	0
W	W	$\frac{a}{b^5}$
$\notin W \cup T$	W	$\frac{a}{b^4}$
T	$\notin W \cup T$	$\frac{a}{b^3}$
W	$\notin W \cup T$	$\frac{a}{b^2}$
$\notin W \cup T$	$\notin W \cup T$	$\frac{a}{b}$

7. Implementation of the ACO

7.1. Implementation

The MMAS ant system was implemented for constructing tours on the demand graph using Multi-threading coding with OpenMP and Visual C++. The code was run on a platform with 3.6 GHz Quad Core CPU and 8 GB of RAM. The demand graph is implemented using the Boost Graph Library (BGL) which is a C++ template graph library [40] [16]. The demand graph is generated in two stages to reduce the computation time. Firstly, the representation tree of each subexpression is generated. Secondly, each representation tree is traversed using the depth first search (DFS) algorithm [16]. The DFS is modified to work as a subexpression tree algorithm as described in Section 2. The C++ Eigen template library for linear algebra was used to handle fast vector operations such as calculating the subexpressions [22].

The demand graph is used as a shared data structure between the threads. Each thread implements a computational ant which searches the demand graph independently. It is not necessary to add actual deadheading arcs

to the demand graph and increase the storage requirement. The augmentation is implemented during ant tour by permitting ants to backtracking to the desired ancestor nodes. Listing 1 shows the implementation of the MMAS algorithm. Steps 9-12 show that each ant is assigned a rank. The ants' tours are constructed by distributing them over the vertices with adder step $\delta = 1$. At each new iteration a random shuffle is applied at the departure subexpressions (step 8) to prevent possible stagnation. If an ant reaches vertex v_k, it starts calculating its next move by determining the arc with minimum heuristic value and maximum pheromone level. It does this by calling a 'sort' function that sorts the out arcs of v_k in ascending order of complexity which is determined from the heuristic value and the pheromone level. These arcs are pushed onto the stack in this order (step 19). Therefore an arc with minimum complexity will be at the top of stack. If an ant is chosen to cross an arc with unsynthesized demand (as shown in steps 28 and 32 of Listing 1), a predecessor map is searched recursively (steps 29 and 33) until finding the first synthesized ancestors. This requires pushing again the last popped arc onto the stack (step 30). At this point the tour is redirected to the ancestors of the demand as shown in steps 31 and 35 of Listing 1. This redirection is a dynamic behavior. The Synthesize function in step 25 checks the adder step of vertex w before an ant moves beyond this vertex. When an ant reaches a vertex that violates the logic depth constraint, it is used to prune the corresponding path that led to this vertex. Pruning also includes paths that led to leaf subexpressions not in W (subexpressions waiting for synthesis). The costs of the ant tours are calculated and compared using the function Cost which is called in step 49 of Listing 1. This function returns the value of the best-so-far tour. Pheromone evaporation and updating functions are called in steps 50 and 51 respectively. Each function is implemented to loop over the graph arcs and change the pheromone levels according to the pheromone updating formulas given by Equations 16 and 17. Pheromone levels are reinitialized to prevent early stagnation as shown in step 54. The reinitializing occurs when the pheromone levels on one arc or more become greater than the specified upper limit $\tau_{\max}$, or when they become less than the specified lower limit $\tau_{\min}$. The new initial pheromone level equals the previous initial level plus a small increment Δ. The small increment is chosen to be the reciprocal of best-so-far tour length, i.e. $\Delta = 1/R_{bs}$.

7.2. Results

The parameters α and β were tuned experimentally. A set of 100 MCMs with $N = 15$ and wordlengths $8 \leq L \leq 20$ was used. The initial pheromone level was adjusted to the value $\tau_0 = 1$ (after several experiments) and the maximum pheromone level was set to $\tau_{\max} = 10$ and the minimum pheromone level was set to $\tau_{\min} = 10^{-4}$. The pheromone levels are re-initialized about every ten generations. The number of ant agents was set to equal the out-degree of vertex 1. The percentage of the number of ants that found the best-so-far tour was recorded against the values of α and β. The best values found are $\alpha = 1$ and $\beta = 2$, which coincides with the experiments of [15] (P71) for parameter values for MMAS.

The MMAS algorithm was used to synthesize the filters S_2 [39], L_1 [32], and D [13] and results are shown in Table 2. It is found that using 8 is sufficient in finding the minimum solution. The time required to generate the demand graph is denoted by t_g and that required to search the graph is denoted by t_s.

Table 2: Synthesis of the filters S_2 , L_1, and D using the parallel MMAS algorithm. t_g and t_s are the generating and the search times respectively.

Filter	vertices	edges	ants	t_g	t_s	LD	LO
S_2	45	503	22	3.5	2	2	28
D	649	861623	20	70	980	3	18
L_1	1203	2375596	24	420	3620	3	53

The MMAS algorithm was compared with the pattern preservation algorithm (PPA) [4], subexpression tree algorithm (STA) [5], difference based adder graph (DBAG) [23], and Yao [47], using a set of 100 MCM with $N = 5$. The wordlength was changed in steps starting from $L = 6$ to 15 and for each wordlength the average LO, LD, and time (t) are calculated as shown in Fig. 12. A minimum LO is obtained in the case of MMAS and DBAG as shown in Fig. 12 (a) at the expense of LD as shown in Fig. 12 (b). However, the MMAS succeed in obtaining realizations of relatively shorter LD than that of the DBAG method. The high quality of the solutions is found by the MMAS because it searches a large space of candidate solutions and there is a high probability of finding one or more optimal solutions (if they exist) as a result of using multi computational agents. A minimum LD constraint was used in the STA and Yao algorithms as shown in Fig. 12 (b). The saving in the case of STA algorithm is better

than that of the Yao and PPA algorithms. The LD was not constrained in the case of PPA but its adder saving was poorer than the others except Yao. The time complexity performance of the algorithm shown in Fig. 12 (c) reveals that the MMAS needs a longer run time than other algorithms except the PPA. This is because of the large space searched by the algorithm.

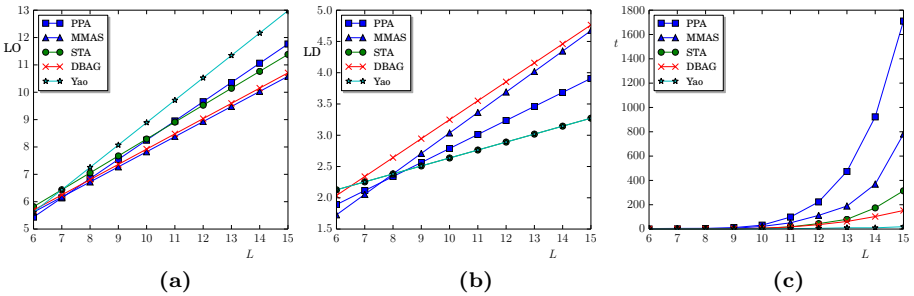


Figure 12: (a)The average number of LO versus wordlength obtained when the algorithms PPA [4], MMAS, STA [5], DBAG [23], and Yao [47] used to synthesize random MCMs of 5 coefficients. The LD was constrained to the minimum in the case of Yao and STA algorithms. (b) The average number of LD versus wordlength obtained when the algorithms PPA [4], MMAS, STA [5], DBAG [23], and Yao [47] used to synthesize random MCMs of 5 coefficients. The LD was constrained to the minimum in the case of Yao and STA algorithms. (c) Comparison between the run time of the algorithms PPA [4], MMAS, STA [5], DBAG [23], and Yao [47] used to synthesize random MCMs of 5 coefficients.

The experiments were repeated using a set of 100 MCM with $N = 10$ as shown in Fig. 13. The PPA algorithm was excluded from the comparison because of its relatively long run time as compared with the other algorithms.

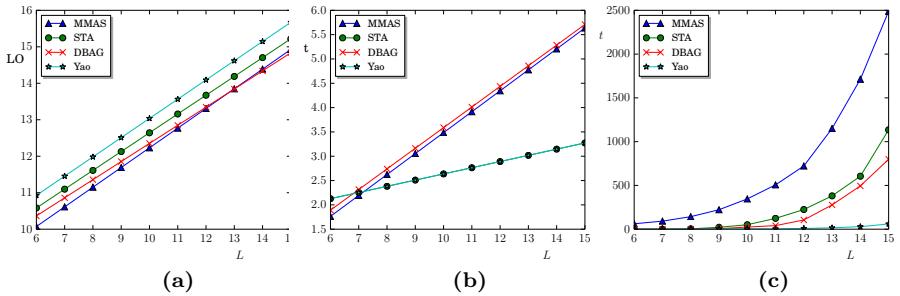


Figure 13: (a)The average number of LO versus wordlength obtained when the algorithms PPA [4], MMAS, STA [5], DBAG [23], and Yao [47] used to synthesize random MCMs of 10 coefficients. The LD was constrained to the minimum in the case of Yao, STA, and MMAS algorithms. (b) The average number of LD versus wordlength obtained when the algorithms PPA [4], MMAS, and DBAG [23] used to synthesize random MCMs of 10 coefficients. The LD was constrained to the minimum in the case MMAS algorithm. (c) Comparison between the run time of the algorithms PPA [4], MMAS, STA [5], DBAG [23], and Yao [47] used to synthesize random MCMs of 10 coefficients.

The MMAS algorithm was compared with the algorithm Hcub [46] found on [41]. A ten random MCMs were used, each of length $N = 10$ and with wordlengths $8 - 19$. The result of this comparison is shown in Fig. 14 which reveals that the MMAS algorithm performs better than the Hcub [46] algorithm. Adding to this the higher degree of freedom provided by the MMAS of obtaining multi minimum solutions.

8. Conclusions

A combinatorial model for the MCM problem is developed. The model is found to be a graph and called the demand graph. Constructing routes over the demand graph solves the MCM problem. Routing over the demand graph gives all the possible minimum realizations. This helps in choosing the one with the smallest wordlength in order to reduce the adder width.

Ant colony metaheuristic is proposed to find all the minimum solutions in the search space while avoiding the enumeration of all the possible routes. The solutions that obtained from using the traditional heuristics optimization

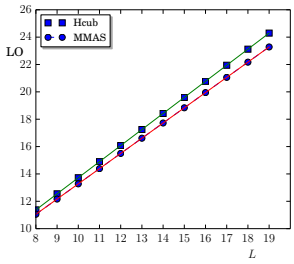


Figure 14: Comparing the MMAS algorithm with Hcub [46] algorithm used to synthesize random MCMs of 10 coefficients.

methods such as the CSE and GD methods become a subspace of the minimum solutions space. The results of using ACO are compared with several MCM optimization methods that found in the literature. It is found that the ACO has a better performance.

Now days there is a revolution in large scale graph data base such as twitter cassovary, GraphChi, Neo4j and many other libraries. Some of these data bases help to store and query graphs of sizes of billions of vertices and arcs on a PC. This opens the way to solve large sized MCMs problems in the near future which makes developing the combinatorial model of the MCM a major contribution in this field.

References

[1] Levent Aksoy, Eduardo Costa, Paulo Flores, and José Monteiro. Optimization of Gate-Level Area in High Throughput Multiple Constant Multiplications. *20th European Conference on Circuit Theory and Design, ECCTD 2011*, pages 588–591, Aug. 2011.

[2] Levent Aksoy, Eduardo Costa, Paulo Flores, and José Monteiro. Optimization Algorithms for the Multiplierless Realization of Linear Transforms. *ACM Transactions on Design Automation of Electronic Systems*, 17(1), Jan. 2012.

[3] Levent Aksoy, Ece Olcay Günes, and Paulo Flores. Search Algorithms for the Multiple Constant Multiplications Problem: Exact and Approximate. *Microprocessors and Microsystems*, 34(5):151–162, Aug. 2010.

[4] F. Al-Hasani, M. Hayes, and A. Bainbridge-Smith. A New Subexpression Elimination Algorithm Using Zero-Dominant Set. In *Proceedings - 2011 6th IEEE International Symposium on Electronic Design, Test and Application, DELTA 2011*, pages 45–50, Jan. 2011.

[5] F. Al-Hasani, M.P. Hayes, and A. Bainbridge-Smith. A Common Subexpression Elimination Tree Algorithm. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 60(9):2389–2400, Sept. 2013.

[6] Tetsuo Asano, Naoki Katoh, and Kazuhiro Kawashima. A New Approximation Algorithm for the Capacitated Vehicle Routing Problem on a Tree. *Journal of Combinatorial Optimization*, 5:213231, June 2001.

[7] M.H. Abolhasani Ashkezari, N. Shahsavari Pour, and H. Mohammadi Andargoli. An ant colony system for solving fuzzy flow shop scheduling problem. *International Journal of Engineering and Technology*, 1(2):44–57, Apr. 2012.

[8] Nilanjan Banerjee, Jung Hwan Choi, and Kaushik Roy. A Process Variation Aware Low Power Synthesis Methodology for Fixed-Point FIR Filters. In *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 147–152, Aug. 2007.

[9] D.R. Bull and D.H. Horrocks. Primitive operator digital filters. *IEE Proceedings, Part G: Circuits, Devices and Systems*, 138(3):401–412, June 1991.

[10] Alberto Colorni, Marco Dorigo, and Vittorio Maniezzo. Distributed Optimization by Ant Colonies. *European Conference on Artificial Life*, pages 134–142, 1991.

[11] Kenneth De Jong. *Evolutionary computation: A unified approach*. MIT Press, 1 edition, 2006.

[12] A.G. Dempster, M.D. Macleod, and O. Gustafsson. Comparison of Graphical and Subexpression Elimination Methods for Design of Efficient Multipliers. *Thirty-Eighth Asilomar Conference on Signals, Systems and Computers*, 1:72–76, Nov. 2004.

[13] Andrew G. Dempster and Malcolm D. Macleod. Use of Minimum-Adder Multiplier Blocks in FIR Digital Filters. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 42(9):569–577, Sep. 1995.

[14] K.F. Doerner, R.F. Hartl, G. Kiechle, M. Lucka, and M. Reimann. Parallel Ant Systems for the Capacitated Vehicle Routing Problem. *Evolutionary Computation in Combinatorial Optimization: Lecture Notes in Computer Science*, 3004:72–83, 2004.

[15] Marco Dorigo and Thomas Stützlea. *Ant Colony Optimization*. MIT Press, 1 edition, 2004.

[16] Nick Edmonds, Douglas Gregor, and Andrew Lumsdaine. Parallel boost graph library, 2009.

[17] H.A. Eiselt, Michel Gendreau, and Gilbert Laporte. Arc Routing Problems, Part I: The Chinese Postman Problem. *Operation Research*, 43(2):231–242, March-April 1995.

[18] H.A. Eiselt, Michel Gendreau, and Gilbert Laporte. Arc Routing Problems, Part II: The Rural Postman Problem. *Operation Research*, 43(3):399–414, May-June 1995.

[19] James R. Evans and Edward Minieka. *Optimization Algorithms for Networks and Graphs*. Maecel Dekker Inc., 2 edition, 1992.

[20] M.A. Farahani, E.C. Guerra, and B.G. Colpitts. Efficient Implementation of FIR Filters Based on a Novel Common Subexpression Elimination Algorithm. *Canadian Conference on Electrical and Computer Engineering*, pages 1–4, May 2010.

[21] Mathias Faust and Chip-Hong Chang. Minimal Logic Depth Adder Tree Optimization for Multiple Constant Multiplication . *2010 IEEE International Symposium on Circuits and Systems: Nano-Bio Circuit Fabrics and Systems, ISCAS 2010*, pages 457–460, May 2010.

[22] Free Software. Eigen 3.2.0, 2013.

[23] Oscar Gustafsson. A Difference Based Adder Graph Heuristic for Multiple Constant Multiplication Problems. In *Proceedings - IEEE International Symposium on Circuits and Systems*, pages 1097–1100, May 2007.

[24] Oscar Gustafsson. Lower Bounds for Constant Multiplication Problems. *IEEE Transactions on Circuits and Systems-II: Express Briefs*, 54(11):974–978, Nov. 2007.

[25] Oscar Gustafsson. Towards Optimal Multiple Constant Multiplication: A Hypergraph Approach. *Asilomar Conference on Signals, Systems and Computers*, pages 1805–1809, Oct. 2008.

[26] Oscar Gustafsson, Andrew G. Dempster, Kenny Johansson, Malcolm D. Macleod, and Lars Wanhammar. Simplified Design of Constant Coefficient Multipliers. *Circuits, Systems, and Signal Processing*, 25(2):225–251, April 2006.

[27] Hisashi Handa, Lee Chapman, and Xin Yao. Dynamic Salting Route Optimisation Using Evolutionary Computation. *2005 IEEE Congress on Evolutionary Computation, IEEE CEC 2005*, pages 158–165, Sep. 2005.

[28] Richard I. Hartley. Optimization of Canonic Signed Digit Multipliers for Filter Design. In *Proceedings - IEEE International Symposium on Circuits and Systems*, volume 4, pages 1992–1995, June 1991.

[29] Yuen-Hong Alvin Ho, Chi-Un Lei, Hing-Kit Kwan, and Ngai Wong. Optimal Common Subexpression Elimination of Multiple Constant Multiplications with A Logic Depth Constraint. *IEICE Trans. Fundamentals*, E91-A(12):3568–3575, Dec. 2008.

[30] William Kamp. *Alternate Number Systems for Optimizing DSP Performance in FPGA*. PhD thesis, Canterbury University, 2010.

[31] Leon Y. Li and Richard W. Eglese. An Interactive Algorithm for Vehicle Routeing for Winter-Gritting. *Journal of the Operational Research Society*, 47(2):217–228, Feb. 1996.

[32] Yong Ching Lim and Sydney R. Parker. Discrete Coefficient FIR Digital Filter Design Based Upon An LMS Criteria. *IEEE Transactions on Circuits and Systems*, CAS-30(10):723–739, Oct. 1983.

[33] R. Mahesh and A.P. Vinod. A New Common Subexpression Elimination Algorithm for Realizing Low-Complexity Higher Order Digital Filters. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(2):217–229, Feb. 2008.

[34] R. Montemann, L.M. Gambardella, A.E. Rizzoli, and A.V. Donati. Ant Colony System for a Dynamic Vehicle Routing Problem. *Journal of Combinatorial Optimization*, 10:327343, Aug. 2005.

[35] L.M. Moreira, J.F. Oliveira, A.M. Gomes, and J.S. Ferreira. Heuristics for a Dynamic Rural Postman Problem. *Computers and Operations Research*, 34(11):3281–3294, Nov. 2007.

[36] Martín Pedemonte, Sergio Nesmachnow, and Héctor Cancela. A Survey on Parallel Ant Colony Optimization. *Applied Soft Computing Journal*, 11(8):5181–5197, Dec. 2011.

[37] Miodrag Potkonjak, Mani B. Srivastava, and Anantha P. Chandrakasan. Multiple Constant Multiplications: Efficient and Versatile Framework and Algorithms for Exploring Common Subexpression Elimination. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 15(2):151–165, Feb. 1996.

[38] A.E. Rizzoli, F. Oliverio, R. Montemanni, and L.M. Gambardella. Ant Colony Optimization for Real-World Vehicle Routing Problems: From Theory to Applications. *Dalle Molle Institute for Artificial Intelligence Research, IDSIA*, pages 1–50, 2004.

[39] Henry Samueli. An Improved Search Algorithm for the Design of Multiplierless FIR Filters with Powers-of-Two Coefficients. *IEEE Transactions on Circuits and Systems*, 36(7):1044–1047, July 1989.

[40] Jeremy Siek, Lie-Quan Lee, and Andrew Lumsdaine. *The Boost Graph Library: User Guide and Reference Manual*. Addison-Wesley, 1 edition, 2002.

[41] Software and Hardware Generation for DSP algorithms. Multiplier block generator, 2009.

[42] T. Stüetzle and H. Hoos. The Max-Min Ant System and Local Search for Combinatorial Optimization Problems. *2nd International conference on metaheuritics - MIC 97*, July 1997.

[43] Jason Thong and Nicola Nicolici. Time-Efficient Single Constant Multiplication Based on Overlapping Digit Patterns . *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 17(9):1353–1357, Sep. 2009.

[44] Jason Thong and Nicola Nicolici. An Optimal and Practical Approach to Single Constant Multiplication. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 30(9):1373–1386, Sep. 2011.

[45] A.P. Vinod and Edmund M-K Lia. On the Implementation of Efficient Channel Filters for Wideband Receivers by Optimizing Common Subexpression Elimination Methods. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24(2):295–304, Feb. 2005.

[46] Yevgen Voronenko and Markus Püschel. Multiplierless Multiple Constant Multiplication. *ACM Transactions on Algorithms*, 3(2), May 2007.

[47] Chia-Yu Yao, Hsin-Horng Chen, Tsuan-Fan Lin, Chiang-Ju Chien, and Chun-Te Hsu. A Novel Common Subexpression Elimination Method for Synthesizing Fixed-Point FIR Filters. *IEEE Transactions on Circuits and Systems-I: Regular Papers*, 51(11):2215–2221, Nov. 2004.

[48] B. Yu, Z.-Z. Yang, and J.-X. Xie. A Parallel Improved Ant Colony Optimization for Multi-Depot Vehicle Routing Problem. *Journal of the Operational Research Society*, 62(1):183–188, Jan. 2011.

Listing 1: Parallel implementation of the MMAS algoritm.

```
main
input H, LD, α, β, τ_max, P                                   ▷ Input parameters
W ← Prepare(H)                                                ▷ Make the coefficients unique
G ← GenerateDemand(W, LD)                                     ▷ Generate the demand graph
G ← Initialize(G, τ_0)                                        ▷ Initialize pheromone
ant ← ant.resize(M)                                           ▷ Number of ants M ≤ P
for (k = 0; k ≤ 30; ++ k) do                                  ▷ Number of iterations
    W_2 ← RandomShuffle(W_2)                                  ▷ Random shuffle of δ = 1 subexpressions
    start ← rank                                              ▷ Distribute computation between processes
    stop ← start + 1
    for (m = start; m < stop; ++ m) do                        ▷ Ants departure
        w_2 ← W_2[m]                                          ▷ Get departure vertex value w_2
        ant[m] ← ant[m].clear()                               ▷ Clear ant m containers
        ant[m].w ← W                                          ▷ Initialize coefficients container
        ant[m].t ← {1}                                        ▷ Initialize synthesized coefficients container
        ant[m].r ← {}                                         ▷ Initialize visited arcs container
        ant[m] ← Synthesize(ant[m], w_2, 1, 1, LD)            ▷ Synthesis w_2
        ant[m].r ← ant[m].r + arc(1, w_2)                     ▷ Add visited arc to ant m path
        ant[m].stack ← SortArcs(ant[m].stack, Vertex(w_2, G)) ▷ Best next move at the top of stack container
        while (ant[m].stack ≠ φ ∧ ant[m].w ≠ φ) do
            arc(s_1, w) ← Pop(ant[m].stack)                   ▷ Extract s_1 and w from popped arc
            (s_2, τ) ← Weight(arc, G)                         ▷ Find arc attributes, s_2 and τ
            if w ∉ ant[m].t then                              ▷ Unsynthesized vertex
                if s_1 ∈ ant[m].t ∧ s_2 ∈ ant[m].t then        ▷ Optimum case
                    ant[m] ← Synthesize(ant[m], w, s_1, s_2, LD)   ▷ Synthesis vertex w
                    ant[m].r ← ant[m].r + arc(s_1, w)
                    ant[m].stack ← SortArcs(ant[m].stack, Vertex(w, G))
                else if s_1 ∈ ant[m].t ∧ s_2 ∉ ant[m].t then
                    arc(p_1, p_2) ← predecessors(s_2, G)      ▷ Find synthesized predecessors of s_2
                    ant[m].stack ← Push(ant[m].stack, arc(s_1, w))
                    ant[m].stack ← Push(ant[m].stack, arc(p_1, p_2))   ▷ Redirect the tour
                else if s_2 ∈ ant[m].t ∧ s_1 ∉ ant[m].t then
                    arc(p_1, p_2) ← predecessors(s_1, G)
                    ant[m].stack ← Push(ant[m].stack, arc(s_1, w))
                    ant[m].stack ← Push(ant[m].stack, arc(p_1, p_2))
                end if
            else if (ant[m].stack = φ ∧ ant[m].w ≠ φ) then ▷ Check if all the required arcs have been serviced
                for each u_2 ∈ W_2 do                         ▷ Paths for the non serviced arcs
                    if arc(1, u_2) ∉ ant[m].r then
                        ant[m] ← Synthesize(ant[m], u_2, 1, 1, LD)
                        ant[m].r ← ant[m].r + arc(1, u_2)
                        ant[m].stack ← SortArcs(ant[m].stack, Vertex(w_2, G))
                        break
                    end if
                end for
            end if
        end while
    end for
    bs ← Cost(G, ant)                                         ▷ Calculate the best-so-far tour cost
    G ← PheromoneEvaporate(G, ant)
    G ← PheromoneUpdate(G, ant, bs)
    for arc ∈ G do                                            ▷ Check for stagnation
        if τ_arc ≥ τ_max ∨ τ_arc ≤ τ_min then
            G ← Initialize(G, τ_0 + 1/bs)                     ▷ Re-initialize pheromone levels
            break
        end if
    end for
end for
end main
```