PROCEEDINGS OF SPIE

Independent Component Analyses, Wavelets, Unsupervised Nano-Biomimetic Sensors, and Neural Networks VI

Harold H. Szu
F. Jack Agee
Editors

17–19 March 2008
Orlando, Florida, USA

Volume 6979

SPIE

The Canonical Minimised Adder Graph Representation

Uwe Meyer-Bäse[a], Oscar Gustafsson[b], and Andrew Dempster[c]

[a]FAMU-FSU, ECE Dept., 2525 Pottsdamer Street, Tallahassee, FL 32310, USA;
[b]Department of Electrical Engineering, Linköping University;
[c]School of Surveying and Spatial Information Systems, University of New South Wales, Sydney 2052, Australia

ABSTRACT

In this paper we introduce a canonical minimised adder graph (CMAG) representation that can easily be generated with a computer. We show that this representation can be used to efficiently develop code generation for MAG graphs. Several code optimizations methods are developed in the computation of the non-output fundamental sum (NOFS) computation, which allows the computation of all graphs up to cost-5 be accomplished in a reasonable timeframe.

Keywords: minimised adder graph (MAG), n-dimensional Reduced Adder Graph (RAG-n), Multiple Constant Multiplier (MCM)

1. INTRODUCTION

There are only a few applications (e.g., adaptive filters) where a general programmable filter architecture is needed. In many applications, the filters are linear time-invariant (LTI) systems and the coefficients do not change over time. In this case, the hardware effort can essentially be reduced by exploiting the constant coefficient multiplier coding and adder (trees) used to implement the FIR filter multiplier block.[4]

2. GRAPH REPRESENTATION

Most single coefficient graphs can be described by three basic graph types: additive, multiplicative, and leapfrog graphs, as shown in Fig. 1. Data flow is from left to right. Edges have a power-of-two weight, while the vertices represent adders. The costs of the graphs are measured by the number of adders in use. For additive graphs, the cost is computed as the sum of the subgraph costs plus one; the multiplicative graphs have a total cost of the sum of the two subgraphs. The leapfrog graph requires at least one cost-1 graph (the first subgraph on the left), and for leapfrog-L we have at least a cost of $L-1$, i.e., there exists one leapfrog-2 graph of cost-3. All graphs up to cost-3 use only these three types.

In cost-4 we have only three graphs (i.e., 2, 6, 30) according to the numbering by Dempster and Macleod[1] (see the Appendix) that do not exactly match this pattern. Cost-4 graphs 2 and 6 may be described as "parallel" leapfrog graphs, i.e., graphs that have more than 2 branches starting at the same vertex, while graph 30 can be called a 9 o'clock graph. A 9 o'clock graph has at least 3 adders plus a cost-1 subgraph first, i.e., the cheapest 9 o'clock graph is a cost-4 graph. Table 1 shows a detailed classification of all graphs according to these classes.

The non-output fundamental sum (NOFS) now is the sum of all (except the output) odd fundamentals. Fundamentals are the vertex adder output values divided by a power-of-2 factor such that the odd part remains. Notice that additive graphs are commutative, i.e., $\text{NOFS}(C_k + C_m) = \text{NOFS}(C_m + C_k)$ while this is not true for multiplicative graphs, i.e., $\text{NOFS}(C_k \times C_m) \neq \text{NOFS}(C_m \times C_k)$. Note also that the subgraph classification is in general not unique. For instance, let us assume that a cost-4 graph is built as a product of four cost-1's. This graph can be considered as a $C_2 \times C_2, C_1 \times C_3$ or $C_3 \times C_1$ graph. In Table 1 we used the classification that starts with the largest known subgraph, in this example $C_3 \times C_1$.

Further author information: (Send correspondence to U.M.-B.)
U.M.-B.: E-mail: umb@eng.fsu.edu, Telephone: (850) 410-6220
O.G.: E-mail: oscarg@isy.liu.se
A.D.: E-mail: a.dempster@unsw.edu.au

Independent Component Analyses, Wavelets, Unsupervised Nano-Biomimetic Sensors, and Neural Networks VI, edited by Harold H. Szu, F. Jack Agee, Proc. of SPIE Vol. 6979, 69790P, (2008) · 0277-786X/08/$18 · doi: 10.1117/12.776928

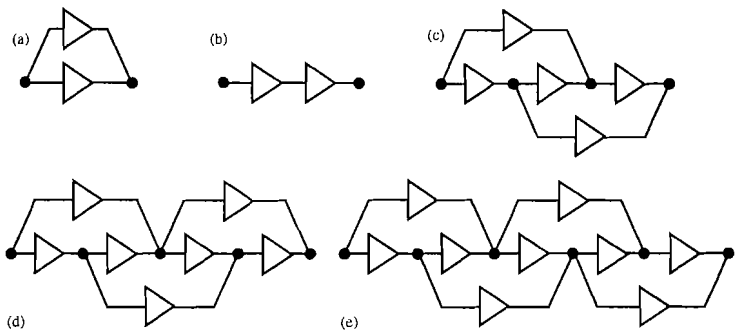


Figure 1. Most frequent used graph classes. **(a)** Additive. **(b)** Multiplicative. **(c)** leapfrog-2, **(d)** leapfrog-3, and **(e)** leapfrog-4.

Table 1. Type of graphs found in cost-1 to 5 MAG. (Add = additive graph; Mul = multiplicative graph; lf = leapfrog graph).

Cost-1	$C_0 + C_0$					
1	1					
Cost-2	$C_1 + C_0$	$C_2 \times C_1$				
2	1	1				
Cost-3	$C_2 + C_0$	$C_1 + C_1$	$C_2 \times C_1$	$C_1 \times C_2$	lf−2	
7	2	1	2	1	1	
Cost-4	Mul	Add	lf	Others		
32	13	9	7	3		
	$C_3 \times C_1$	$C_2 \times C_2$	$C_1 \times C_3$			
	7	2	4			
	$C_3 + C_0$	$C_2 + C_1$	lf−2	lf−3	9 o'clock	Parallel lf
	7	2	6	1	1	2
Cost-5	Mul	Add	lf	Others		
193	66	42	38	47		
	$C_4 \times C_1$	$C_3 \times C_2$	$C_2 \times C_3$	$C_1 \times C_4$		
	32	7	8	19		
	$C_4 + C_0$	$C_3 + C_1$	$C_2 + C_2$			
	32	7	3			
	lf−2	lf−3	lf−4	9 o'clock	TBA	
	29	8	1	7	40	

3. CANONICAL GRAPH REPRESENTATION

As we have seen in the previous section, the number of graphs grows more fast than factorial with the cost of the graphs. In the past, for each subgraph a different subroutine needed to be used to compute, for instance, the NOFS. A canonical representation is needed to allow a computer program to use a "worklist" of all graphs to be considered. Such a canonical representation can be built using the following rules:

1) Enumerate and sort in linear order all $C + 1$ vertices $V_k, k \in [0, C]$ of a cost C graph.

2) Allow for all vertices $V_k; k \in [1, C]$ two input edges (e_{k1}, e_{k2}) and without loss of generality set $(e_{k1} \leq e_{k2})$, e.g., $(2, 1) \rightarrow (1, 2)$. V_0 has fan-in zero.

3) All input edges start from the left, i.e., $e_{k1}, e_{k2} \leq k$ and each vertex has at least a fan-out of **1**.

4) Remove all graphs from the list that are simple permutations of the vertices V_k, $k \in [1, C-1]$; use the graph that starts with the largest known subgraph.

The first three rules allow us to build all graphs up to cost-3 in a unique way. Consider, however, a cost-4 graph with a $C_2 + C_1 = C_1 + C_2$ graph

```
(0,0)(1,1)(0,0)(2,3) = (0,0)(0,0)(2,2)(1,3)
```

See cost-4 graph no. 29 in Appendix A. These two graphs are congruent and will not generate any new NOFS results. Switching only vertex $2 \leftrightarrow 3$ results in another congruent graph.

```
(0,0)(1,1)(0,0)(2,3) = (0,0)(0,0)(1,1)(2,3).
```

Therefore, starting with cost-4 and higher, we also need to eliminate congruent graphs, using Step 4. Since the incoming and outgoing vertex cannot be switched, we need to check $(C-1)!$ permutations for each cost-C graph.

Initially, after performing the first 3 steps for cost-4 (cost-5), 38 (295) graphs were generated, respectively. After Step 4 the list is reduced to the correct value 32 (193). Although in previous publications[2,3] the number of graphs were computed in different ways (e.g., using `nauty` by McKay), the total number of graphs are the same.

Appendix A shows the figures of all graphs up to cost-5. The numbering of cost-1 through cost-4 graphs is in sync with the coding by Dempster and Macleod.[1] Cost-5 graphs are sorted in a weighted fashion, i.e., the graph with the most leading zeros is listed first. The "worklist" of the CMAG coding online[5] is sorted by the computation time.

4. APPLICATION OF NOFS COMPUTATION

One important part of the original RAG-n algorithm is the selection of the NOFs when no more cost-1 or cost-2 coefficients can be added to synthesize the target coefficients. In this case, the algorithm with the smallest coefficient with the minimum NOFS must be added. This is motivated by the fact that additional small NOFs will generate more additional coefficients than a larger NOF at no cost. For example, assume that the coefficient 45 needs to be added and that an NOF value has to be decided upon. The NOF LUT lists all possible NOF values as 3, 5, 9, or 15. It can now be argued that if 3 is selected, more coefficients are generated than if any other NOF is used, since 3, 6, 12, 24, 48, ... can be generated without additional effort from NOF 3. Other NOF values, such as 15, can generate 15, 30, 45,... etc., so the choices towards producing other coefficients at no cost are significantly restricted.

4.1. NOFS Computation Optimizations

The computation time for all cost-4 graphs is reasonable even for 19-bit data as shown in Fig. 2(a). However, the computation time for the cost-5 graphs may be a concern. For 19-bit (plus a guard bit) signed data each edge has about 40 different values $\pm 2^k, k \in [0, 20]$ and for a cost-5 graph, then, $40^{10} = 10485760 \times 10^9$ different graphs with different edge weights are possible. Assuming that a fast computer/hardware can do 1G=10^9 checks per second, 2912 hours would still be required for each graph. Since we are only interested in the graphs with the minimum NOFS we can therefore implement (in the CMAG code generation) the following simplifications.

1) The odd fundamental of the graph does not change if we move a power-of-2 factor across a vertex. In case the fan-out is one, we select the outer edge as factor 1 since this saves the most run time, see Fig. 3(a). CSD graphs are optimal in this regard, and we save a factor 40 of graphs for each edge to which we can apply this simplification.

2) If we have a cost-1 subgraph that is followed by a vertex with two or more outputs, the optimization from 1 does not work. However, we can choose to have one of our input edges be a factor ± 1, see Fig. 3(b) For a 19-bit table, the savings is, instead of all $2 \times 361 = 722$ cost-1 subgraphs, only 74 iterations needing to be computed, resulting in a factor of 9.7 run time improvement.

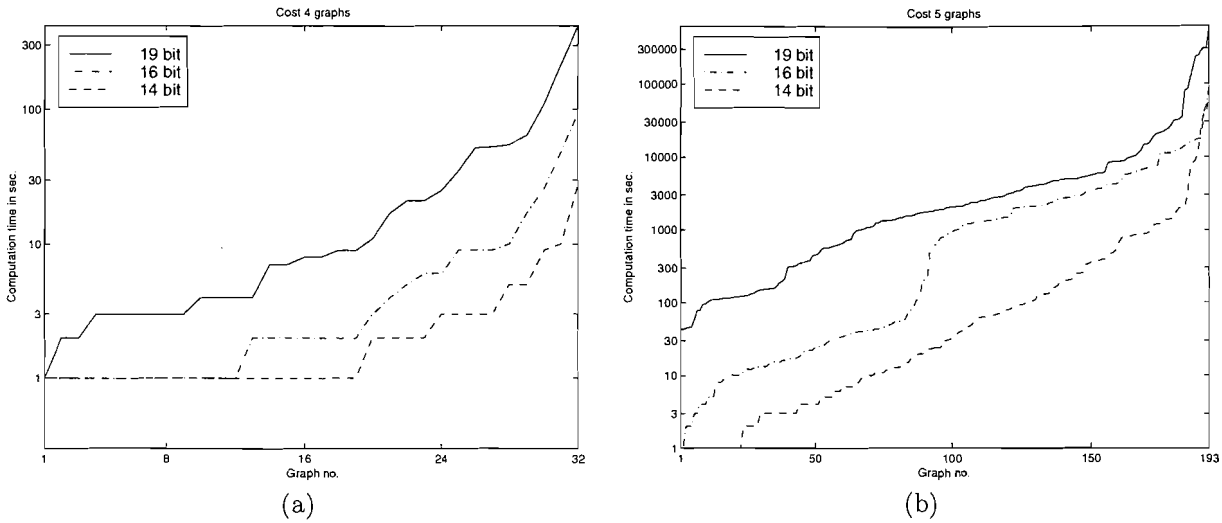


Figure 2. Computation time for all **(a)** 32 cost-4 **(b)** 193 cost-5 graphs.

3) When evaluating subsequent graphs we can stop the computation if an odd fundamental occurs that is larger than the maximum NOFS over all coefficients. While a direct factor for improvement cannot be computed, it was observed that this is especially helpful for the graphs that could not be sped up with method 1+2 described above, e.g., in the leapfrog graphs.

Using these three improvements makes it possible to compute all 193 cost-5 graphs in a reasonable time. We used a $120K Sun-Fire-880 server with 8 processors and 32 GB main memory in the first author's ASIC design lab to run the programs. Compared with a typical PC, a single processor program runs about twice as fast on the Sun compared with a 2 GHz PC. The code (20103 lines of C-code) for all cost-1 to cost-5 graphs was produced in less than a second. The compilation without optimization took 13 seconds, while with maximum optimization (`-x 04` for Sun Studio 11 C++ Compiler Sun 5.8 `cc`) the 20K lines of C-code required 679 seconds to compile. The `04` code runs about a factor of 10 times faster with maximum optimization than without. Figure 2(b) shows the computation time for the cost-5 graphs only. Total CPU time was 885 hours for all processors together, or 110 hours (4.5 days) for our 8 processor machine. The most demanding graph was the leapfrog-4 graph, see Fig. 1(e),

```
(0,0) (0,1) (1,2) (2,3) (3,4)
```

which took up 19% of the computation time alone! The last 8 graphs took up to 71% of the run time; in other words, 95% of the graphs from the total of 193 graphs required only 28% of the total CPU time. However, removing the expensive graphs from the "work list" was not a good idea, since the leapfrog-4 could improve the NOFS of 49106 graphs for 19-bit coefficients alone.

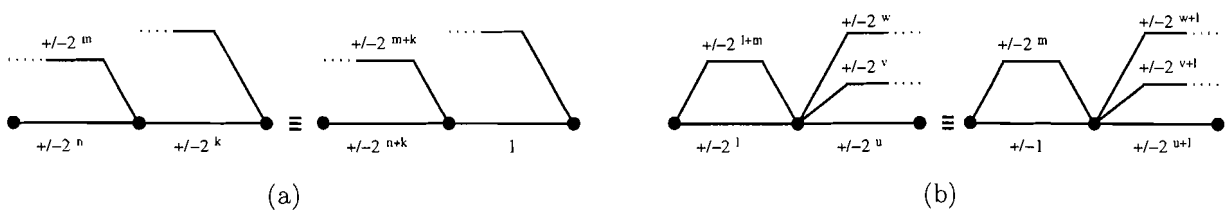


Figure 3. NOFS simplifications (a) for fan-out one. (b) Cost-1 subgraph followed by a fan-out larger than one.

After our speed-up trick 3 from above, it is recommended that we try to lower the NOFS before we run the expensive graphs; we have used the run-time for lower bitwidth, i.e., 14- and 16-bits to sort the graphs in such a way that the more expensive graphs are computed last. We could take advantage of the lower NOFS computed in the previous graphs. However, when we wished to split up the jobs among the 8 processors, we computed the NOFS parallel and had to merge the results for the processors.

5. SUMMARY

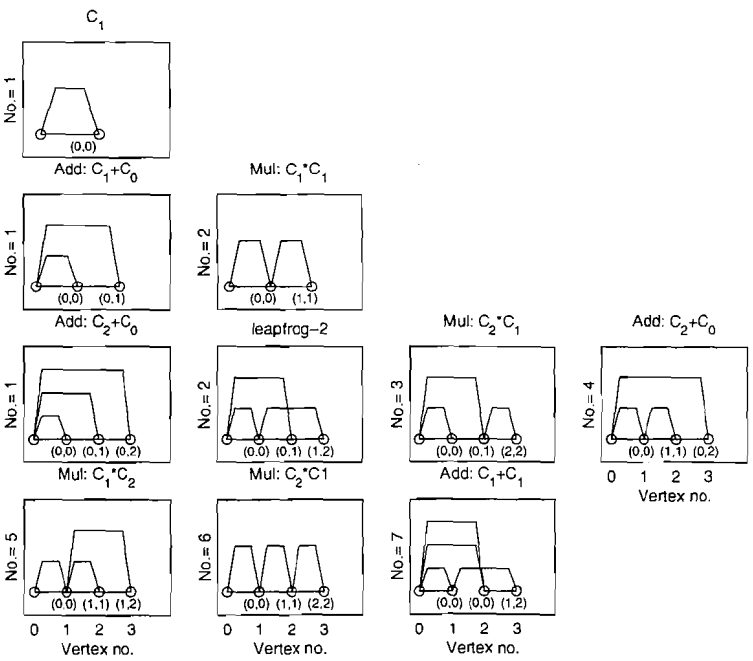
We have introduced the canonical minimised adder graph (CMAG) representation that allows an automatic computation of MAGs with a computer. C-code description (about 20K lines of C-code) of all 235 cost-1 to cost-5 graphs are generated within a second. The CMAG was then applied to NOFS computations, and within a reasonable time (4.5 days) all graphs in 19-bit precision were computed. We have described several optimization methods that enable a faster computation of the NOFS tables.

REFERENCES

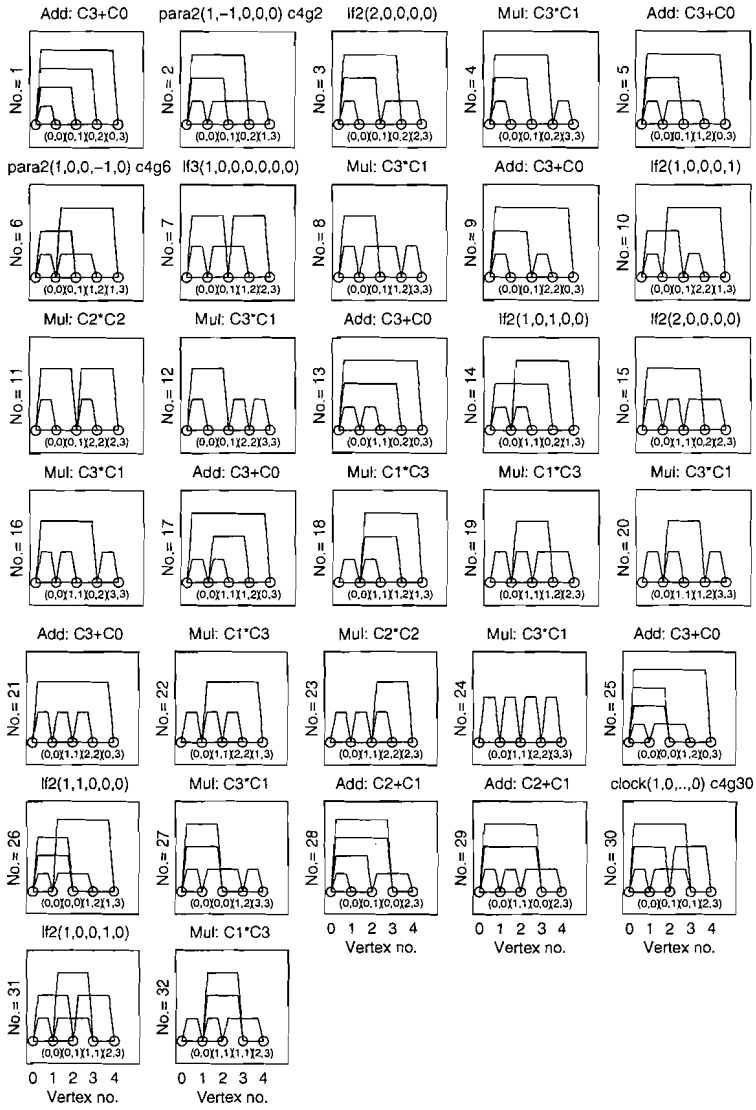
1. A. Dempster and M. Macleod, "Constant integer multiplication using minimum adders," *IEE Proceedings - Circuits, Devices & Systems* , vol. 141, pp. 407–413, Oct. 1994.
2. O. Gustafsson, A. Dempster, and L. Wanhammar, "Extended Results for Minimum-Adder Constant Integer Multipliers," in *Proceedings ISCAS*, May 2002, pp. I-73–76.
3. O. Gustafsson, A. G. Dempster, K. Johansson, M. D. Macleod, and L. Wanhammar, "Simplified Design of Constant Coefficient Multipliers," *Circuits, Systems and Signal Processing*, vol. 25, no. 2, pp. 225–251, Apr. 2006.
4. U. Meyer-Baese: *Digital Signal Processing with Field Programmable Gate Arrays* , 3nd edn. (Springer-Verlag, Berlin, 2007), 774 pages.
5. U. Meyer-Baese, "Online resource," at *http://www.eng.fsu.edu/~umb/cmag.htm.*

6. APPENDIX A: COST 1-5 MAG FIGURES

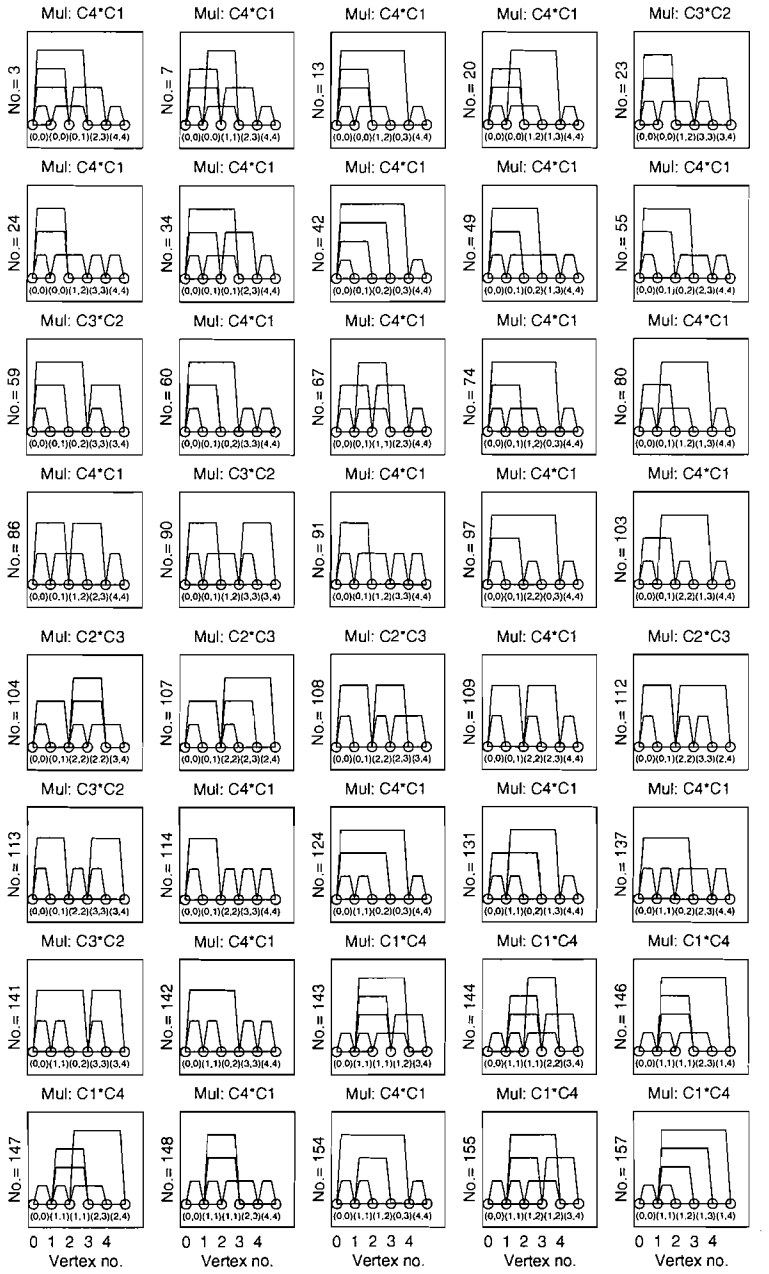
All cost-1 to 3 MAG graphs:



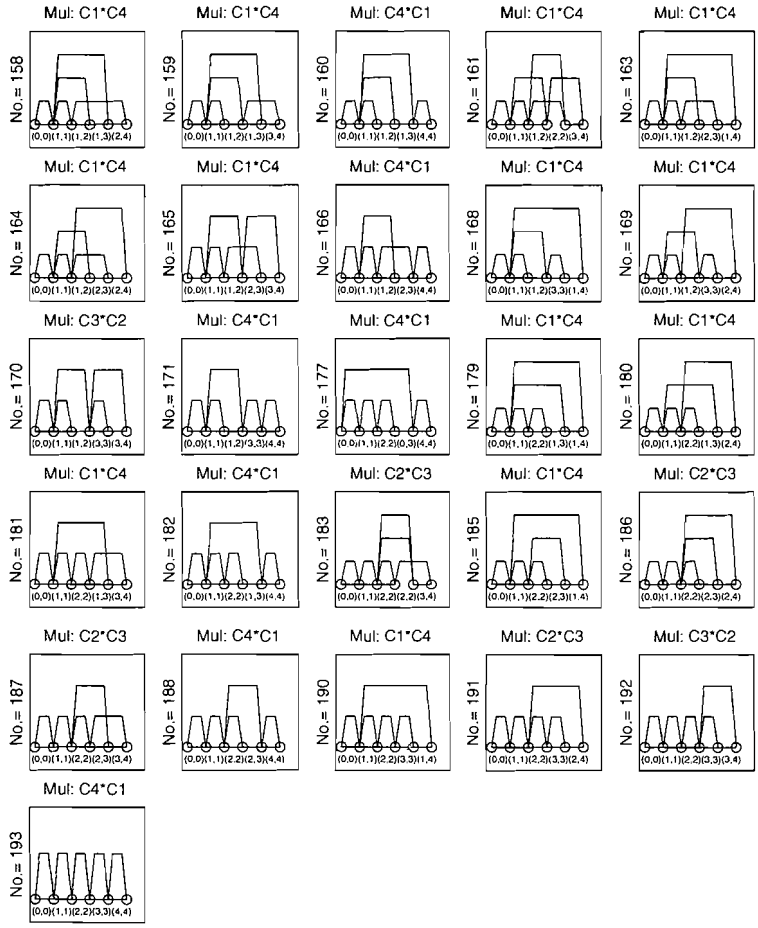
All 32 Cost-4 MAG Graphs:



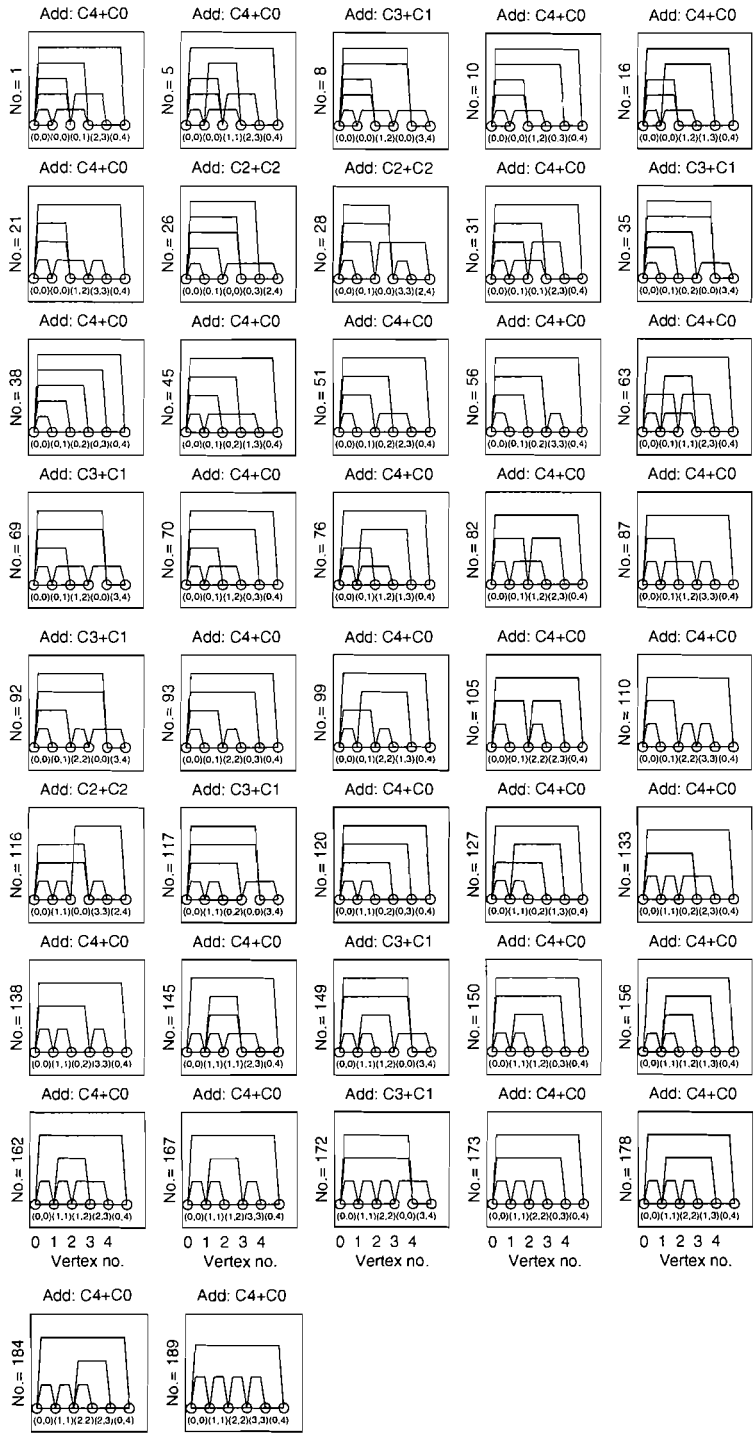
First Come the Cost-5 Multiplier Graphs



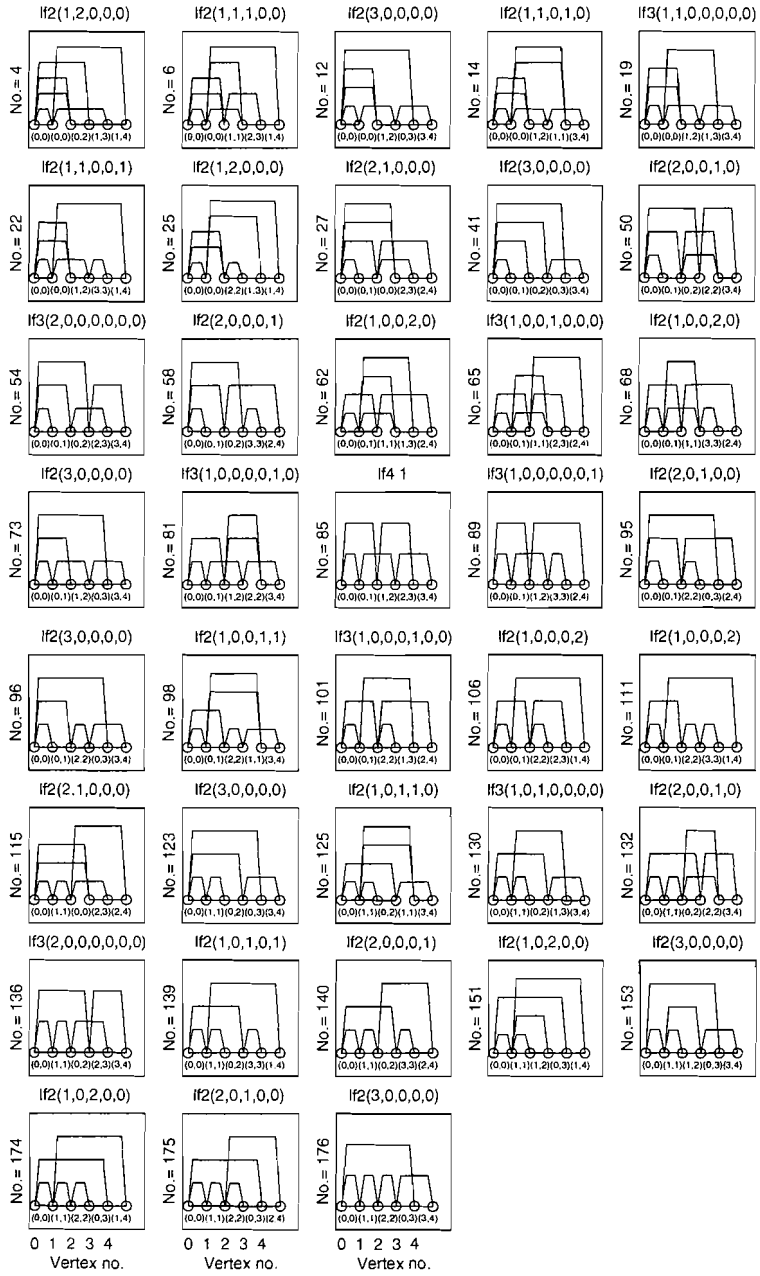
Cost-5 Multiplier Graphs (cont.)



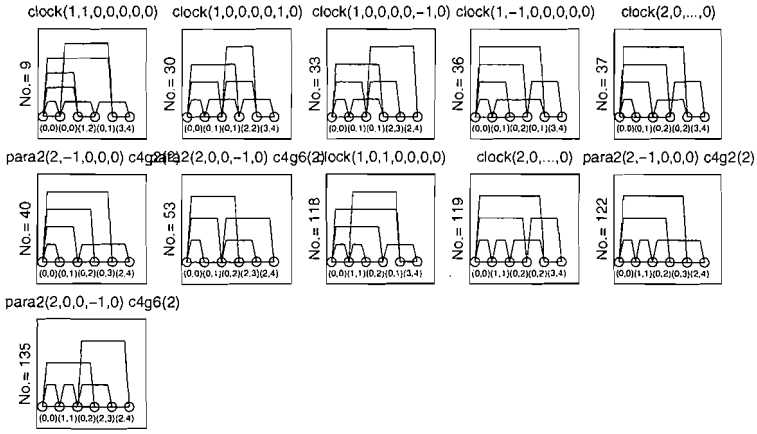
Second-to-Come ADDER Graphs



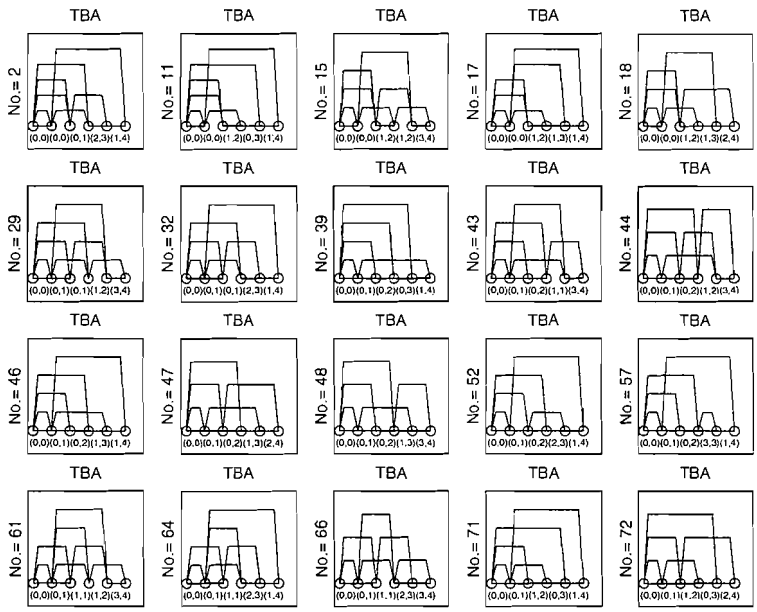
Next-to-Come Cost-5 leapfrog Graphs



Next Come the OTHER Known Graphs



Last-to-Come Are the Cost-5 NAMELESS Graphs



Cost-5 NAMELESS Graphs (cont.)