Parametric Analog Behavioral Modeling Based on Cancellation-Free DDDs

X.-D. Sheldon Tan[†] and C.-J. Richard Shi[‡]

[†] Department of Electrical Engineering
University of California, Riverside, CA 92521, USA
stan@ee.ucr.edu
[‡] Department of Electrical Engineering
University of Washington, Seattle, WA 98195, USA
shi@ee.washington.edu

Abstract

This paper presents an efficient approach to generating simplified symbolic expressions for behavioral modeling of large linear analog circuits. The approach is based on a compact determinant decision diagram representation of exact transfer functions and characteristics of analog circuits. We show how cancellation-free, s-expanded DDDs (sDDDs) can be constructed from DDDs. With several DDD properties characterized in this paper, we compare two efficient algorithms for finding dominant terms based respectively on shortest paths and dynamic programming. We show that all these algorithms take time linear in terms of the number of DDD vertices. Experimental results show that our shortest path based algorithm is more memory efficient than the dynamic programming based algorithm.

Key words: Behavioral Modeling, Symbolic Analysis, Determinant Decision Diagrams.

I Introduction

Behavioral modeling aims at generating compact and simulation ready models for analog circuit blocks that capture the circuit characteristics of interests. Behavioral models are essential to full system design verification. Parametric, in contrast to purely numerical, behavioral modeling is to generate behavioral models that contain circuit and layout design parameters. Parametric behavioral models would allow rapid architecture exploration, parasitic-aware system-level synthesis and optimization, and what-if analysis of mixed-signal systems-on-chips [6]. It is encouraging to note that several very recent attempts to numerical behavioral model generation of nonlinear analog circuits all rely on efficient linear circuit analysis, such as piece-wise linear modeling [8], quadratic modeling [1], and linear time-varying modeling [7].

One way to derive parametric behavioral models of analog modules is by means of symbolic analysis. As illustrated in [3], simple yet accurate symbolic expressions can also be interpretable by analog designers to gain the insight into circuit behavior, performance and stability, and are important for many applications in circuit design such as transistor sizing and optimization, topology selection, sensitivity analysis, behavioral modeling, fault simulation, testability analysis and yield enhancement [4].

Recently we proposed an efficient and systematic method of deriving simple yet accurate symbolic expressions that represent circuit characteristics in terms of circuit parameters for linear(ized) analog circuits [13]. The DDD-based approximation method compares very favorably with existing symbolic analysis tools.

In this paper, we present a more efficient way to obtain cancellation-free s-expanded DDDs in our symbolic approximation framework. Inspired by the work [14], we also implemented a dynamic programming (DP) based dominant term generation algorithm on the basis of DDD graphs. We will show that the algorithm works only when certain properties are held in the DDD graphs and it is not applicable to cancellation-free s-expanded DDDs in general. The detailed comparison of the dynamic algorithm based algorithm and the shortest path (SP) based algorithm will presented in the experimental result section.

This paper is organized as follows. Section II reviews the concepts of DDDs and s-expanded DDDs. Section III presents the new method for constructing cancellation-free s-expanded DDDs and Section IV explains implementation of DP based term generation algorithm based on DDD graphs. Experimental results are described in Section V. Section VI concludes the paper.

II DDDs and s-Expanded Coefficient DDDs

In this section, we provide a brief overview of the notion of determinant decision diagrams [10]. We review how a s-expanded DDD can be used to represent the symbolic coefficients of a s polynomial.

Determinant Decision Diagrams [10] are compact and canonical graph-based representation of determinants. The concept is best illustrated using a simple RC filter circuit shown in Fig. 1. Its system equations can be written as

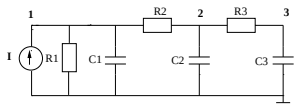

Fig. 1. A simple RC circuit.

$$\begin{bmatrix} \frac{1}{R_1}+sC_1+\frac{1}{R_2} & -\frac{1}{R_2} & 0 \\ -\frac{1}{R_2} & \frac{1}{R_2}+sC_2+\frac{1}{R_3} & -\frac{1}{R_3} \\ 0 & -\frac{1}{R_3} & \frac{1}{R_3}+sC_3 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = \begin{bmatrix} I \\ 0 \\ 0 \end{bmatrix}$$

We view each entry in the circuit matrix as one distinct symbol, and rewrite its system determinant in the left-hand side of Fig. 2. Then its DDD representation is shown in the right-hand side. Please refer to [10] for the formal definition of a DDD graph.

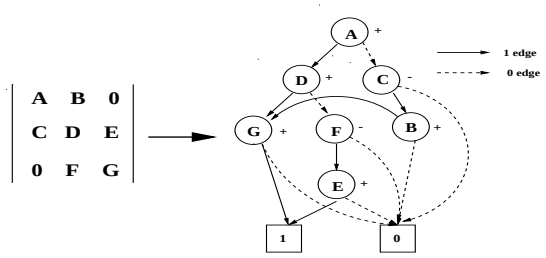

Fig. 2. A matrix determinant and its DDD.

Given a symbolic matrix where each entry is a distinct symbol, and a fixed order of each label appears in the DDD starting from its root. Then the expansion of matrix can be carried out with respect to a non-zero element:

$$det(\mathbf{A}) = a_{r,c}(-1)^{r+c} det(\mathbf{A}_{a_{r,c}}) + det(\mathbf{A}_{\overline{a}_{r,c}}). \qquad (1)$$

A DDD vertex pointed by 0-edge, which comes from vertex $a_{r,c}$, represents a determinant obtained by setting $a_{r,c} = 0$ and is denoted as $det(\mathbf{A}_{\overline{a}_{r,c}})$. If a DDD is constructed by using Laplace development, all the 0-edge linked DDD vertices will have the same row index or column index. Then there exists a *unique DDD representation* of the determinant. This is the DDD canonicity.

To exploit the DDD to derive circuit characteristics, we need to directly represent circuit parameters not matrix entries. To this end, s-expanded DDDs are introduced [12]. Consider again the circuit in Fig. 1 and its system determinant. Let us introduce a unique symbol for each circuit parameter in its admittance form. Specifically, we introduce $a = \frac{1}{R_1}$, $b = f = \frac{1}{R_2}$, $d = e = -\frac{1}{R_2}$ $g = k = \frac{1}{R_3}$, $i = j = -\frac{1}{R_3}$, $C_1 = c, h = C_2, l = C_3$. Then the circuit matrix can be rewritten as

$$\begin{bmatrix} a+b+cs & d & 0 \\ e & f+g+hs & i \\ 0 & j & k+ls \end{bmatrix}$$

The original 3 product terms will be expanded to 23 product terms in different powers of s. We can represent these product terms nicely using a slight extension of the original DDD, as shown in Fig. 3. This DDD has exactly the same properties as the original DDD except that there are four roots representing coefficients of s^0, s^1, s^2, s^3. Each DDD root represents a symbolic expression of a coefficient in the corresponding s polynomial. Each such DDD is called a *coefficient* DDD, and the resulting DDD is a *multiple-root* DDD. The original DDD in which s is contained in some vertices is called *complex* DDD.

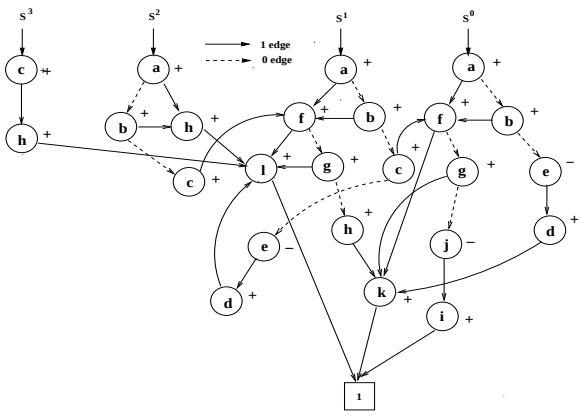

Fig. 3. An s-expanded DDD.

The s-expanded DDD can be constructed from the complex DDD in linear time in the size of the original complex DDD [11, 12]. In Fig. 4, we present a version of the algorithm called COEFFCONST. COEFFCONST takes a complex DDD rooted at D and returns its corresponding coefficient DDD, where,

- *D.child1* and *D.child0* denote, respectively the vertices pointed to by the 1-edge and 0-edge of vertex D.
- Let each circuit node be connected by at most m devices. Then each matrix entry can be a sum of at most m individual elements, each denoted by $D.x_i$ can be of type s^0 or s^1 based on the MNA formulation. For example, for a matrix entry $a + b + cs$, we have $m = 3$, $D.x_1 = a, D.x_2 = b$, and $D.x_3 = c$, and their types are s^0, s^0, and s^1.
- COEFFUNION(P_1, P_2) computes the union of two coefficient DDDs, P_1 and P_2.
- COEFFMULTIPLY(P, v) computes the product of coefficient DDD P and coefficient DDD vertex v.
- $P * s$ increments the power of s in coefficient DDD P.

```
COEFFCONST(D)
1   if ( D = 0 or D = 1)
2       return NULL
3   L0 = COEFFCONST(D.child0)
4   L1 = COEFFCONST(D.child1)
5   Presult = NULL
6   for i = 1 to m do
7       if (type(D.xi) = s0)
8           Pg = COEFFMULTIPLY(L1,D.xi)
9           Presult = COEFFUNION(Pg, Presult)
10      if (type(D.xi) = s1)
11          Pc = COEFFMULTIPLY(L1 * s,D.xi)
12          Presult = COEFFUNION(Pc, Presult)
13  return COEFFUNION(Presult, L0)
```

Fig. 4. Coeffi cient DDD construction.

III Construction of Cancellation-Free s-Expanded DDDs

In our method, modified nodal analysis (MNA) is employed to describe the linear(ized) analog circuit. Our approximation starts with exact DDD representations of network transfer functions. Before we construct s-expanded DDDs, we first simplify DDDs by discarding insignificant terms [13]. This step will significantly reduce the number of terms in the DDDs as many bias circuitry has very marginal impacts on the signal data paths in typical analog modules. After this, the cancellation-free s-expanded DDDs are constructed efficiently from DDDs as detailed in the following.

Canceling terms arise from the use of the MNA formulation and device matching in analog circuits. For instance, consider the s-expanded DDD in Fig. 3. Since $g = k = \frac{1}{R_3}$ and $i = j = -\frac{1}{R_3}$, term $agks^0$ cancels term $-ajis^0$ in the coefficient DDD of s^0. To ease our discussion, we refer it to as *symbolic cancellation* if terms are canceling each other symbolically or *numerical cancellation* if terms are canceling each other numerically, i.e., only if their normal numeric values are used.

Removing canceling terms is useful for enhancing expression interpretability, the efficiency and accuracy of numerical evaluation. It also facilitates efficient dominant term generation, since most of generated terms can be canceling terms. Our experimental results show that up to 70-90 percent product terms can be canceling terms for a typical analog circuit.

In this subsection, we focus on building symbolic cancellation free s-expanded DDDs. This is important since the MNA formulation leads naturally to a lot of canceling terms. In practice, removing canceling terms has been known to be a difficult problem in determinant-based symbolic analysis methods [3, 9]. Numerical cancellation can be taken care of during the dominant term generation step.

Symbolic canceling terms can be easily detected by considering several MNA matrix patterns shown in Fig. 5. For

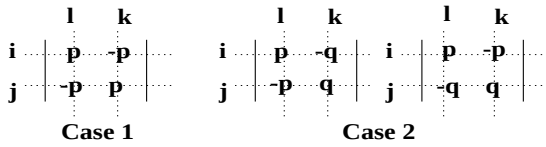

Fig. 5. Matrix patterns causing term cancellation.

example, case 1 may come from the rectangular appearance of a floating resistor in the nodal admittance formulation.

Canceling terms caused by matrix patterns shown in Fig. 5 can be removed from a DDD by using basic DDD operations [13] after s-expanded DDDs are constructed. But as a number of DDD operations are performed on the whole s-expanded DDDs, the de-cancellation process may take a huge amount of temporary memory specially for large s-expanded DDDs and is typically the most time consuming step in the whole approximation operation.

It turns out that canceling terms can be removed more efficiently during the construction of s-expanded DDDs from original complex DDDs. To this end, we need to build a canceling label list $CL(L_x)$ for each label L_x such that $L_x > L_y, \forall L_y, L_y \in CL(L_x)$. With this, we first loop through all

COEFFMULPLTY($P, D.x$)
1 **for** $i = 0$ to $p - 1$ do
2 for each $L_y \in CL(D.x)$
3 $P[i]$ = REMAINDER($P[i], L_y$)
4 $P[i]$ = MULTIPLY($P[i], D.x$)
5 return P

Fig. 6. Cancellation-free COEFFMULTIPLY.

the devices and build the CL for each unique symbol. Then we call function COEFFCONST to construct the s-expanded DDDs. The pseudo code for the cancellation-free COEFFMULTIPLY in COEFFCONST is shown in Fig. 6, where lines 2 to 3 are used not to generate canceling terms.

For this example the resulting cancellation-free DDD shown in Fig. 7 is smaller than the original s-expanded DDD in Fig. 3. Our experimental results, however, show that the cancellation-free s-expanded DDDs can be larger than the normal s-expanded DDDs [13].

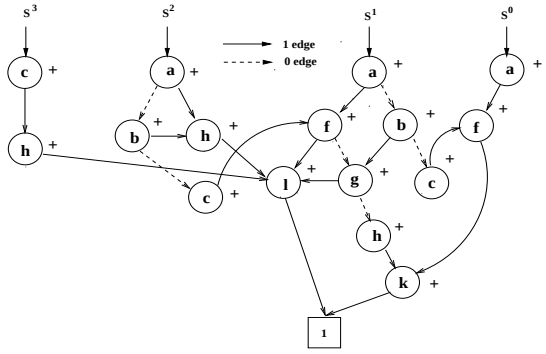

Fig. 7. Cancellation-free multi-root DDD.

IV Dominant Term Generation Methods

Many circuit characteristics are dominated by a small number of product terms called *significant* or *dominant* terms. In the follow section we present a DP based algorithm for generating dominant terms using DDD graphs.

A. Dynamic Programming (DP) Based Generation of Dominant Terms

Our algorithm is inspired by the recent work of Verhaegen and Gielen [14]. The proposed DP algorithm replies on some theoretical properties of DDDs, as characterized below. First, we note that the vertex ordering heuristic used to construct DDDs is based on Laplace expansion of a circuit matrix along the row or column [10]. We also know from the canonicity of DDD that the structure of DDD is unique under a fixed vertex order, i.e., independent of how it is constructed. We thus have the following lemma.

Lemma 1 *All 0-edge linked vertices comes from either from the same row or the same column of the original circuit matrix.*

Proof: As we know that a complex DDD is constructed by using Laplace development. To ease our proof, we repeat the the determinant expansion rule (1) in the following:

$$det(\mathbf{A}) = a_{r,c}(-1)^{r+c} det(\mathbf{A}_{a_{r,c}}) + det(\mathbf{A}_{\overline{a}_{r,c}}).$$

In DDD graph, the addition relationship between cofactor $a_{r,c}(-1)^{r+c}det(\mathbf{A}_{a_{r,c}})$ and remainder $det(\mathbf{A}_{\overline{a}_{r,c}})$ is represented by a 0-edge between two DDD vertices representing the two expressions. Since all the $a_{r,c}$ are selected from a row or a column by Laplace development rule, so all the 0-edge linked DDD vertices will have the same row index or column index. Further the row or column must exist in the remainder $det(\mathbf{A}_{\overline{a}_{r,c}})$ □.

With Lemma 1 as the basis, we can show the following main result.

Lemma 2 *The incoming edges of a non-terminal vertex in a complex DDD or s-expanded DDD are either all 0-edges or all 1-edges.*

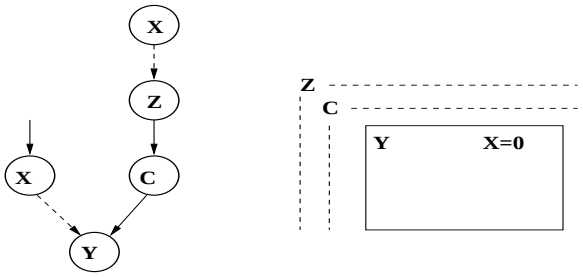

Fig. 8. A DDD vertex with both incoming 1-edge and 0-edge

Proof: Suppose there exists a DDD vertex Y that has an incoming 0-edge from DDD vertex X, and an incoming 1-edge from vertex C as shown in Fig. 8. The DDD subgraph rooted at Y therefore represents a remainder by the development of X. According to the determinant development rule (1), the 0-edge between X and Y means that both the row and column of X exist in the remainder represented by Y (actually, X and Y are in the same row or column according to Lemma 1). Also the element of X in the remainder Y will become zero.

On the other hand, Y and C are not in the same row and column as Y is obtained by Laplace development with respect to C. Since element X is zero, X will appear before C with a 0-edge in the path (or there could be a number of elements in between linked with 0-edges; as they must come from the same row or column). Let the first non 0-edge linked vertex to be Z (Z can be C). The fact that Z is 1-edge linked implies the minor after Z is developed will not have the row and column of Z. As a result, X would have been crossed out since it is in the same row or column as Z. This contrasts with the fact that X appears in the remainder Y.

This lemma also holds for s-expanded DDDs by noticing that 0-edges between DDD vertices may also represent the relationship among the different symbolic circuit parameters in a non-zero element in a determinant. □

Now we are ready to describe a DDD-based DP algorithm for generating k dominant terms. From Lemma 1 and Lemma 2, we only need to calculate dominant terms for 1-edge pointed DDD vertices. Let D be a 1-edge pointed vertex. We use *D.counter* to keep track of the number of dominant terms generated for the vertex D (D is included in the terms). We use an ordered array, denoted as *D.term-list*, to keep track of those generated dominant terms in the minor represented by D, where *D.term-list[1]*, *D.term-list[2]*,... represents the largest term (first dominant term), the second largest term (second dominant term). *D.counter* is initially set to 1 for all 1-edge pointed vertices, and can be increased up to k.

```
GETKDOMINANTTERMS(D, k)
1  if ( D = 1)
2      return 1
3  if ( D = 0 )
4      return NULL
5  else if ( D.term-list[k] exists )
6      return D.term-list[k]
7  else
8      COMPUTEKDOMINANTTERMS(D, k)
9  return D.term-list[k]

COMPUTEKDOMIANTTERMS(D, k)
1  if ( D = 1)
2      return 1
3  while (D.term-list[k] not exists) do
4      for each 0-edge linked vertex V starting with D do
5          if ( V = 1 and V.counter > 1)
6              continue
7          term = GETKDOMINANTTERMS(V.child1, V.counter)
8          if ( term exists)
9              new_term = UPDATETERM(term, V)
10             term_value = COMPUTETERMVALUE(new_term)
11             if (term_value is the largest)
12                 new_largest_term = new_term
13                 vertex_need_update = V
14     if (new_largest_term exists)
15         D.term-list.push(new_largest_term)
16         vertex_need_update.counter++
17     else
18         break
19 return
```

Fig. 9. Dynamic programming based dominant term generation algorithm.

As shown in pseudo code in Fig. 9, to find the k dominant terms at a 1-edge pointed vertex D, we first check if such k terms already exist in the *term-list* in GETKDOMINANTTERMS(D, k). If they do not exist, they will be generated by invoking COMPUTEKDOMINANTTERM(D, k). In COMPUTEKDOMINANTTERM(D, k), the largest term is computed and stored in the *term-list* of D by visiting all the 0-edge linked DDD vertices. Each time a largest term is computed,

the corresponding vertex $V.counter$ will be increased by 1. UPDATETERM() adds a vertex (its symbol) into a term represented by a DDD tree. COMPUTETERMVALUE() computes the numerical value of a given term. We use $V.child1$ to represent the vertex pointed to by the 1-edge originating from vertex V.

We can see that this algorithm takes linear time in terms of the size of a DDD, if UPDATETERM() and COMPUTETERMVALUE() are implemented to use constant time each time when a vertex is added to a term. This can be accomplished by using memory caching.

We note that cancellation-free s-expanded DDDs do not satisfy Lemma 2. For example, in Fig. 7, vertex g in the coefficient of s^1 has both incoming 1-edge and incoming 0-edge. Verhaegen and Gielen [14] resolved this problem by duplicating vertex g. Their approach, however, will destroy the DDD canonicity, a property crucial to the efficiency of many DDD-based graph manipulations. In this paper, we apply the proposed DP based term generation approach on the s-expanded DDDs before de-cancellation.

B. Shortest Path (SP) based Generation of Dominant Terms

We proposed a very elegant algorithm for finding k dominant terms that runs most time as efficiently as DP but always uses much less memory in [13]. Further it does not require DDDs to satisfy Lemma 1 and 2, and thus can be applicable to DDDs after de-cancellation. The algorithm is based on the observation that the k dominant product terms can be transformed to the k shortest paths in a DDD.

The shortest path in a coefficient DDD, which is a DAG, can be found by depth-first search in time $O(V + E)$, where V is the number of DDD vertices $|DDD|$ and E is number of edges [2]. For DDDs, $E = 2|DDD|$, thus the SP based algorithm takes time $O(|DDD|)$.

A nice property of DDD is that after we find the shortest path from a DDD, we can subtract it from the DDD using a basic DDD operation [10], and then we can find the next shortest path in the resulting DDD. In this manner, we can find the k shortest paths in time $O(k \cdot |DDD|)$.

This procedure can be performed on the cancellation-free s-expanded DDD. Error controlling is carried out by enumerating the dominant terms from all the coefficient DDDs simultaneously according to certain criteria, until the generated terms, coming from different coefficient DDDs, well approximate the exact expressions in terms of magnitudes and phases.

Following the same strategy in [15], our approach also handles numerical cancellation. Since numerical canceling terms are extracted one after another, they can be eliminated by examining two consecutive terms.

We note that since the variant of DDD used by Verhaegen and Gielen in [14] does not satisfy the canonicity, and thus cannot apply the SP based algorithm.

V Experimental Results

The proposed approach has been implemented and tested on a number of practical analog circuits. For each circuit, DC analysis is first carried out using SPICE and our program reads in small-signal element values from the SPICE output. The algorithms described in [10, 12] are used to construct complex DDDs and s-expanded DDDs.

Table I summarizes the comparison of SP based and DP based algorithms for dominant term generation in terms of CPU time and memory usage. The implementation of our dynamic DP method is similar to the cache-based term generation algorithm in [14]. A total of 10000 dominant terms are generated for a number of test circuits ranging from more structured ladder circuits to less-structured such as $Cascode$ and $\mu A741$ amplifiers. In Table I, column 1, 2 and 3 list for each circuit, respectively, its name *Circuit*, the number of nodes *#nodes*, and the number of nonzero elements *#nonzero* in its circuit MNA matrix. Columns 4 to 5 and 6 to 7 show, respectively, the CPU time and memory usage for generating 10000 dominant terms by the DP based and SP based algorithms.

From Table I, we can see that the SP based algorithm outperforms the DP based algorithm for all the circuits in the memory usage. The difference becomes even more significant for circuits with regular structures like ladder circuits. For *rclad100*, the memory usage of the SP method is about a factor of 7 smaller than that of DP based method. On the other hand, dynamic programming may often be faster than the SP based algorithm.

But for ladder circuits, the SP based algorithm consistently outperforms DP based method in terms of CPU time and memory usage.

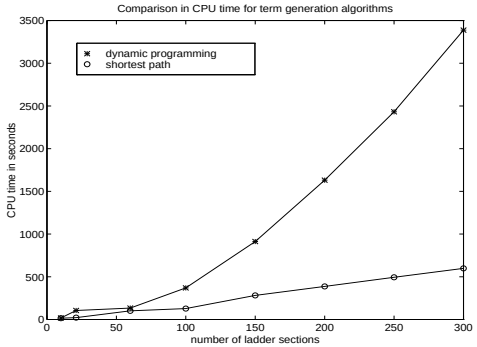

Fig. 10. CPU time vs number of ladder sections.

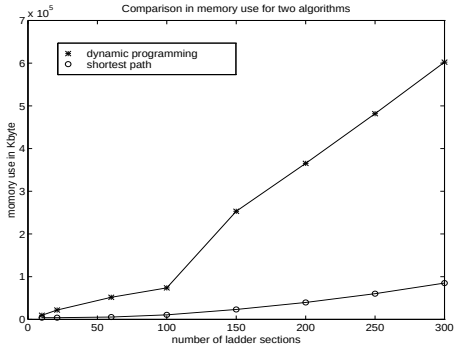

Fig. 11. Memory use vs number of ladder sections.

Fig. 10 and Fig. 11 shows the CPU time and memory usage for different ladder circuits. The CPU time increases almost linearly with the size of ladder circuits for both algorithms.

The memory usage of DP based method increases linearly with the sizes of ladder circuits, while the SP-based algorithm takes much smaller amount of memory for the same set of ladder circuits. This picture will become even worse for DP based method when more terms are generated as more memory will be used for caching the generated terms at each 1-edge pointed DDD vertex.

VI Conclusions

An efficient approach is proposed to construct cancellation-free DDD representations of symbolic expressions for ac characteristics of large linear analog circuits. Inspired by the work [14], we also implemented a dynamic programming based dominant term generation algorithm based on DDD graphs. We showed that the algorithm works only when certain properties are held in the DDD graphs and it is not applicable to cancellation-free s-expanded DDDs in general. Experimental results indicate that our shortest path based algorithm is more memory efficient than the dynamic programming based algorithm and thus is better positioned to handle large analog circuits.

References

[1] Y. Chen and J. White, "A quadratic method for nonlinear model order reduction", in *Proc. International Conference on Modeling and Simulation of Microsystems*, pp. 477–480, 2000.

[2] T. Cormen, C. E. Leiserson and R. L. Rivest, *Introduction to Algorithms*, The MIT Press, Cambridge, Massachusetts 1990.

[3] G. Gielen and W. Sansen, *Symbolic Analysis for Automated Design of Analog Integrated Circuits*, Kluwer Academic Publishers, 1991.

[4] G. Gielen, P. Wambacq and W. Sansen, "Symbolic analysis methods and applications for analog circuits: A tutorial overview", *Proc. IEEE*, vol. 82, no. 2, pp. 287–304, Feb. 1994.

[5] J.-J. Hsu and C. Sechen, "DC small signal symbolic analysis of large analog integrated circuits", *IEEE Trans. Circuits and Systems-I: Fundamental*, vol. 41, no. 12, pp. 817–828, Dec. 1994.

[6] K. Kundert, "A formal top-down design process for mixed-signal circuits", in *Analog Circuit Design*, R. J. van de Plassche, J. H. Huigsing and W. M. C. Sansen (eds), Kluwer Academic Publishers, 2000.

[7] J. R. Phillips, "Automatic extraction of nonlinear circuit macromodels", in *Proc. IEEE Custom Integrated Circuits Conference*, pp. 451–454, 2000.

[8] M. Rewienski and J. White, "A trajectory piecewise-linear approach to model order reduction and fast simulation of nonlinear circuits and micromachined devices", in *Proc. IEEE Int. Conf. Computer-Aided Design (ICCAD)*, pp. 252-258, Nov. 2001.

[9] P. Sanniti and N. N. Puri, "Symbolic network analysis—an algebraic formulation", *IEEE Trans. Circuits and Systems*, vol. 27, no. 8, pp. 679–687, Aug. 1980.

[10] C.-J. Shi and X.-D. Tan, "Canonical symbolic analysis of large analog circuits with determinant decision diagrams", *IEEE Trans. Computer-Aided Design*, vol. 19, no. 1, pp. 1–18, Jan. 2000.

[11] C.-J. Shi and X.-D. Tan, "Efficient derivation of exact s-expanded symbolic expressions for behavioral modeling of analog circuits", in *Proc. IEEE Custom Integrated Circuits Conf. (CICC)*, pp. 463–466, 1998.

[12] C.-J. Shi and X.-D. Tan, "Compact representation and efficient generation of s-expanded symbolic network functions for computer-aided analog circuit design", *IEEE Trans. Computer-Aided Design*, vol. 20, No. 7, pp. 813-827, July 2001.

[13] X.-D. Tan and C.-J. Shi, "Interpretable symbolic small-signal characterization of large analog circuits using determinant decision diagrams", in *Proc. Design, Automation and Test in Europe (DATE'99)*, pp. 448–453, Munich, Germany, Mar. 10-13, 1999.

[14] W. Verhaehen and G. Gielen, "Efficient DDD-based symbolic analysis of large linear analog circuits", in *Proc. ACM/IEEE 38th Design Automation Conference (DAC)*, pp. 139–144, Las Vegas, June 2001.

[15] P. Wambacq, G. Gielen and W. Sansen, "A cancellation-free algorithm for the symbolic simulation of large analog circuits", in *Proc. IEEE Int. Symp. Circuits and Systems*, pp. 1157–1160, May 1992.

[16] Q. Yu and C. Sechen, "A unified approach to the approximate symbolic analysis of large analog integrated circuits", *IEEE Trans. Circuits and Systems*, vol. 43, no. 8, pp. 656–669, Aug. 1996.

TABLE I
COMPARISON OF SHORTEST-PATH AND DYNAMIC-PROGRAMMING BASED ALGORITHMS.

Circuit	#nodes	#nonzero	Dynamic Programming		Shortest Path	
			CPU time(sec.)	Memory use(kb)	CPU time(sec.)	Memory Use(kb)
rclad10	8	31	17.3	9560	14.8	4048
rclad21	22	64	105.1	21816	21.5	3976
rclad60	61	181	133.8	51784	101.0	5432
rclad100	101	301	369.6	73832	172.8	10568
rclad150	151	451	912.2	253072	281.7	23280
rclad200	201	601	1630.9	365264	387.3	39616
rclad250	251	751	2431.3	481688	493.8	60160
rclad300	301	901	3388.4	602360	598.0	85088
rctreeA	40	119	41.4	38840	45.6	12880
rctreeB	53	158	132.9	41472	60.4	14304
Cascode	14	76	21.3	34880	620.1	28696
$\mu A741$	23	90	50.6	78024	1412.2	69184
bigtst	32	112	91.7	40808	144.1	12496