Analog Integr Circ Sig Process (2013) 74:331–343
DOI 10.1007/s10470-011-9773-8

A survey on binary decision diagram approaches to symbolic analysis of analog integrated circuits

Guoyong Shi

Received: 1 December 2010 / Revised: 9 June 2011 / Accepted: 30 August 2011 / Published online: 13 September 2011

Abstract Applying symbolic techniques for analog circuit analysis is a traditional research subject, which has lasted for over half a century. The past decade has witnessed a significant advancement of the symbolic techniques developed specifically for large analog integrated circuits. The key methodology introduced is a data structure called binary decision diagram (BDD) which was established originally for logic design and verification. The application of the BDD technique for analog circuit analysis has the following features: (1) It is a compact data structure so that data redundancy in symbolic analysis can be eliminated. (2) It provides a mechanism for implicit enumeration method so that exhaustive enumeration commonly performed in symbolic analysis can be avoided. (3) Numerical evaluation on a BDD can be made extremely efficient, making it an excellent means for repetitive analysis. More advanced features are yet to be explored. This survey brings together the significant research results published in the past decade and provides a tutorial overview on the basic principles of applying BDD to analog circuit analysis. Some new directions that are potentially valuable for developing future analog design automation tools are discussed and a design example is given to illustrate the application of symbolic techniques.

Keywords Analog integrated circuits · Binary decision diagram (BDD) · Design optimization · Sensitivity · Symbolic analysis

G. Shi (✉)
School of Microelectronics, Shanghai Jiao Tong University, Shanghai, China
e-mail: shiguoyong@ic.sjtu.edu.cn

1 Introduction

A nonlinear circuit can in general be described by the following differential equation

$$f\left(x(t), \frac{dx(t)}{dt}, u(t), t\right) = 0, \tag{1}$$

where $x(t)$ is the state of the circuit which describes the nodal voltages or branch currents introduced necessarily for circuit analysis and $u(t)$ stands for the externally applied voltage or current sources. A SPICE simulator is capable of numerically solving the differential equation and providing a transient analysis result given an initial condition.

Small-signal analysis is based on the linearization of the nonlinear devices around a pre-characterized circuit operating point. Linearization of the nonlinear differential equation 1 would produce the following linear differential equation

$$C\frac{dx(t)}{dt} + Gx(t) = Fu(t), \tag{2}$$

where a time-invariant circuit is assumed. The construction of a small-signal model is mainly for the alternate current (AC) analysis when the circuit is excited by a small magnitude signal with varying frequency and phase components. Taking Laplace transform of the Eq. 2 produces the following algebraic equation

$$(Cs + G)X(s) = FU(s), \tag{3}$$

where s is the complex Laplace variable $s = \sigma + j\omega$. By choosing an appropriate output, one can write the output equation as

$$Y(s) = LX(s). \tag{4}$$

The input–output transfer function is therefore described by the following equation

$$Y(s) = L(Cs + G)^{-1}FU(s). \tag{5}$$

The coefficient of $U(s)$, defined by $H(s) := L\,(Cs + G)^{-1}F$, is the transfer function of the original circuit with the selected input and output.

The frequency-dependent gain and phase information is among the most valuable for analog designers. For example, shown in Fig. 1 is the AC response of a CMOS operational amplifier (op-amp) circuit given certain sizing and biasing. The usefulness of the frequency response plots lies in the fact that the designer can easily read many frequency-domain design metrics from the plots, such as DC gain, bandwidth, unity-gain frequency, gain-margin, and phase-margin, etc., together with rough information on the distribution of the poles and zeros. Despite the usefulness, such frequency response curves do not provide the designer any information on how to optimize devices for AC performance improvement.

The above consideration is probably the key thrust for the research on *symbolic* circuit analysis. Comparing to symbolically solving a nonlinear differential equation as given in (1) in the time-domain, symbolically solving a linearized equation in the frequency domain (or so called the s-domain) as given in (3) is tractable. Hence, for over half a century, symbolic circuit analysis mainly has been performed in the AC domain, and its usefulness is unquestionable [6, 7, 16]. The key research problems in this area are mainly the computational efficiency issues and tool functionality. The CAD researchers are responsible for providing easy to use and helpful tool utilities acceptable to general analog designers.

Important performance parameters to be considered in the design of an analog circuit components like an op-amp include power dissipation, maximum allowable capacitive load, open-loop voltage gain, unity-gain frequency, output voltage swing, settling time, input flicker noise and thermal noise, power supply rejection ratio (PSRR), common-mode rejection ratio (CMRR), supply capacitance, and die area,

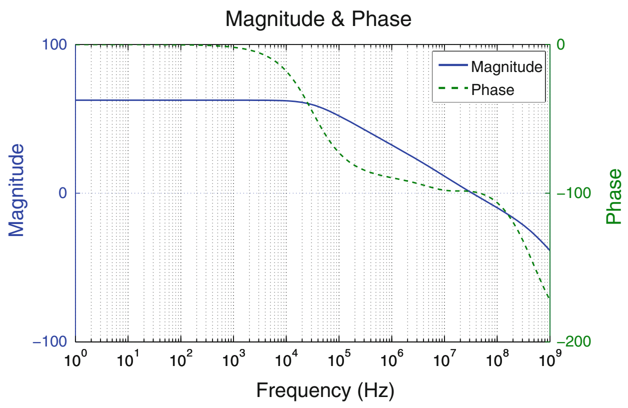

Fig. 1 A frequency response example

etc. [8]. Although a symbolic AC transfer function $H(s)$ cannot provide valuable information for all the design metrics listed above, it still can be useful in many aspects.

For example, an AC plot as shown in Fig. 1 results from evaluating a symbolic transfer function $H(s)$ by assigning to the symbols (or parameters) their numerical values. If one selects a symbol and changes its value within a range, it is possible to develop a graphical user interface to visualize the instantaneously varying AC response curves, by which the effect of changing a circuit parameter on the circuit behavior can be monitored. Doing the same inspection on a SPICE simulator would have to run SPICE simulation repeatedly, which is apparently less efficient. Along the same line, a symbolic tool even can derive analytical sensitivity expressions for $H(s)$ with respect to (w.r.t.) any circuit parameter, including MOS device sizes.

As another application, one may attempt to derive *interpretable* byproducts from a symbolic transfer function, such as simplified transadmittances and input–output impedances, etc., as normally done by an analog designer. Some researchers even propose to derive approximate symbolic poles and zeros [10, 21]. However, due to the inherent computational difficulties involved, interpretable simplification and symbolic pole/zero extraction must incorporate nonstandard postprocessing procedures and subjective judgement of the tool developers, which might not be fully apprehensible by circuit designers.

It is debatable whether or not a symbolic CAD tool should provide *exact* symbolic expressions or *approximate* but *interpretable* results. The ultimate criterion is by the acceptance of general analog designers. This survey is focused on the recently developed *exact* methods that are capable of analyzing analog circuit blocks containing up to 40 and more MOS transistors. Although exact symbolic results are not interpretable in any sense, it can be useful for purposes such as sensitivity analysis. On the other hand, because no approximation is made, the design space exploration capability is superior based on exact analysis.

Because managing exact symbolic expressions for analog circuits involving a dozen of transistors would exceed the capability of all traditional symbolic methods introduced such as in [16], it would not be possible without using more powerful modern data structures. The computation technique called binary decision diagram (BDD), which was once developed for digital circuit synthesis and verification [2], is such an excellent candidate. The promising feature of BDD for symbolic circuit analysis originates from the fact that the complexity of exponentially increasing number of terms can be suppressed to such a level that most practically designed op-amp circuits can be handled in reasonable time and memory.

The purpose of this survey is to review the significant research milestones made in the past decade meanwhile present a tutorial on the technical details that are not widely appreciated due to the delicacy involved in the implementation of BDD. It is also anticipated that analog designers might be interested in learning the state-of-the-art of the emerging CAD technology.

The application of BDD has been very successful in the logic synthesis field in the past 20 years. However, its applications in other areas have been very limited. In the author's opinion, the main reasons are probably two-fold; on the one hand, formulating a general enumeration problem in BDD requires creative thinking, and on the other hand, implementing a problem-specific BDD program requires another kind of skills. For the tutorial purpose, this survey starts from solving a simple circuit example by some representative symbolic methods in Sect. 2 and proceeds to their BDD formulations in Sect. 3, where two well-developed BDD methods are explained in detail. More advanced symbolic techniques intended for application are introduced in Sect. 4, including a recently developed hierarchical method, the notion of symbolic sensitivity and its computation, and finally a design example that illustrates the application of symbolic sensitivity for circuit optimization. Concluding remarks are made in Sect. 5.

2 Basic methods for symbolic analysis

Numerous symbolic analysis methods have been proposed in the literature before the year of 1990. They can largely be categorized into *algebraic* methods and *topological* methods. Most of the methods are obsolete in the sense that they are either inefficient or inflexible for a modern treatment. The monograph by Lin [16] published in 1991 documented most of the traditional methods for symbolic circuit analysis. Other overview works on this subject are [6, 7].

We use the common source MOS amplifier shown in Fig. 2 to introduce two representative symbolic analysis methods that can be reformulated into more advanced algorithms with BDD. Shown in Fig. 3 is its small-signal model with the gate-drain parasitic capacitor C_{gd} introduced explicitly to make the analysis nontrivial.

A simple manual analysis derives the following input–output voltage transfer function

$$\frac{v_{out}}{v_{in}} = \frac{sC_{gd} - g_m}{g_{ds} + s(C_L + C_{gd})}. \tag{6}$$

Formally, we can use either an algebraic method, which is used in numerical circuit simulators (such as SPICE), or a graphical method to derive the same symbolic solution. The well-known algebraic method is the so called modified nodal analysis (MNA) method [12].

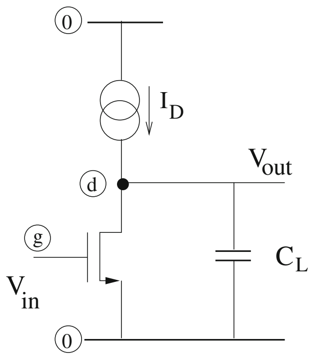

Fig. 2 Common source MOS amplifier

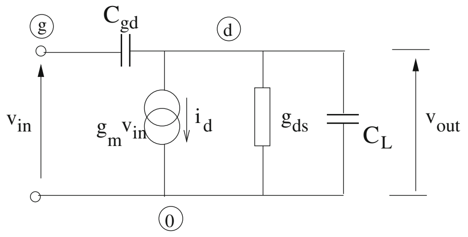

Fig. 3 Small-signal model of the common source MOS amplifier

With the MNA formulation we get the following linear equation

$$\begin{pmatrix} sC_{gd} & -sC_{gd} & 0 & 1 \\ -sC_{gd} & s(C_{gd} + C_L) + g_{ds} & 1 & 0 \\ g_m & 0 & -1 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} v_g \\ v_d \\ i_d \\ i_{vin} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ v_{in} \end{pmatrix}, \tag{7}$$

where v_g and v_d are respectively the small-signal voltages at gate and drain, i_d is the drain current and i_{vin} is the current flowing through the input voltage. Numerically, this equation can be solved by Gaussian elimination or the standard LU factorization. Symbolically, this equation can be solved by Cramer's rule, which is written as follows:

$$v_{out} = \frac{N(s)}{D(s)} v_{in}, \tag{8}$$

where N and D are two determinants given below

$$N(s) = \begin{vmatrix} sC_{gd} & 0 & 0 & 1 \\ -sC_{gd} & 0 & 1 & 0 \\ g_m & 0 & -1 & 0 \\ 1 & 1 & 0 & 0 \end{vmatrix}, \tag{9}$$

$$D(s) = \begin{vmatrix} sC_{gd} & -sC_{gd} & 0 & 1 \\ -sC_{gd} & s(C_{gd} + C_L) + g_{ds} & 1 & 0 \\ g_m & 0 & -1 & 0 \\ 1 & 0 & 0 & 0 \end{vmatrix}. \tag{10}$$

It is easily verified that the quotient of the two determinants evaluates to the transfer function in (6).

The same circuit problem can also be solved graphically (or topologically) by enumerating certain spanning trees of a properly formulated graph shown in Fig. 4, which contains two pairs of edges, $v_{out} - v_{in}$ and $v_{in}^{'} - g_m v_{in}^{'}$, edge C_{gd}, and edge $\hat{g}$. The pair $v_{out} - v_{in}$ stands for the input–output relationship, while the pair $v_{in}^{'} - g_m\, v_{in}^{'}$ stands for the transconductance. The edge $\hat{g}$ is a composition with $\hat{g} = g_{ds} + sC_L$.

According to the rules established in [26], the input–output (I/O) pair is modeled as a controlled source. For the example with a voltage input and a voltage output, the I/O pair is treated as a *voltage controlled voltage source* (VCVS), with the output being the *controlling* voltage source (VC) and the input being the *controlled* voltage source (VS) (see an explanation in [26]). The novel aspect of the enumeration rules that differentiate them from those classic spanning tree approaches, such as the *two-tree* method used in [38], is in the treatment of the four dependent sources. Those classic methods have problem with the dependent sources other than VCCS in that extra conversions must be made to handle those dependent sources of VCVS, CCCS, and CCVS, while the rules developed in [26] have lifted such restrictions.

A rule-based enumeration method developed in [26] generates the spanning trees shown in Fig. 5, from which four *signed* product terms are determined, two from the two spanning tree pairs (Fig. 5(a, b)) and two from the two pairs of spanning trees (Fig. 5(c, d)).

The rules established in [26] are literally lengthy (hence not repeated here), but can be implemented without difficulty [4]. The four spanning tree objects shown in Fig. 5 would generate the following four *signed* product terms summing up to zero, i.e.,

$$X \cdot g_m - X \cdot sC_{gd} + sC_{gd} + \hat{g} = 0, \tag{11}$$

where X represents the coefficient of the VCVS pair, namely, $v_{in} = X \cdot v_{out}$. Equation 11 leads to

$$\frac{v_{out}}{v_{in}} = X^{-1} = \frac{sC_{gd} - g_m}{sC_{gd} + sC_L + g_{ds}}, \tag{12}$$

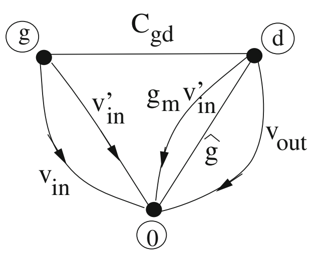

Fig. 4 The graph corresponding to the common source circuit for spanning tree enumeration

which is the same voltage transfer function derived in (6) for the common source circuit given in Fig. 2.

3 Binary decision diagrams for symbolic analysis

The two representative classes of methods mentioned in the preceding section would not be of practical use without using a superior data structure to manipulate large determinants or graphs. Here, by “large” we mean that the number of circuit nodes could be in the range of 20 to 40 or more, which would meet the design needs of practical analog IC blocks.

The past 10 years of research have validated that the BDD data structure proposed by Randal Bryant in 1986 [2] can be used as an efficient data structure for not only a compact representation of symbolic transfer functions, but also a means of efficient data manipulation and evaluation.

BDD was originally developed for efficient representation of logic functions, which traditionally are processed in the form of sum-of-products (SOP) or product-of-sums (POS) expressions. When the number of literals reaches a moderate level, the count of terms appearing in such expressions would reach the limit of computing resource. More critically, the SOP or POS expressions are not *canonical* in that more than one literally different boolean expressions exist for the same truth table. Bryant’s pioneering contribution in his 1986 paper [2] is the establishment of BDD *canonicity* that comes with an ordered list of symbols. An optimal symbol order is not sought in practice because the problem itself is NP-complete. Rather, heuristics are used in practice. The canonicity of BDD is in fact achieved by a *sharing* mechanism enforced in all BDD implementations, which induces many important applications in logic synthesis and formal verification [3]. The BDD canonicity can be used for a compact representation of symbolic transfer functions that used to be generated by enumeration. It is the inherent sharing mechanism of BDD that makes an explicit enumeration *implicit*, which contributes greatly to the enumeration efficiency. Furthermore, a BDD by itself is also a computation structure in which arithmetic operations are embedded in the data linkage. Consequently, advanced analytical operations (such as taking derivatives) can be implemented on a BDD easily and efficiently.

The application of BDD to symbolic circuit analysis emanates from a simple observation of the similarity between the boolean SOP representations of logic circuits and the algebraic SOP expressions arising from the determinant based algebraic method or the spanning tree based topological method for analog circuits. The immediate question along this line is: *How to construct a BDD for a*

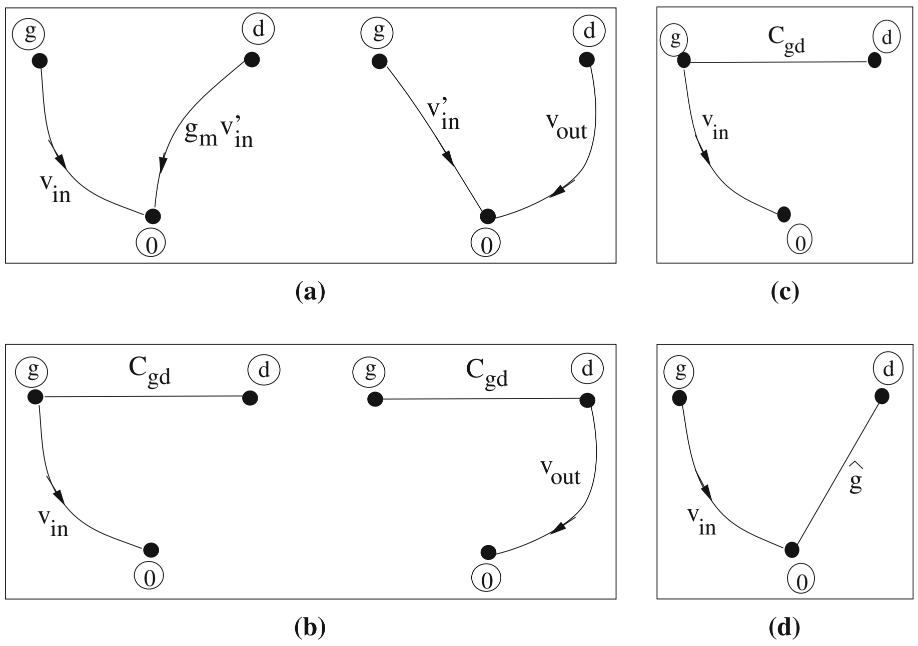

Fig. 5 Two spanning trees and two spanning tree-pairs for the graph in Fig. 4: (a) tree pair 1 with the term $+X \cdot g_m$, (b) tree pair 2 with the term $-X \cdot sC_{gd}$, (c) tree 3 with the term $+sC_{gd}$, and (d) tree 4 with the term $+\hat{g}$

transfer function given a linearized circuit? In a standard BDD package for logic synthesis, a routine specifically designed for *If-Then-Else* (ITE) is sufficient for the purpose of the BDD construction (see [1]). However, directly applying such a package for analog circuits does not seem to be straightforward, unless some special problem transformation as done in [29] is performed. Analog circuits are typically analyzed algebraically or topologically, as stated earlier. Hence, a suitable approach would be to construct BDDs directly from algebraically or topologically formulated objects, such as an MNA matrix or a graph representation of a circuit. In this regard, a BDD way of thinking is considered crucial to the formulation of an efficient construction algorithm.

3.1 Determinant decision diagram

Shi and Tan [22] contributed the pioneering work of applying BDD to symbolic circuit analysis. In that work, they developed algorithms for expanding a determinant using a set-based BDD package (called *Zero-suppressed BDD*, ZBDD) [18, 19]. As stated earlier, by Cramer's rule, the I/O transfer function can be expressed as the quotient of two determinants. So, as long as the expansions of the two determinants can be constructed in one BDD (shared), a symbolic representation of the transfer function is available in the computer memory. The authors of [22] named such a data structure determinant decision diagram (DDD).

The "sharing" mechanism in BDD can be implemented in computer programming by a *hash table*. The design of a hash scheme is nonstandard and very much implementation dependant. However, the efficiency of a BDD-based symbolic package is largely dependent on the implementation strategy of a hash scheme and a properly organized hash table.

Two operations must be defined when one would like to construct a BDD for a problem. In the case of a determinant, one may define the operations of "*Minor*" and "*Remainder*" with respect to any selected matrix element for the binary decisions [22]. Here, "*Minor*" refers to the operation of deleting a row and a column intersected at the selected element, while "*Remainder*" refers to setting the selected element to *zero*. This can simply be expressed by the following equation

$$\det A = (-1)^{i+j} a_{i,j} \cdot Minor(A, a_{i,j}) + Rem(A, a_{i,j}), \qquad (13)$$

where the binary decision is made at the ith row and jth column element a_{ij} and "*Rem*" stands for "*Remainder*". Equation 13 is obviously valid in the sense of linear algebra. Note that in this equation $Minor(A, a_{i,j})$ is a reduced dimensional determinant of dimension $(n - 1)$ while $Rem(A, a_{i,j})$ is still a determinant of dimension n, assuming that A is an $n \times n$ matrix. As long as either $Minor(A, a_{i,j})$ or $Rem(A, a_{i,j})$ is nonsingular, the binary expansion of (13) can be continued until the final determinants are either *scalar* or *singular*.

Two things crucial to the expansion process are: *order of expansion* and *minor sharing*, which make a DDD canonical [22]. The *order of expansion* refers to the notion of symbol ordering in BDD. One has to choose either an explicit (pre-chosen) order or an implicit (runtime) order for sequencing the determinant expansion. A pre-chosen order requires extra computation time, while a runtime order does not. The issue of *minor sharing* is most critical, because without sharing the minors, the expansion of (13) is nothing else than an explicit binary expansion, its exponential growth rate would not be favorable for the computer memory.

The possibility of *sharing* in determinant expansion can be illustrated by the following example. Suppose we have a 4 × 4 determinant

$$\begin{vmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{vmatrix}. \tag{14}$$

We observe that expanding the determinant by the two successive *Minor* operations on a_{11} followed by a_{22} would arrive at the lower-right 2 × 2 minor

$$\begin{vmatrix} a_{33} & a_{34} \\ a_{43} & a_{44} \end{vmatrix} = a_{33}a_{44} - a_{34}a_{43}. \tag{15}$$

The same minor is also arrived at by applying the other two successive *Minor* operations on a_{12} followed by a_{21}. Hence, the minor (15) can be shared in that it suffices to expand the 2 × 2 minor only once. This mechanism is the "*implicit enumeration*" we referred to earlier, which is the underlying secret that makes BDD efficient for large enumeration problems.

The design of a hash scheme has to take into account of the objects under operation based on the methodology chosen for symbolic circuit analysis. Both the determinant objects in the algebraic approach and the graph objects in the topological approach are the candidate objects for hash operations. Discussed next is the most critical component that one must consider for implementing a BDD-based symbolic package.

Since the main objects for sharing in determinant expansion are those sub-dimensional minors, the key issue one should consider is how to hash (or look up) the minors in the speediest way. Here comes with the technical subtlety in implementation. The original implementation strategy outlined in [22] missed a good property inherent in the *ordered* expansion, which was later discovered in [24]. It is stated here: *Given an expansion order, the row and column indices of minor are sufficient for uniquely identifying a minor, regardless of the minor elements*. This simple property is significant in that it can be used to greatly simplify the implementation of DDD, which is realized in the recent work [25].

With the advancement achieved by the principle of DDD and some recent improvements, circuits of the scale of op-amp μA741 and μA725 containing about 20 to 26 bipolar transistors can be analyzed in *exact* symbolic expressions without enforcing any approximation. But solving circuits of size larger than μA741 and μA725 remains challenging by one-layer DDD analysis. Therefore, more sophisticated techniques (like hierarchical analysis) must be incorporated, which will be discussed in Sect. 4.

A number of extensions based on the DDD algorithm have been published in the literature, such as [32, 23, 33, 30, 31, 28, 29], among others. Although these later works have made explorations in a variety of directions, such as introducing hierarchical analysis methods and an alternate BDD construction method [29], a theoretical complexity analysis of DDD computation has remained unknown until the recent work of [24], where the complexity of applying DDD to a full matrix is established. It is revealed that the essence of DDD is a significant reduction of exponential growth rate by introducing a properly implemented sharing engine. Other limitations in the past DDD research are: (1) The proposed DDD-based hierarchical analysis methods require loose interconnection of the partitioned circuit blocks, which causes limitation in applying such methods for very large analog op-amps containing more than 40 MOSFETs, in which no loose interconnection exists. (2) A full examination on what kind of tool functionality should be developed to maximally utilize the inherent efficiency made possible by a BDD data structure. Some recent attempts made for relieving the limitations will be reviewed further in Sect. 4 after we introduce another graph-based BDD method for symbolic circuit analysis.

3.2 Graph-pair decision diagram

The success of DDD as an application of BDD for a *compressed* representation of determinant expansion has inspired a new direction of work that uses BDD for representing the process of spanning-tree enumeration in a *non-exhaustive* way. Since the spanning-tree enumeration can be reformulated in sharable objects in the form of reduced subgraphs (to be explained later), we can again make use of the BDD data structure to enumerate spanning-trees *implicitly*. This basic consideration leads us to another novel BDD-based symbolic analysis tool that takes the conventional approach of graphical (topological) formulation, but reincarnated in the form of BDD.

Applying BDD to spanning-tree enumeration first appeared in the work of [4, 26] around 2007. This work was inspired by reformulating the Minty algorithm in a BDD. Enumerating all spanning-trees of a connected graph was studied by Minty in his one page paper [20] published in 1965, where Minty invented a *binary* graph decomposition process by applying one of the two operations "*In*" and "*Out*" to an ordered sequence of edges, with "*In*" meaning that the edge is retained and "*Out*" meaning that the edge is removed. Figure 6 shows an example of applying the Minty algorithm to enumerate the five spanning trees of a graph containing four edges. The operations follow the edge order of $e_1 < e_2 < e_3 < e_4$, where '<' reads *precedes*. A checking mechanism is needed to monitor whether a spanning tree has formed or whether the edges left are insufficient for forming a spanning-tree. It is apparent that the Minty algorithm is a binary process that enumerates all spanning trees exhaustively, which are T_1 (e_1, e_2), T_2 (e_1, e_3), T_3 (e_1, e_4), T_4 (e_2, e_3), and T_5 (e_2, e_4), as shown in the figure.

At the time 1965 Minty was not aware of sharing the subgraphs generated in the decomposition process. Therefore, what Minty obtained was an exhaustive binary decomposition procedure, whose exponential complexity does not allow for solving large graphs.

Considering from the perspective of BDD, the Minty algorithm can be modified slightly to incorporate the *sharing* mechanism in the binary decomposition process. The modification is made to the way the subgraphs are produced. In order to take the advantage of sharing, the subgraphs in the process of decomposition should be made comparable for isomorphism. A good way of achieving such a property is by *graph reduction* instead of *retaining* and *removing* edges. More specifically, when an edge is to be retained, one should *collapse* the edge into one node instead of retaining the edge, while an edge to be removed is simply *removed*. In this sense, the graph operations are still binary, but become edge *collapsing* and edge *removal*. As successive edge operations are applied, the original graph is gradually reduced while the numbers of edges in the subgraphs decrease. The reduction process is terminated if one of the following cases occurs: (i) only one node is left, (ii) the leftover edges are insufficient for completing a spanning-tree, or (iii) the subgraph becomes disconnected.

Figure 7 illustrates the application of the modified Minty algorithm to the same graph appeared in Fig. 6. We see that the middle subgraph in the third row (containing edges e_3 and e_4) is shared by two preceding subgraphs in the second row. Note that, without collapsing the edges e_1 and e_2, the two subgraphs in the middle of the third row in Fig. 6 are not identical. Since one binary decision is always applied to a selected edge, the selected edge can be the name of a BDD vertex. Also note that the three terminations marked by "1" in Fig. 7 are essentially a degenerated subgraph containing a single node, they can share one single BDD vertex to indicate the termination of spanning-trees. Putting these considerations together we arrive at the BDD shown in Fig. 8. The subgraph attached to each BDD vertex illustrates the subgraph under operation there.

Note that we use *solid* arrows in Fig. 8 for the "*In*" operations and *dashed* arrows for the "*Out*" operations. Therefore, the spanning trees represented in the constructed BDD can be read out as follows: Starting from the root vertex, we traverse downward to the vertex "1". If a solid arrow emerges from a vertex, the vertex name (i.e., the edge) makes an edge in a spanning tree. If a dashed arrow emerges from a vertex, that edge does not go to the spanning tree. There exist five paths starting from the root vertex and arriving at the vertex "1", which result in five spanning trees by the stated rule (see those listed in Fig. 8.) In this sense, we say that all the spanning trees for the given graph are *implicitly* listed in the constructed BDD.

Similarly to the determinant expansion where reduced dimensional minors obtained by the operations of "*Minor*" and "*Remainder*" find sharing among themselves, the intermediately reduced subgraphs generated by the operations of edge "*In*" (i.e., *collapsing*) and edge "*Out*" (i.e., *removal*) also find sharing among themselves. It is possible to devise a graph comparison mechanism to identify the

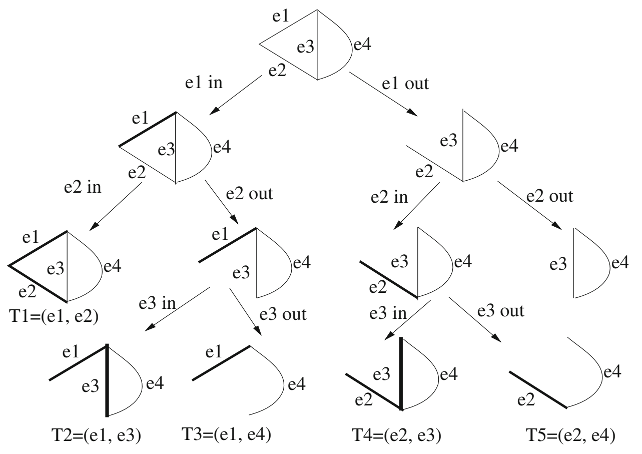

Fig. 6 Minty algorithm for enumerating all spanning trees of a four-edge graph

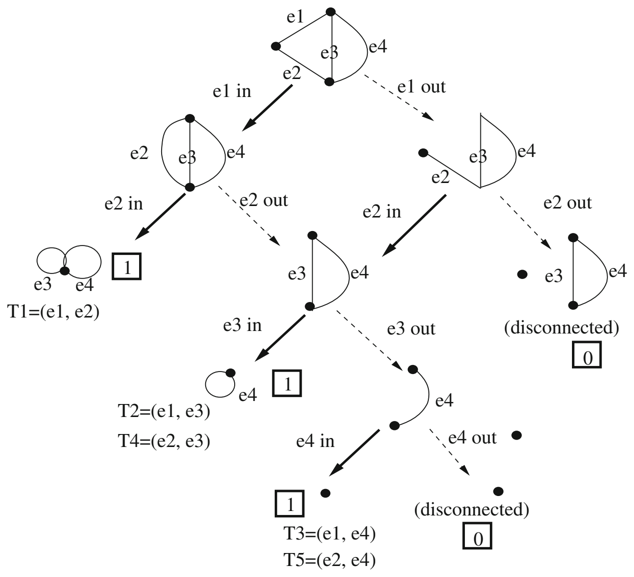

Fig. 7 Minty algorithm improved for sharing the subgraphs. The edges forming loops can be removed

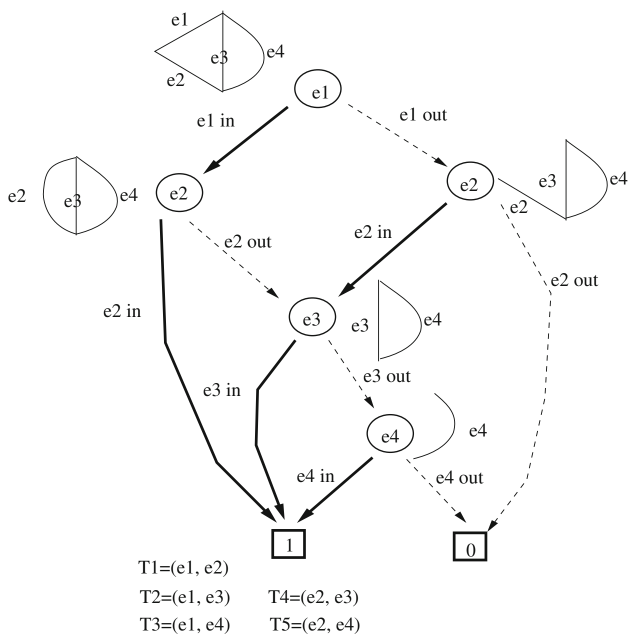

Fig. 8 Five spanning trees represented in a BDD

isomorphism of two equal dimensional subgraphs for sharing [4].

While promising, the above technique is not yet ready to be a symbolic tool for circuit analysis, because, as we mentioned earlier in the topological method for solving circuit problems, the existence of dependent sources in circuits complicates things. It prevents us from analyzing the circuit just by enumerating spanning-trees only. More sophisticated rules must be developed for those dependent graph edges so that the circuit analysis problem can still be formulated as a spanning-tree enumeration problem. It seems that there does not exist effective enumeration algorithms in the literature that are directly applicable for dealing with all four types of dependent sources. The work [26] has developed a new set of enumeration rules that enumerate all admissible spanning trees in the form of tree-pairs without making any conversion of the four types of dependent sources. More importantly, since the enumeration rules are easily amenable to graph reduction operations, a BDD-based graph reduction algorithm similar to the modified Minty algorithm can be developed.

In the new formulation, a notion called *pair of spanning trees* (or simply called *tree-pairs*) is introduced, which is analogous to the *two-tree* method, but not exactly the same, because the two-tree method is not directly applicable to all four types of dependent sources (see [38]). Not all spanning tree-pairs would contribute to a symbolic transfer function of a circuit; only those satisfying certain edge conditions are admissible [26]. The modified Minty algorithm reformulated in BDD can be extended to *implicitly* enumerate all admissible tree-pairs in that at each decision step a set of rules are enforced for a single edge or a pair of edges (see the details developed in [4, 26]). The resulting BDD is termed *Graph-Pair Decision Diagram* (GPDD). The detailed implementation procedures are presented in [4]; the experimental results reported there show that the GPDD method was able to solve those circuits solvable by DDD with comparable memory and time efficiency, sometimes a better performance was observed if a heuristic symbol order happened to be good.

In summary, two BDD-based symbolic algorithms have been developed in the past decade, with radically different problem formulations. The critical difference between the two algorithms lies in the definition of symbols and its consequences. In DDD, the MNA matrix entries are the symbols, but they are not directly the circuit elements, as seen from the example matrix in (7). For example, the (2, 2) entry of the coefficient matrix is $a_{22} = s(C_{gd} + C_L) + g_{ds}$, which contains three circuit elements but is treated as a single symbol in DDD. Also the same circuit element C_{gd} appears in the four matrix entries of a_{11}, a_{12}, a_{21} and a_{22}, but these entries are treated as different symbols in DDD. A negative consequence of the DDD symbol definition is the problem of term cancellation, i.e., the existence of identical product terms with opposite term signs. In contrast, the symbols in the GPDD representation are always the circuit elements and the tree enumeration based construction naturally excludes the term cancellation problem [4, 26]. Numerically speaking, term cancellation might not be a very serious problem. But some applications do require the elimination of term cancellation, such as in the application of variational analysis based on interval algebra [11]. Moreover, the different symbol definitions could affect the implementation of some advanced analysis functionalities, such as sensitivity analysis to be discussed later [17, 27].

4 More advanced methods and applications

The reader should be aware of how a BDD improves the efficiency for those problems with exponential complexity. It is the *sharing* mechanism enforced in BDD that helps reduce the enumeration complexity. Whenever a substructure is shared, the detailed operations on the substructure are performed only once, rather than being repeated as in those exhaustive enumeration methods. This fact must be observed in all successful applications of BDD.

However, it is worth pointing out that the application of a BDD does not change the nature of an exponentially complex problem; that is, a set of problems with exponential complexity would in general remain exponentially complex regardless of whether or not a BDD is used. A BDD could possibly reduce the base factor of exponential increase. This fact has been less well understood for a while until the work [24] recently made it clear by analyzing the computational complexity of DDD. It is shown that the optimal DDD size (i.e., the number of BDD vertices) for any $n \times n$ full matrix (without any zero element) is $(n\,2^{n-1})$, which apparently is still an exponentially increasing number, but the growth rate is significantly lower than that by explicitly expanding a determinant.

Reducing the base factor of exponential growth simply implies that much larger problems can be solved by a BDD reformulation. However, it does not mean that arbitrarily large problem always can be solved because the nature of exponential complexity has not been changed in general.

4.1 Hierarchical analysis methods

The two BDD-formulated symbolic methods, DDD and GPDD, have radically improved the symbolic problem-solving capacity, but the capacity limitation still remains when one comes to solve op-amp circuits containing more than 30 bipolar or MOS devices. This limit must be lifted because many practical analog circuit blocks contain more than this number of transistors. Among all possibilities, hierarchical methods are those that suit the BDD formulation the best.

Two hierarchical schemes were developed based on DDD in [32] and [30]. The basic idea was a utilization of the Schur decomposition of a block matrix. When the block submatrices are loosely coupled, the DDD-based hierarchical solution is feasible and efficient. However, this technique may not be equally efficient for circuits without a partition with loose interconnections, such as those large op-amp circuits containing over 30 active devices. A research effort that circumvents the difficulty is by introducing a novel concept called *symbolic stamps*, which was studied in the recent work [35]. This technique makes use of a hybrid data structure with a GPDD used for building the symbolic stamps and a DDD used for symbolically analyzing the assembled MNA matrix comprising of the symbolic stamps.

The key idea of hybrid decision diagram analysis proposed in [35] can be described by the diagram illustrated in Fig. 9, where the subcircuit modules at the lower level are treated by the GPDD technique to construct the symbolic stamps while the upper layer is an MNA matrix which assembles the symbolic stamps and is solved by the DDD technique. The GPDD technique can very easily be extended to a multi-root form to represent a multi-port symbolic stamp. The experimental results reported in [35] show that an op-amp circuit containing 44 MOS transistors can be solved exactly using the hierarchical algorithm. It is emphasized that no loose coupling between the subcircuit modules is assumed in this new hierarchical analysis method.

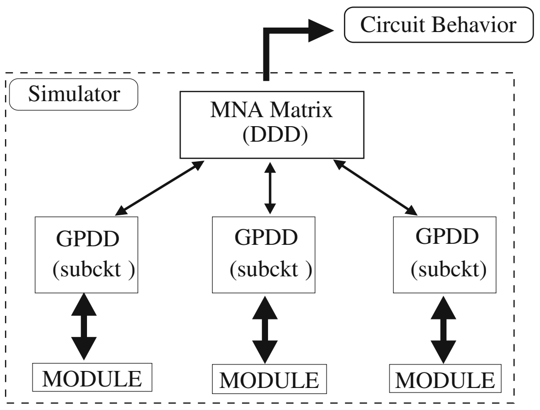

Fig. 9 Hierarchical analysis structure [35]

Table 1 summarizes a comparison of the performances reported by those published works on hierarchical symbolic analysis. Except for the work [9] which implemented an *approximate* hierarchical analysis scheme, the symbolic stamp method of [35] is superior for *exactly* analyzing those large amplifier circuits containing around 50 MOSFETs.

4.2 Symbolic sensitivity analysis

One important application of *exact* symbolic analysis is for analytical sensitivity analysis. The circuit sensitivity analysis is traditionally known to be useful. Numerical circuit simulators provide some standard forms of sensitivity analysis [34]. One standard numerical sensitivity analysis is to compute the DC or AC small-signal sensitivity to all nonzero device parameters. Such computation is not only costly but also has very limited use, because the designer is usually interested in the sensitivity to certain selected parameters, and moreover, the designer would like to know how the change of the selected parameters affects the circuit performance. For this a quick turnaround in computation is required. While Lee et al. studied a moment-based approximate computation technique for approximate analysis of pole-zero sensitivities [14], to the author's knowledge there hardly exists any other extensive study on the subject of symbolic sensitivity computation and applications.

The symbolic AC small-signal sensitivity computation was lightly mentioned in the work on DDD [22], but no serious application was considered. A preliminary research on sizing by symbolic AC sensitivity was attempted in the work of Yang et al. [36, 37] by embedding a variant of DDD in an analog synthesis flow. A serious fact neglected in [36, 37] is that the sizing sensitivity is not a quantity

Table 1 Performance comparison of some representative hierarchical methods

Publication	Max Ckt size (#Q/J/M)	Method	Accuracy
[32]	20 (BJT)	DDD + Schur decomp.	Exact
[5]	26 (BJT)	Regularity + Sharing	Exact
[9]	83 (MOS)	Ckt reduction + Two-graph	Approx.
[30]	26 (BJT)	DDD + De-cancellation	Exact
[35]	44 (MOS)	DDD + GPDD	Exact

local to each active device, as the size adjustment of one active device would affect the biasing condition of other active devices. The work of [27] studied an application of symbolic sensitivity for characterizing variational effects of analog circuits. The more recent work [17] further investigated the application of GPDD for MOS device sizing analysis, where it was found that the plots of sensitivity curves of $H(s)$ w.r.t. device sizes could exhibit the relative strength of sensitivity each device has on the frequency response, even with certain revelation of the pole-zero dependence on the active devices. These preliminary research works have indicated the usefulness of symbolic AC sensitivity for analog circuit design.

It is worth emphasizing that a BDD-represented symbolic transfer function is particularly advantageous for the purpose of analytical sensitivity computation, because the sensitivity-related data manipulation can be easily implemented on the underlying BDD data structure. For example, tracing or eliminating a symbol in a BDD is a simple operation on BDD with hardly any computational overhead, which is explained now.

As we commented before, the GPDD data structure has a special property that each symbol is a circuit element. Hence, the implementation of sensitivity calculation based on a GPDD can further be simplified in that even no chained derivative operation is needed. This fact can be illustrated by a simple example. Consider the SOP expression

$$E(a,x,b,c) = axb + xbc + abc, \tag{16}$$

where the variables are ordered in $a < x < b < c$. It is readily derived that

$$\begin{aligned}\mathrm{Sens}(E,x) &= \frac{x}{E}\cdot\frac{\partial E}{\partial x} = \frac{x}{E}(ab+bc)\\ &= \frac{axb+xbc}{E} = \frac{E - E|_{x=0}}{E},\end{aligned} \tag{17}$$

where in the last expression $E|_{x=0}$ represents those product terms not involving x. Hence, the normalized sensitivity of the SOP expression w.r.t. the variable x can be calculated by a truncated SOP (retaining those terms involving x) divided by the SOP itself.

The above example indicates the necessity of tracing a sensitivity variable, which can be performed fairly easily on a BDD. Shown in Fig. 10(a) is the BDD that represents the SOP expression (16). The BDD given in Fig. 10(b) results from retaining only those terms containing the sensitivity variable x. The sensitivity operation is simply to terminate all "dashed" arrows of the vertices containing the sensitivity variable at the terminal vertex "0". Hence, in Fig. 10(b) all the dashed arrows of vertices containing "x" are terminated at the vertex "0". The normalized sensitivity is then the quotient of the BDD root value of Fig. 10(b) divided by the BDD root value of Fig. 10(a).

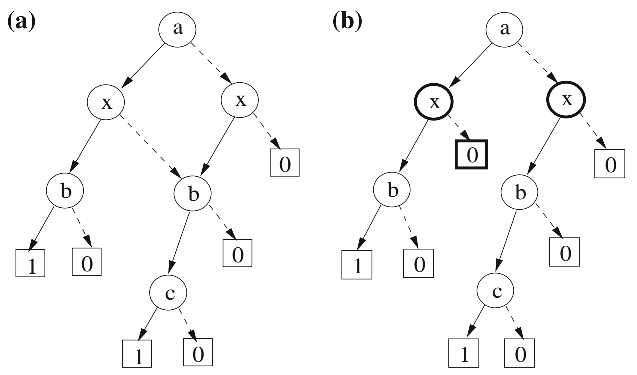

Fig. 10 (a) The BDD before sensitivity operation. (b) The BDD after the sensitivity operation

The implementation of the sensitivity operation on a DDD is analogous, except for a few more chained operations needed to take care of the composite symbols that are functions of the sensitivity variable. More details are discussed in the recent work [15], where the sensitivity operations are performed on both GPDD and DDD.

4.3 Design application

The design techniques for a basic two-stage CMOS op-amp (shown in Fig. 11) were described in detail in the classical paper [8]. The pole-splitting compensation capacitor C_c and the nulling resistor R_z are important elements for shaping the frequency response. This design technique is still used in more advanced circuit configuration [13].

This op-amp circuit consists of eight MOSFETs. Initially, the transistors are sized at random just to make sure that all are biased in the saturation region without considering the design goals.

We have developed a symbolic simulator that can calculate the AC sensitivities with respect to all MOS device sizes. Plotted in Fig. 12(a, b) are the real and imaginary parts of the AC sensitivities to all device sizes. We note that the curves exhibit oscillations at certain frequency points, which in fact are locations corresponding to the poles and zeros existing nearby, as can be verified by HSPICE simulation. More interestingly, these sensitivity curves show different magnitudes, meaning that each device has different sensitivity strength for the performance metrics like gain and phase (or poles and zeros). Such perception always exists in the analog designer's mind, which can now be visually exposed with the aid of a symbolic tool. The sensitivity plots clearly reveal that the dominant poles or zeros are mainly dependent on the device M_6 (at output) and the compensation capacitor C_c, because the corresponding sensitivity curves vary relatively more strongly than others nearby the pole locations.

Besides the graphical exhibition of the circuit behaviorial dependence on devices, the analytical AC sensitivity can also be used for optimizing the selected devices to meet a target, say, the phase margin. For example, we choose to optimize the value of the nulling resistor R_z in the Miller op-amp (Fig. 11) to achieve a better phase margin.

After the initial sizing, the phase margin of the amplifier was only about 25° by HSPICE simulation. Part of the reason is because the first zero is away from the second dominant pole (see Fig. 13(a)). We would like to increase the phase margin to 60° (a typical design requirement) and decide to achieve the goal by optimizing the compensation resistor R_z using the AC sensitivity. The update formula of R_z is given by

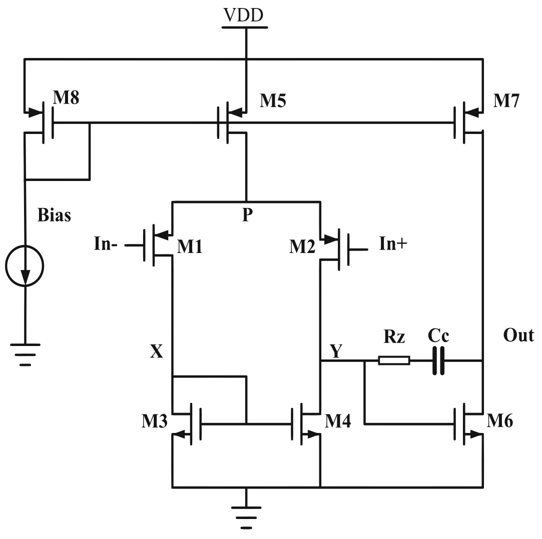

Fig. 11 Two-stage operational amplifier with a nulling resistor R_z in series with a compensation capacitor C_c

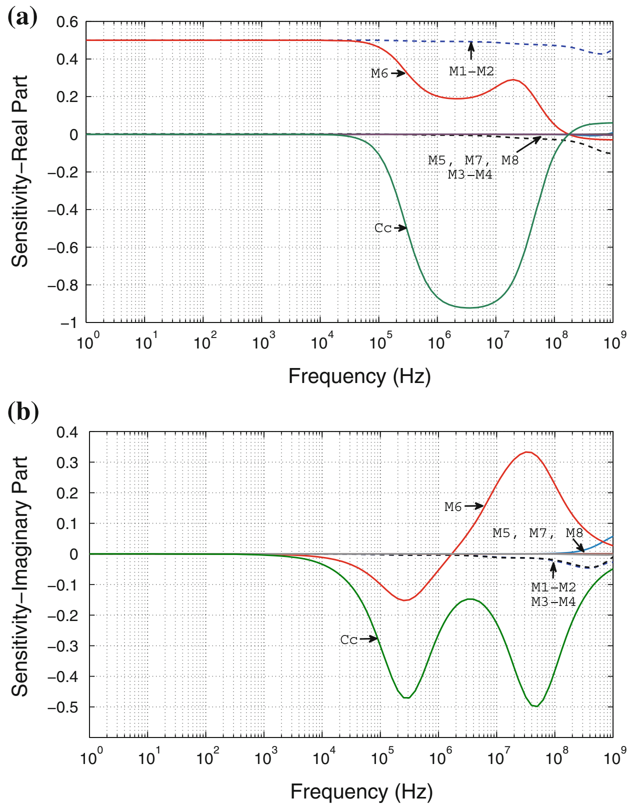

Fig. 12 (a) Plot of Re{Sens($H(s)$, W_k)}. (b) Plot of Im{Sens($H(s)$, W_k)}

$$R_z^{(n+1)} = R_z^{(n)} + \frac{\partial(R_z^{(n)})}{\partial(\angle H(s))}\Delta(\angle H(s)) = R_z^{(n)} + \frac{R_z^{(n)}\cdot\Delta(\angle H(s))}{\mathrm{Im}\left\{\mathrm{Sens}(H(s), R_z^{(n)})\right\}}, \quad (18)$$

where Sens($H(s)$, $R_z^{(n)}$) stands for the normalized sensitivity of $H(s)$ w.r.t. R_z at the nth iteration and is computed at the unity-gain frequency [15].

After four iterations (see Table 2), the phase margin is almost 60°. We see from the plot of Fig. 13(b) that the first zero now has moved closer toward the second dominant pole after optimization. A similar optimization procedure using the AWE-computed pole-zero sensitivity was introduced in [14]. Comparing to that work, the amount of computation is much less using the symbolic AC sensitivity for optimization.

5 Concluding remarks

In about 30 years of CMOS analog integrated circuit design, the dominating design tool has been unexceptionally by SPICE. Despite the continuous research effort on symbolic analysis methods, hardly any of them has grown

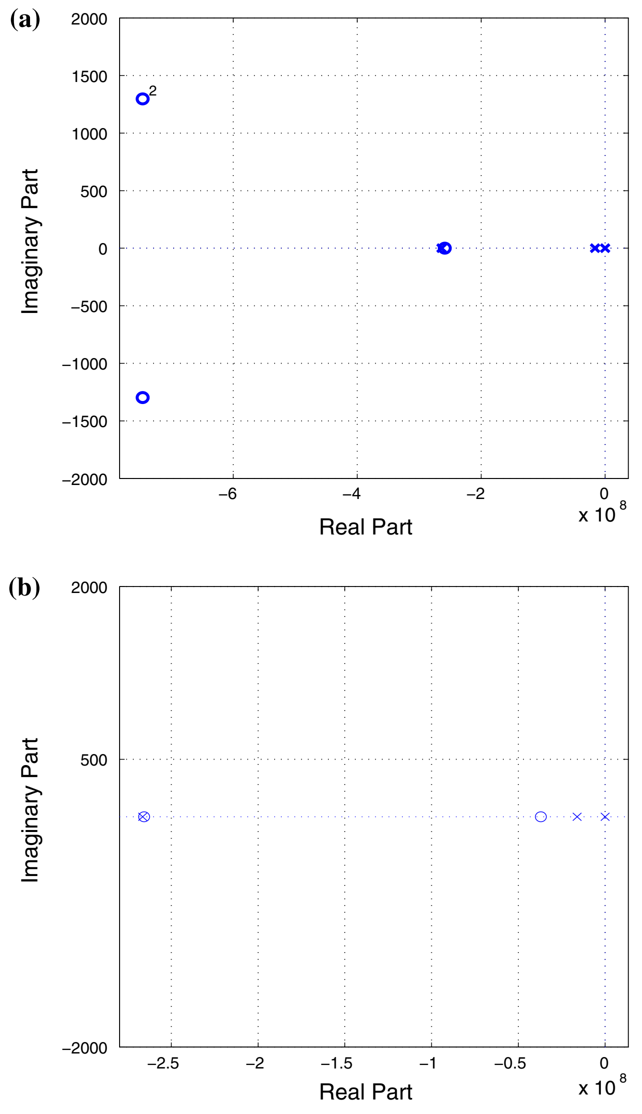

Fig. 13 Dominant poles and zeros by HSPICE: (a) Locations of pole and zero before R_z optimization. (b) Locations of pole and zero after R_z optimization

Table 2 Optimization process of resistor R_z

Iter. #	$R_z(k\Omega)$	$\angle H(s)(^\circ)$	$Sens(H(s), R_z)$	ΔR_z (Ω)
1	1.00	−155.6	−0.0661	3,460
2	4.46	−125.0	−0.2431	739
3	5.20	−120.5	−0.2347	95.5
4	5.29	−120.0		

into a popular industrial tool like SPICE. Whether or not this situation is going to change remains uncertain.

The most recently developed symbolic techniques reviewed in this survey might have shed some light. A design automation tool for analog integrated circuits should in many ways be different from a synthesis tool for digital integrated circuits. The reason is simple; an analog designer has to deal with too many design targets, some are purely time-domain and nonlinear, not easily tractable by a symbolic approach, while some are in the frequency-domain, tractable by the advanced symbolic methods studied in this survey. A reasonable analog design automation tool should emphasize the feature of *interaction* rather than full automation. Here, by *interaction* we mean that the designer can play around with a circuit to gain a full understanding of the circuit behavior. Actions like parameter tuning or device sizing can be traced visually for the effects on performance. Optimization is not by minimizing some mathematical objective functions subject to idealized constraints, rather is based on the designer's judgement with the help of visible performance plots, including the sensitivity plots. Until such a design assistance tool becomes available, the CAD tool developers must work closely with analog circuit designers and maximally exploit the potential of advanced data structures or computation strategies to realize the tool features that are not available in the conventional design tools.

The symbolic analysis techniques based on BDDs have opened the door to many possibilities, although deeper and broader research work remains to be done. The symbolic sensitivity is a promising notion worth further investigation and exploration. It is believed that, as more knowledge is gained on the advanced symbolic analysis methods, the art of analog circuit design would become a more enjoyable art in the near future.

Acknowledgment This research was supported by the National Natural Science Foundation of China (Grant No. 60876089).

References

1. Brace, K. S., Rudell, R. L., Bryant, R. E. (1990). Efficient implementation of a BDD package. In: *Proceedings of the 27th ACM/IEEE Design Automation Conference* (pp. 40–45). Orlando, FL.
2. Bryant, R. E. (1986). Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers, C-35*(8), 677–691.
3. Bryant, R. E., & Kukula, J. H. (2003). Formal methods for functional verification. In A. Kuehlmann (ed.), *The best of ICCAD, 20 years of excellence in computer-aided design* (pp. 3–15). Norwell, MA: Kluwer Academic Publishers.
4. Chen, W., & Shi, G. (Dec. 2006). Implementation of a symbolic circuit simulator for topological network analysis. In *Proceedings of the Asia Pacific conference on circuits and systems (APCCAS)* (pp. 1327–1331). Singapore.
5. Doboli, A., & Vemuri, R. (2001). A regularity-based hierarchical symbolic analysis methods for large-scale analog networks. *IEEE Transactions on Circuits and Systems—II: Analog and Digital Signal Processing, CAS-48*(11), 1054–1068.
6. Gielen, G., & Sansen, W. (1991). *Symbolic analysis for automated design of analog integrated circuits*. Norwell, MA: Kluwer Academic Publishers.

7. Gielen, G., Wambacq, P., & Sansen, W. M. (1994). Symbolic analysis methods and applications for analog circuits: A tutorial overview. *Proceedings of the IEEE, 82*(2), 287–303.
8. Gray, P. R., & Meyer, R. G. (1982). MOS operational amplifier design—A tutorial overview. *IEEE Journal of Solid-State Circuits, SC-17*(6), 969–982.
9. Guerra, O., Roca, E., Fernández, F. V., & Rodríguez-Vázquez, A. (2002). Approximate symbolic analysis of hierarchically decomposed analog circuits. *Analog Integrated Circuits and Signal Processing, 31*, 131–145.
10. Guerra, O., Rodríguez-García, J. D., Fernández, F. V., & Rodríguez-Vázquez, A. (2002). A symbolic pole/zero extraction methodology based on analysis of circuit time-constants. *Analog Integrated Circuits and Signal Processing, 31*, 101–118.
11. Hao, Z., Tan, S. X. D., Shen, R., & Shi, G. (June. 2011). Performance bound analysis of analog circuits considering process variations. In *Proceedings of the IEEE/ACM design automation conference (DAC)*. CA, USA. (accepted for publication).
12. Ho, C. W., Ruehli, A. E., & Brennan, P. A. (1975). The modified nodal approach to network analysis. *IEEE Transactions on Circuits and Systems, CAS-22*(6), 504–509.
13. Ho, K. P., Chan, C. F., Choy, C. S., & Pun, K. P. (2003) Reversed nested Miller compensation with voltage buffer and nulling resistor. *IEEE Journal of Solid-State Circuits, SC-38*(10), 1735–1738.
14. Lee, J. Y., Huang, X., & Rohrer, R. A. (1992). Pole and zero sensitivity calculation in asymptotic waveform evaluation. *IEEE Transactions on Computer-Aided Design, 11*(5), 586–597.
15. Li, X., Xu, H., Shi, G., & Tai, A. (May 2011). Hierarchical symbolic sensitivity computation with applications to large amplifier circuit design. In *Proceedings of the international conference on circuits and systems (ISCAS)* (pp. 2733–2736). Rio de Janeiro, Brazil.
16. Lin, P. M. (1991). *Symbolic network analysis*. New York: Elsevier.
17. Ma, D., Shi, G., & Lee, A. (Dec. 2010). A design platform for analog device size sensitivity analysis and visualization. In *Proceedings of the Asia Pacific conference on circuits and systems (APCCAS)* (pp. 48–51). Malaysia.
18. Minato, S. (1993). Zero-suppressed BDD's for set manipulation in combinatorial problems. In *Proceedings of the 30th IEEE/ACM design automation conference* (pp. 272–277). Dallas, TX.
19. Minato, S. (1996). *Binary decision diagrams and applications for VLSI CAD*. Norwell, MA: Kluwer Academic.
20. Minty, G. J. (1965). A simple algorithm for listing all the trees of a graph. *IEEE Transactions on Circuit Theory, CT-12*, 120.
21. Nebel, G., Kleine, U., & Pfleiderer, H. J. (1995). Symbolic pole/zero calculation using SANTAFE. *IEEE Journal of Solid-State Circuits, 30*(7), 752–761.
22. Shi, C. J. R., & Tan, X. D. (2000). Canonical symbolic analysis of large analog circuits with determinant decision diagrams. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 19*(1), 1–18.
23. Shi, C. J. R., & Tan, X. D. (2001). Compact representation and efficient generation of s-expanded symbolic network for computer-aided analog circuit design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 20*(7), 813–827.
24. Shi, G. (2010). Computational complexity analysis of determinant decision diagram. *IEEE Transactions on Circuits and Systems—II: Express Briefs, 57*(10), 828–832.
25. Shi, G. (2010). A simple implementation of determinant decision diagram. In *Proceedings of the International Conference on Computer-Aided Design (ICCAD)* (pp. 70–76). San Jose, CA, USA.
26. Shi, G., Chen, W., & Shi, C. J. R. (Jan. 2007). A graph reduction approach to symbolic circuit analysis. In *Proceedings of the Asia South-Pacific design automation conference (ASPDAC)* (pp. 197–202). Yokohama, Japan.
27. Shi, G., & Meng, X. (May 2009). Variational analog integrated circuit design by symbolic sensitivity analysis. In: *Proceedings of the international symposium on circuits and systems (ISCAS)* (pp. 3002–3005). Taiwan, China.
28. Tan, S. X. D. (2005). A general hierarchical circuit modeling and simulation algorithm. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 24*(3), 418–434.
29. Tan, S. X. D. (2006). Symbolic analysis of analog integrated circuits by boolean logic operations. *IEEE Transactions on Circuits and Systems—II: Express Briefs, 53*(11), 1313–1317.
30. Tan, S.X.D., Guo, W., & Qi, Z. (2004). Hierarchical approach to exact symbolic analysis of large analog circuits. In *Proceedings of the design automation conference* (pp. 860–863).
31. Tan, S. X. D., & Shi, C. J. R. (2004). Efficient approximation of symbolic expressions for analog behavioral modeling and analysis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 23*(6), 907–918.
32. Tan, X. D., & Shi, C. J. R. (2000). Hierarchical symbolic analysis of analog integrated circuits via determinant decision diagrams. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 19*(4), 401–412.
33. Verhaegen, W., & Gielen, G. E. (2002). Efficient DDD-based symbolic analysis of linear analog circuits. *IEEE Transactions on Circuits and Systems—II: Analog and Digital Signal Processing, 49*(7), 474–487.
34. Vlach, J., & Singhal, K. (1983). *Computer methods for circuit analysis and design*. New York: Van Nostrand Reinhold Company.
35. Xu, H., Shi, G., & Li, X. (Jan. 2011). Hierarchical exact symbolic analysis of large analog integrated circuits by symbolic stamps. In *Proceedings of the Asia South-Pacific design automation conference (ASPDAC)* (pp. 19–24). Yokohama, Japan.
36. Yang, H., Agarwal, A., & Vemuri, R. (2005). Fast analog circuit synthesis using multi-parameter sensitivity analysis based on element-coefficient diagrams. In *Proceedings of the IEEE computer society annual symposium on VLSI* (pp. 71–76). Tampa, Florida, USA.
37. Yang, H., Ranjan, M., Verhaegen, W., Ding, M., Vemuri, R., & Gielen, G. (Jan. 2005). Efficient symbolic sensitivity analysis of analog circuits using element-coefficient diagrams. In *Proceedings of the Asia South-Pacific design automation conference (ASPDAC)* (pp. 230–235). Yokohama, Japan.
38. Yu, Q., & Sechen, C. (1996). A unified approach to the approximate symbolic analysis of large analog integrated circuits. *IEEE Transactions on Circuits and Systems—I: Fundamental Theory and Applications,43*(8), 656–669.

Guoyong Shi is currently a professor of School of Microelectronics, Shanghai Jiao Tong University in China. Before joining the university, he was a post-doctoral research fellow at the Department of Electrical Engineering, University of Washington in Seattle. He received his Ph.D. degree in Electrical Engineering from Washington State University in Pullman. He was co-recipient of the 2007 Donald Pederson Best Paper Award. Dr. Shi is currently interested in the research on mixed-signal design automation tools.