Linear Decision Diagrams[*]

Eugene W. Stark, Wenxin Song
Department of Computer Science
State University of New York at Stony Brook
Stony Brook, NY 11794 USA
{stark,wenxin}@cs.sunysb.edu

Abstract

We introduce linear decision diagrams *(LDDs) as a special class of linear representations of formal power series. LDDs can be seen as a generalization of some previously proposed structures, such as MTBDDs and matrix diagrams, that have seen successful application in the compact representation of Markov models with large state spaces. Besides providing some possibilities for additional compression, LDDs have an interesting and useful reversibility property that is not shared by previously considered BDD variants. In addition, LDDs have the advantage that most LDD operations can be performed without any assumption that the arguments are fully reduced or "canonical." This suggests the possibility of using multiple reduction heuristics that trade off reduction "strength" for computation cost. We present some experimental results that compare the sizes of MTBDD and LDD representations for rate matrices obtained from some standard benchmark examples.*

1. Introduction

Binary decision diagrams (BDDs) [1, 4, 21] and their variants are a class of data structures that has seen successful application in the formal verification of systems with large state spaces. Variations of BDDs have been proposed to support quantitative calculations of the type that are required for verification and performance analysis of systems modeled using Markov chains. For example, *multi-terminal binary decision diagrams* (MTBDDs) [2, 11, 12] are a generalization of BDDs in which there can be multiple leaf nodes, each labeled by a distinct value drawn from a set of $\mathcal{V}$ of possible values (for example, the real numbers). An n-level MTBDD thus represents a $\mathcal{V}$-valued function of n boolean arguments, or equivalently, a 2^n-dimensional vector with entries drawn from $\mathcal{V}$. An MTBDD representation of a vector can be very compact, assuming that the set of distinct values appearing as entries in the vector is small. MTBDDs can also be used to represent matrices, and computations such as vector/matrix multiplication can be performed efficiently in terms of MTBDD representations. A variety of other variations of BDDs have been proposed, including *edge-valued* and *factored edge-valued* BDDs [20, 29], *binary moment diagrams* (BMDs) [5], *multi-valued decision diagrams* (MDDs) [17], and *matrix diagrams* (MDs) [8].

This paper proposes *linear decision diagrams* (LDDs) as a data structure for formal verification. LDDs are a special case of the *linear representations* that have been studied in the context of the theory of formal power series (see, *e.g.* [3, 13, 24]). Like MTBDDs, linear representations describe mappings from finite strings of symbols (*e.g.* from the two-element alphabet $\{0, 1\}$) to values. LDDs are a class of linear representations that exhibit a "level structure" like that of BDDs. In fact, the so-called "quasi-reduced" MTBDDs [22] can be regarded as a restricted class of linear representations. When LDDs are used to represent matrices there is also a close relationship to MDs. However, the additional structure of LDDs provides some possibilities for additional compression beyond that possible with previous BDD variants. LDDs also have an interesting and useful reversibility property that BDDs do not have. In addition, LDDs have the advantage that most LDD operations can be performed without any assumption that the arguments are fully reduced or "canonical." This suggests the possibility of using multiple reduction heuristics that trade off reduction "strength" for computation cost.

The remainder of this paper is organized as follows. In Section 2 we give a general definition of linear representations and briefly indicate how various operations can be performed with them. A key point here is the fact that there

[*] This research was supported in part by the National Science Foundation under Grant CCR-9988155 and the Army Research Office under Grants DAAD190110003 and DAAD190110019. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation, the Army Research Office, or other sponsors.

exists a *minimization algorithm* ([25]) that can be applied to minimize the number of dimensions (nodes) in a linear representation so as to avoid the blow-up that would otherwise occur as successive operations are performed. In Section 3 we introduce linear decision diagrams (LDDs) as a restricted class of linear representations. We indicate how LDDs can be used to perform vector and matrix computations, and discuss the relationship between LDDs, MTBDDs, and MDs. In Section 3.5 we discuss some issues related to the implementation of LDDs. In Section 4 we report the results of some experiments in which we compare the sizes of LDD matrix representations under various reduction "strengths." Finally, in Section 5 we draw some brief conclusions from our investigation of LDDs.

2. Linear Representations

Linear representations are a kind of vector automata that compute mappings of finite words over an alphabet Σ to values in a suitable space $\mathcal{V}$. In a classical setting, such mappings are called *formal power series*, in view of the observation that a power series in several non-commuting variables can be usefully described as a mapping that takes each finite sequence of variables (*i.e.* a "monomial") to its associated coefficient. In the most general setting considered in the classical theory, it is required only that the space $\mathcal{V}$ be a *semiring*, which is a structure having an addition and a (not necessarily commutative) multiplication operation with distinguished elements playing the role of 0 and 1, but in which neither multiplicative nor additive inverses need exist. In this paper, though, we restrict our attention to the special case in which $\mathcal{V}$ is a field; for example, the field of rational numbers or the field of real numbers.

Formally, if $\mathcal{V}$ is a field, then we use $\mathcal{V}^{1\times d}$, $\mathcal{V}^{d\times 1}$, and $\mathcal{V}^{d\times d}$ to denote, respectively, the set of all d-dimensional row vectors, the set of all d-dimensional column vectors, and the set of all $d \times d$-matrices, with entries in $\mathcal{V}$. Vector and matrix addition and multiplication are defined in the usual way. It is convenient to identify the set $\mathcal{V}^{1\times 1}$ of 1×1-matrices with $\mathcal{V}$ itself.

A function $S : \Sigma^* \to \mathcal{V}$ is called a *formal series over* Σ *with coefficients in* $\mathcal{V}$. The notation $\mathcal{V}\langle\langle\Sigma\rangle\rangle$ is traditionally used to denote the set of all such formal series. The value of S on a word w is traditionally denoted by (S, w). A series $S \in \mathcal{V}\langle\langle\Sigma\rangle\rangle$ is called *recognizable* if there exists

- an integer $d \geq 0$,
- a row vector $C \in \mathcal{V}^{1\times d}$,
- a column vector $D \in \mathcal{V}^{d\times 1}$, and
- a monoid homomorphism $M : \Sigma^* \to \mathcal{V}^{d\times d}$,

such that for all $w \in \Sigma^*$ we have

$$(S, w) = C(Mw)D.$$

The triple $R = (C, M, D)$ is called a *linear representation* of S and d is the *dimension* of the linear representation. In the degenerate case $d = 0$, the spaces $\mathcal{V}^{1\times d}$, $\mathcal{V}^{d\times 1}$ and $\mathcal{V}^{d\times d}$ each contain only the zero vector, and the representation recognizes the identically zero formal series. We will say that two linear representations R and R' over Σ and $\mathcal{V}$ are *equivalent* if they recognize the same formal series.

An interesting property of linear representations is that every linear representation $R = (C, M, D)$ has a *dual* $R^{\mathrm{op}} = (D^T, M^T, C^T)$. It is easy to see that if R recognizes formal series S, then its dual R^{op} recognizes the formal series S^{op} defined by $(S^{\mathrm{op}}, w) = (S, w^R)$, where w^R denotes the *reversal* of the word w.

2.1. Minimization

An important fact about linear representations is the following, which requires that $\mathcal{V}$ be a field.

Proposition 1 (Schützenberger [25]) *If $S \in \mathcal{V}\langle\langle\Sigma\rangle\rangle$ is a recognizable formal series, then there exists a linear representation $R_{\min}$ that is* minimal *in the sense that it has minimum dimension among all representations that recognize S. Moreover, there exists an algorithm for computing a minimal linear representation of S given an arbitrary linear representation of S as input.*

We do not give here a complete proof of Proposition 1. For that, the reader may refer to the original paper of Schützenberger [25], or to the book [3] which has a more algebraic presentation. For our purposes, though, it will be helpful to consider in concrete terms the problem of how, given an arbitrary representation R, one can compute a minimal representation R' equivalent to R. Essentially the same minimization algorithm as we describe here has been discussed previously in ([25, 7, 3]). Our presentation here centers around notion of a "reduction" on a representation R, and the related concepts of "reachability" and "observability." These concepts will be helpful later in understanding how the minimization algorithm specializes to LDDs.

Formally, suppose $R = (C, M, D)$ is a representation of dimension d. We call R *reachable* if the set $\{C(Mu) : u \in \Sigma^*\}$ spans $\mathcal{V}^{1\times d}$. Similarly, we call R *observable* if the set $\{(Mv)D : v \in \Sigma^*\}$ spans $\mathcal{V}^{d\times 1}$. If R is both reachable and observable we call it *canonical*. The following result is obvious, but useful to note.

Proposition 2 *Let R be a representation. Then R is observable if and only if R^{op} is reachable.*

The next result shows why reachability and observability are of interest with respect to minimization.

Proposition 3 *Let R be a representation. Then R is canonical if and only if it is minimal.*

A *reduction* on a representation $R = (C, M, D)$ is a pair of matrices (P, Q), where $P \in \mathcal{V}^{d \times d'}$ and $Q \in \mathcal{V}^{d' \times d}$, such that the following conditions are satisfied:

1. $QP = I_{d'}$
2. $CPQ = C$
3. $Q(M\sigma)PQ = Q(M\sigma)$ for all $\sigma \in \Sigma$

Note that condition (1) implies that $d' \leq d$. If in fact $d' < d$ then the reduction is called *nontrivial*.

The notions of reachability and reduction are connected as follows:

Proposition 4 *A representation R is reachable if and only if there is no nontrivial reduction on R.*

Suppose $R = (C, M, D)$ is a representation, and let (P, Q) be a reduction on R. Let representation $R' = (C', M', D')$ be defined as follows:

1. $C' = CP$.
2. $D' = QD$.
3. $M'\sigma = Q(M\sigma)P$ for all $\sigma \in \Sigma$.

We call R' the *reduct* of R by the reduction (P, Q).

Proposition 5 *Suppose $R = (C, M, D)$ is a representation, let (P, Q) be a reduction on R with $P \in \mathcal{V}^{d \times d'}$ and $Q \in \mathcal{V}^{d' \times d}$, and let $R' = (C', M', D')$ be the reduct of R by (P, Q), Then R' is equivalent to R. Moreover, if R is observable then so is R'.*

Proposition 6 *Suppose $R = (C, M, D)$ is a representation of dimension d. Let $\mathcal{S}$ be any linear subspace of $\mathcal{V}^{1 \times d}$ that includes C and is closed under $M\sigma$ for all $\sigma \in \Sigma$. Then there is a reduction (P, Q) on R such that $\mathcal{S}$ is the image of the idempotent PQ.*

From the above results we can see how to minimize a representation. Given a representation $R = (C, M, D)$ of dimension d, compute the least subspace $\mathcal{S}$ of R that contains C and is closed under $M\sigma$ for all $\sigma \in \Sigma$. Associated with the subspace $\mathcal{S}$ is a reduction (P, Q), such that $\mathcal{S}$ is the image of the idempotent PQ. Let R' be the reduct of R by the reduction (P, Q), then R' is reachable and equivalent to R. Now repeat the procedure on the dual $(R')^{\mathrm{op}}$ of R' to obtain R'', equivalent to $(R')^{\mathrm{op}}$, which is both reachable and observable and hence minimal. The dual $(R'')^{\mathrm{op}}$ of R'' is then a minimal representation that is equivalent to the original representation R.

2.2. Computing with Linear Representations

Since there is just one linear representation of dimension zero, it follows from the results of the preceding section that there is an algorithm to determine whether a given linear representation R recognizes the identically zero series: simply minimize R and then check whether the resulting representation has dimension zero. In addition, algorithms exist for computing a variety of other operations on linear representations, including:

(Scaling) Given a value $a \in \mathcal{V}$ and a representation R that recognizes formal series S, compute a representation aR that recognizes the series aS defined by $(aS, w) = a(S, w)$.

(Addition) Given representations R_1 and R_2 that recognize formal series S_1 and S_2, respectively, compute a representation $R_1 + R_2$ that recognizes the series $S_1 + S_2$ defined by $(S_1 + S_2, w) = (S_1, w) + (S_2, w)$.

(Hadamard Product) Given representations R_1 and R_2 that recognize formal series S_1 and S_2, respectively, compute a representation $R_1 * R_2$ that recognizes the series $S_1 * S_2$ defined by $(S_1 * S_2, w) = (S_1, w) \cdot (S_2, w)$.

(Cauchy Product) Given representations R_1 and R_2 that recognize formal series S_1 and S_2, respectively, compute a representation $R_1 \cdot R_2$ that recognizes the series $S_1 \cdot S_2$ defined by

$$(S_1 \cdot S_2, w) = \sum_{uv=w} (S_1, u) \cdot (S_2, v).$$

(Equality Test) Given representations R_1 and R_2, determine whether R_1 and R_2 recognize the same formal series.

Though straightforward constructions to implement most of the above operations would be dimension-increasing, by following each construction with an application of minimization one can perform a series of operations without danger of the blow-up in dimension that would otherwise occur.

3. Linear Decision Diagrams

A linear representation $R = (C, D, M)$ over Σ is called a *linear decision diagram* (LDD) if there exists a natural number n and a sequence $d_0, d_1, \ldots, d_n$ of natural numbers, such that for each $\sigma \in \Sigma$ the matrix $M\sigma$ has the block form:

$$\begin{pmatrix} 0 & M_{1,\sigma} & 0 & \ldots & 0 \\ 0 & 0 & M_{2,\sigma} & \ldots & 0 \\ \ldots & & & & \\ 0 & 0 & 0 & \ldots & M_{n,\sigma} \\ 0 & 0 & 0 & \ldots & 0 \end{pmatrix}$$

where $M_{k,\sigma} \in \mathcal{V}^{d_{k-1} \times d_k}$ for $1 \leq k \leq n$, and such that C and D have the block forms:

$$C = \begin{pmatrix} C_0 & 0 & 0 & \ldots & 0 \end{pmatrix} \qquad D = \begin{pmatrix} 0 \\ 0 \\ 0 \\ \ldots \\ D_n \end{pmatrix}$$

where $C_0 \in \mathcal{V}^{1 \times d_0}$ and $D_n \in \mathcal{V}^{d_n \times 1}$.

Alternatively, since the sequence $d_0, d_1, \ldots, d_n$ is implicit in the dimensions of the matrices $M_{k,\sigma}$ $(1 \leq k \leq n)$, we can specify an LDD simply by giving these matrices, together with the nonzero portion C_0 of the input vector C and the nonzero portion D_n of the output vector D. That is, we can regard an LDD as a tuple:

$$(C_0, \{M_{k,\sigma} : 1 \leq k \leq n, \sigma \in \Sigma\}, D_n).$$

such that the evident relationship between the dimensions holds. In the sequel, it will be convenient for us to pass back and forth implicitly between these two ways of regarding an LDD. Note that, due to the special form of an LDD L, if S is the formal series recognized by L, then S is a polynomial whose (finite) support is contained in the set Σ^n. Note also that, like a BDD, an LDD has a "level structure," where the dimensions d_0, d_1, …, d_n correspond to the number of nodes at each of the $n+1$ levels, and the matrices $M_{k,\sigma}$ define the edges between the levels.

3.1. Minimization

As they are a special case of linear representations, the results of Section 2.1 apply to LDDs, yielding the existence of LDDs of minimal dimension and of an LDD minimization algorithm. In fact, the special structure of LDDs can be exploited to obtain the stronger result that minimal LDDs are in fact minimal in *each* of the dimensions d_n, and that minimization of an LDD can be performed in two phases: a forward, "reachability" phase that sweeps through the list of matrices from left to right, and a backward, "observability" phase that sweeps from right to left.

LDD Reachability Algorithm

Input: An LDD $L = (C, \{M_{k,\sigma} : 1 \leq k \leq n, \sigma \in \Sigma\}, D)$.

Output: A reachable LDD

$$\mathcal{R}(L) = (C', \{M'_{k,\sigma} : 1 \leq k \leq n, \sigma \in \Sigma\}, D')$$

that is equivalent to L.

Procedure: If C is the identically zero vector, then output the degenerate LDD having $d_k = 0$ for $0 \leq k \leq n$.

Otherwise, suppose C has a nonzero value c_i in the ith position, for some i with $1 \leq i \leq d_0$. Define $Q_0 \in \mathcal{V}^{1 \times d_0}$ to be the matrix having C as its single row. Define $P_0 \in \mathcal{V}^{d_0 \times 1}$ to be the matrix having as its sole nonzero entry $1/c_i$ in the ith row. Observe that $Q_0 P_0 = I_1$ and that $CP_0Q_0 = C$.

Now, proceeding iteratively for each k from 1 to n, suppose at stage k that we have computed matrix Q_{k-1}. Let Q_k be a matrix whose rows form an independent subset of the set of all rows of the products $Q_{k-1}M_{k,\sigma}$. Let P_k be a matrix such that $Q_k P_k = I_{d_k}$. Although P_k is in general not uniquely determined, the correctness of the algorithm does not depend on how P_k is selected. Note that, regardless of how P_k is obtained, the construction of Q_k and the property $Q_k P_k = I_{d_k}$ implies that $P_k Q_k$ is an idempotent matrix whose image includes each of the rows of the matrices $Q_{k-1}M_{k,\sigma}$, and hence we have

$$Q_{k-1}M_{k,\sigma}P_kQ_k = Q_{k-1}M_{k,\sigma}$$

for all $\sigma \in \Sigma$.

Once the above procedure has been carried out for $1 \leq k \leq n$, we have constructed matrices P_k and Q_k for $0 \leq k \leq n$ satisfying the following properties:

1. $Q_k P_k = I_{d_k}$, for $0 \leq k \leq n$.
2. $CP_0Q_0 = C$.
3. $Q_{k-1}M_{k,\sigma}P_kQ_k = Q_{k-1}M_{k,\sigma}$ for all $\sigma \in \Sigma$ and $1 \leq k \leq n$.

These properties are easily seen to be the appropriate specialization to LDDs of the notion of a reduction from Section 2.1. Let the LDD

$$\mathcal{R}(L) = (C', \{M'_{k,\sigma} : 1 \leq k \leq n, \sigma \in \Sigma\}, D')$$

be defined as follows:

- $C' = CP_0$.
- $D' = Q_n D$.
- $M'_{k,\sigma} = Q_{k-1}M_{k,\sigma}P_k$ for $1 \leq k \leq n$.

Once again, this definition specializes to the case of LDDs the notion of reduct defined in Section 2.1. It then follows that $\mathcal{R}(L)$ is a reachable LDD that is equivalent to L. ▮

Full minimization of an LDD is achieved using two applications of the LDD reachability algorithm presented above:

LDD Minimization Algorithm

Input: An LDD $L = (C, \{M_{k,\sigma} : 1 \leq k \leq n, \sigma \in \Sigma\}, D)$.

Output: A canonical LDD

$$\mathcal{C}(L) = (C', \{M'_{k,\sigma} : 1 \leq k \leq n, \sigma \in \Sigma\}, D')$$

that is equivalent to L.

Procedure: Given L, apply the reachability algorithm to L to obtain a reachable LDD $\mathcal{R}(L)$ that is equivalent to L.

Then, compute the dual $\mathcal{R}(L)^{\mathrm{op}}$ of $\mathcal{R}(L)$, apply the reachability algorithm again to obtain $\mathcal{R}(\mathcal{R}(L)^{\mathrm{op}})$, and finally dualize again to obtain

$$\mathcal{C}(L) = \mathcal{R}(\mathcal{R}(L)^{\mathrm{op}})^{\mathrm{op}}$$

which is a canonical LDD equivalent to L. ∎

3.2. Computing with LDDs

The constructions, mentioned in Section 2.2, for computing various operations on linear representations obviously specialize to LDDs. Often, the special structure of LDDs makes it more efficient to compute certain operations on LDDs than on general linear representations. One case in point is the minimization algorithm itself. The reachability algorithm for a linear representation of dimension d would in general involve the calculation of a basis for the least subspace of $\mathcal{V}^{1\times d}$ that contains the vector C and is closed under multiplication on the right by the matrices $M\sigma$. This closure calculation can be performed via repeated multiplication by M to generate vectors of dimension d, which then have to be checked for independence with respect to the span of the set of previously generated vectors. Although each step in the LDD reachability algorithm also requires a matrix multiplication and the extraction of an independent set of rows, the vectors at the kth stage are only of dimension d_k, rather than the full dimension $d = \sum_{k=0}^{n} d_k$. In addition, for an LDD with $n+1$ levels closure is reached after only n stages, whereas a general linear representation could require up to d iterations to reach closure.

Another calculation that can be performed more efficiently on LDDs is the following:

(Summation) Given an LDD L that recognizes formal series S, compute the value $\sum_{w\in\Sigma^*}(S,w)$.

For an LDD this can be done simply by forming the product:

$$C\left(\prod_{k=1}^{n}\sum_{\sigma\in\Sigma} M_{k,\sigma}\right)D.$$

For an arbitrary linear representation the corresponding calculation would be:

$$C\left(\sum_{\sigma\in\Sigma} M\sigma\right)^{*} D,$$

where $*$ denotes Kleene star.

We now give explicit descriptions of several other useful constructions that can be performed on LDDs. In the following, suppose

$$L = (C, \{M_{k,\sigma} : 1 \le k \le d, \sigma\in\Sigma\}, D)$$

and

$$L' = (C', \{M'_{k,\sigma} : 1 \le k \le d', \sigma\in\Sigma\}, D')$$

are given LDDs.

(Scaling) Define aL to be the LDD

$$aL = (aC, \{M_{k,\sigma} : 1 \le k \le d, \sigma\in\Sigma\}, D).$$

(Addition) Assuming $d = d'$, define $L + L'$ to be the LDD

$$L + L' = (C'', \{M''_{k,\sigma} : 1 \le k \le d, \sigma\in\Sigma\}, D'')$$

where

$$C'' = \begin{pmatrix} C & C' \end{pmatrix} \qquad D'' = \begin{pmatrix} D \\ D' \end{pmatrix}$$

$$M''_{k,\sigma} = \begin{pmatrix} M_{k,\sigma} & 0 \\ 0 & M'_{k,\sigma} \end{pmatrix}$$

(Hadamard Product) Assuming $d = d'$, define $L * L'$ to be the LDD

$$L * L' = (C'', \{M''_{k,\sigma} : 1 \le k \le d, \sigma\in\Sigma\}, D'')$$

where

$$C'' = C \otimes C' \qquad D'' = D \otimes D'$$

$$M''_{k,\sigma} = M_{k,\sigma} \otimes M'_{k,\sigma}$$

and $\otimes$ denotes the Kronecker (tensor) product.

(Cauchy Product) Suppose S and S' are the formal series represented by L and L', so that the support of S lies in Σ^d and the support of S' lies in $\Sigma^{d'}$. Then in this special case the Cauchy product of S and S' is the formal series SS' defined by

$$(SS', w) = \begin{cases} (S,u)(S',u'), & \text{if } w = uu', \\ & u\in\Sigma^d, u'\in\Sigma^{d'} \\ 0, & \text{otherwise.} \end{cases}$$

Define the LDD $L \cdot L'$ by

$$L \cdot L' = (C, \{M''_{k,\sigma} : 1 \le k \le d + d'\}, D')$$

where

$$M''_{k,\sigma} = \begin{cases} M_{k,\sigma}, & \text{for } 1 \le k \le d \\ (D\otimes C')M'_{1,\sigma}, & \text{for } k = d+1 \\ M'_{k-d,\sigma}, & \text{for } d+2 \le k \le d+d' \end{cases}$$

3.3. Vector and Matrix Computation

Since LDDs represent mappings from finite words to values, they can be used to represent vectors and matrices in a fashion similar to the way MTBDDs are used for this purpose. In this section, we briefly sketch the main ideas and indicate some ways in which LDDs differ from MTBDDs. We suppose throughout this section that Σ is the two-letter alphabet $\{0, 1\}$.

An $n + 1$-level LDD over Σ represents a formal power series S over Σ, whose support is contained in Σ^n. By fixing an encoding of elements of the set $\{0, 1, \dots, 2^n - 1\}$ as words in Σ^n, we may regard S as a 2^n-element vector with entries from $\mathcal{V}$. For example, we may choose an encoding in which $i \in \{0, 1, \dots, 2^n - 1\}$ is represented by its binary encoding given least-significant bit first. MTBDDs are used to represent vectors in the same way. However, note that if LDD L represents a vector under a least-significant-bit-first encoding, then its dual L^{op} represents the same vector under a most-significant-bit-first encoding. MTBDDs do not have this duality property.

Consideration of the definitions of reachability and observability for LDDs reveals that for a canonical LDD over Σ, the dimension d_k at the kth level can be at most 2^k for $0 \le k \le n/2$ and at most 2^{n-k} for $n/2 \le k \le n$. Minimal LDDs representing sparse vectors will in general have dimensions much smaller than these exponential bounds, so that LDDs can provide a compact sparse vector representation. This compactness is not necessarily limited to vectors containing large numbers of zeroes, since like MTBDDs a minimal LDD representation also provides compression by identifying identical subvectors when these subvectors occur “aligned” at indices that are powers of 2. Unlike MTBDDs, but like FEVBDDs [20, 29] and matrix diagrams (MDs) [8], LDDs are also capable of identifying subvectors that are multiples of each other. However, LDDs provide the opportunity for even further compression by identifing subvectors that are linear combinations of each other.

Another way in which LDDs are different from MTBDDs is their duality property. A canonical LDD will always have $d_0 = d_n = 1$; that is, it will have a single root node and a single leaf node. Because of this, and because the number of nodes can increase or decrease by no more than a factor of two at each level, as we traverse the levels of an LDD from left to right the number of nodes at each successive level tends to increase toward the middle levels, then decrease again. This need not occur with MTBDDs, since the number of terminal nodes in a reduced MTBDD is not fixed but rather depends on the number of distinct nonzero values in the vector it represents.

Like MTBDDs, LDDs admit efficient implementations of basic operations on vectors. Scaling and addition of vectors can be done using the previously mentioned scaling and addition operations on LDDs. Componentwise product of vectors can be performed using the Hadamard product operation on LDDs (this is essentially the same as the “apply” operation on MTBDDs, in the special case that the operation being applied is multiplication). Dot product of vectors can be performed by applying Hadamard product followed by summation to the LDD representations. The Cauchy product operation on LDDs corresponds to the Kronecker (tensor) product operation on vectors in case the lengths of the vectors in question are powers of 2. This operation is useful because in formal verification, Kronecker product often arises as a basic operation by which components are combined to yield a composite system (*e.g.* [23, 6]).

Besides vectors, LDDs can be used to represent matrices. A square matrix of dimension 2^n can be represented as a $2n + 1$-level LDD. By choosing the encoding of the indices compatibly with that used for vectors, one can perform various vector/matrix operations on the LDD representations. As for MTBDDs, a convenient choice is to encode the row and column indices by their n-bit binary representations, which are then interleaved by alternating row and column bits starting least-signficant bit first to produce a word $w \in \Sigma^{2n}$. With such conventions in place, the vector of column sums of a matrix can be accomplished by reducing a $2n + 1$-level LDD L to a $n + 1$-level LDD $\overline{L}$ via the construction

$$\overline{M}_{k,0} = (M_{2k-1,0} + M_{2k-1,1})M_{2k,0}$$
$$\overline{M}_{k,1} = (M_{2k-1,0} + M_{2k-1,1})M_{2k,1}$$

for $1 \le k \le n$. Multiplying a matrix on the left by a row vector can be performed by first expanding the $n + 1$-level LDD representing the vector to a $2n + 1$-level LDD by inserting new levels consisting of identity matrices in between each of the original levels (this amounts to transposing the original row and replicating it in columns), then computing the Hadamard product of the resulting LDD by the LDD representing the matrix, and finally reducing the result back to a $n + 1$-level LDD by forming column sums. Matrix/matrix multiplication can be accomplished using the same basic idea.

3.4. MTBDDs and Deterministic LDDs

LDDs are closely related to MTBDDs. Both LDDs and MTBDDs represent vectors whose dimensions are powers of 2. For an MTBDD, the distinct values appearing in the vector are displayed as the labels of the terminal nodes. The entry at the ith position in the vector is obtained by starting at the root of the MTBDD and following edges to a leaf node, where the edge followed from a node at level k is determined by the kth bit in the binary expansion of the index i. Unlike LDDs, edges in fully reduced MTBDDs can “skip levels” in the sense that an edge need not always go from a node at level k to a node at level $k+1$. However, we can imagine introducing "dummy nodes" in an MTBDD, having the same 0 and 1-successor nodes, so that levels are never skipped. MTBDDs that otherwise satisfy the conditions for being reduced, except for the presence of dummy nodes, are called *quasi-reduced* [22]. The 0- and 1-labeled edges between levels k and $k+1$ in a quasi-reduced MTBDD can then be represented as zero/one matrices $M_{k,0}$ and $M_{k,1}$. The root node can be represented as the 1-dimensional row vector $C = (1)$, and the distinct values labeling the leaf nodes can be displayed in the form of as a d_n-dimensional column vector D. Thus, a quasi-reduced MTBDD corresponds to an LDD over the alphabet $\Sigma = \{0, 1\}$, having all zero/one entries except at the last level, and which in addition is "deterministic" in the sense that vector C and each matrix $M_{k,0}$ and $M_{k,1}$ has at most one nonzero entry in each row.

Conversely, from a given LDD, it is possible to construct an equivalent LDD that is deterministic and has zero/one entries at all levels except the last. This can be achieved by an algorithm that is similar to the LDD reachability algorithm presented previously, except that at each stage instead of identifying a linearly independent set of rows we extract a set of nonzero rows that is "representative" in the sense that each nonzero row appears exactly once in the selected set. If the originally given LDD is observable, then it can be shown that the LDD resulting from this determinization algorithm corresponds to a quasi-reduced MTBDD. Interestingly, by exploiting the duality properties of an LDD, we also obtain a "codeterminization" algorithm for LDDs. Whereas the determinization algorithm outputs an LDD having the distinct values displayed at the leaf nodes, the codeterminization algorithm produces an LDD having its values displayed at the root nodes. The codeterminization of an LDD can be used as an efficient way to decompose a vector into a sum of "level sets."

From the above correspondences between MTBDDs and LDDs, we can conclude that a vector of dimension 2^n can always be represented at least as compactly by an LDD as it can by a quasi-reduced MTBDD. Moreover, LDDs can provide compact representations of vectors that cannot be represented compactly by MTBDDs; in particular this is true for vectors that have many distinct entries, but where these entries occur in patterns that can be generated as linear combinations of a small number of independent representatives.

3.5. Implementation Issues

We have incorporated an experimental implementation of LDDs into our *PIOATool* analysis engine [26, 32] for probabilistic I/O automata (PIOA) [30, 31, 28, 27]. PIOATool is designed to operate on specifications of continuous-time Markov chains (CTMCs) expressed as the composition of a hierarchically structured system of component PIOA. Besides the specification of the Markov chain, PIOATool takes as input an "observable," which is the description of a particular performance measure to be computed. The two basic operations performed by PIOATool are (1) to "apply" a system behavior to an observable to obtain a new observable, and (2) to "evaluate" an observable to obtain a result. The main operation involved in application is computing the Kronecker products, of each of a set of small transition matrices for the component being applied, with corresponding large transition matrices for observable. Evaluation involves a reachability analysis phase, followed by a phase involving the solution of a large set of linear equations. Both phases can be accomplished by an iterative approach in which the central operation is vector/matrix multiplication.

We now mention some practical experience we gained from the exercise of integrating LDD-based vector and matrices into PIOATool. Perhaps the most important point is that it is hardly ever required, and often not desirable, that LDDs be fully dimension-minimized. This is to be contrasted with the case of MTBDDs, where the algorithms for operating on MTBDDs rely heavily on their arguments being in canonical form. For LDDs, the only typically encountered operation where it is absolutely essential to place an LDD in reduced form is when it is desired to test whether the LDD represents the identically zero vector. The rest of the time an LDD can be maintained according to various weaker requirements. For example, here are three possibilities that we have found useful in practice:

1. Ensure that the block matrices $(M_{k,0}\, M_{k,1})$ and $(M_{k,0}^T\, M_{k,1}^T)$ have no zero rows (this corresponds to ensuring there are no "unreachable" or "unobservable" nodes).
2. Ensure that the block matrices $(M_{k,0}\, M_{k,1})$ and $(M_{k,0}^T\, M_{k,1}^T)$ have full rank (this yields full minimality).
3. By choosing from among the rows of the matrices $M_{k,0}$ and $M_{k,1}$ a set that is "representative" in the sense that no two rows in the set are multiples of each other, and by applying a change of basis associated with the selected set, we can transform the matrices $M_{k,0}$ and $M_{k,1}$ so as to obtain at most one nonzero entry in each row. Apply this transformation at each level unless doing so would result in an increase, rather than a decrease, in the number of columns. Dually, transform the matrices $M_{k,0}$ and $M_{k,1}$ so as to obtain at most one nonzero entry in each column, unless doing so would result in an increase in the number of rows.

In the course of building the experimental LDD implementation, we observed that best performance was *not* achieved when full LDD minimization (case (2) above) was used routinely. Instead, it was often faster to use the weaker form of minimization represented by (3) above, except in situations where full minimization was required for correctness. One reason for this discrepancy seems to be that best performance is achieved when the number of *edges*, rather than the number of *nodes* (*i.e.* dimension) is minimal. Full minimization, represented by case (2), minimizes dimension but sometimes does so at the expense of producing denser matrices $M_{k,0}$ and $M_{k,1}$. In addition, the cost of the independence checking required for case (2) generally is at least quadratic in the number of nodes at a level, whereas hashing techniques can be used in case (3) to reduce the complexity to linear in many cases.

After minimization, the LDD operation that it is most critical to optimize is Hadamard product, which is the central operation to vector/matrix multiplication (this corresponds to the "apply" operation for MTBDDs). A straightforward implementation of the definition in terms of Kronecker product is much too expensive, as it first builds an LDD whose dimensions at each level are the product of the dimensions of the argument LDDs at the corresponding level, and it then immediately applies the minimization algorithm and throws away most of what has been constructed. To avoid this, our implementation performs a preanalysis that permits us to avoid building the full Kronecker products, but rather to construct only that portion that would be left after applying minimization condition (1) above. It is not clear to us that this is the best that can be done.

A feature of MTBDD implementations that is not shared by our LDD implementation is the node cache. In MTBDD implementations, a cache is used to keep track of MTBDD nodes that have already been constructed. This cache serves two purposes: (1) it is used in ensuring that only reduced MTBDDs are constructed; and (2) it speeds up constructions on MTBDDs by avoiding constructing the same subgraph multiple times. LDD operations can be programmed without the use of a cache, assuming instead that one of the variants of the minimization algorithm is applied periodically to avoid explosion in dimension. It is not clear to us at the moment whether the efficiency gains provided by the MTBDD node cache make MTBDDs inherently significantly faster than LDDs, or whether we can program LDD operations in such a way as to make use of an MTBDD-like node cache.

Other implementation issues for LDDs derive from precision requirements and the use of floating point. Our LDD implementation can use either floating point or exact rational arithmetic. When exact arithmetic is used, we know that an LDD constructed to represent a vector or matrix exactly represents that vector or matrix. When floating point is used, the possible accumulation of numerical errors means that this need not be the case. One consequence of this is that near-zero values can tend to accumulate in the LDD matrices, thereby making these matrices more dense and making manipulation of them more time-consuming. Also, the full LDD minimization algorithm requires the selection of independent subsets of the set of rows of the matrices $M_{k,\sigma}$. We have observed situations in which sets of rows turn out to be very nearly dependent. In such cases, the precision required to perform the minimization can grow very rapidly. In the case of rational arithmetic, huge fractions are generated, thereby slowing down the computation very substantially. In the case of floating point arithmetic, inaccurate results can be produced. To avoid such problems, we employ a heuristic that detects when nearly dependent sets of rows are encountered during minimization and unless full minimization is absolutely essential (such as when testing if an LDD represents an identically zero vector) we simply avoid performing any reduction in such cases. We need to better understand the numerical issues associated with LDDs and to take a less *ad hoc* approach to dealing with them.

4. Experimental Results

In order to get an idea of the relative compactness of LDDs and MTBDDs in practical situations, we took rate matrices arising from several case studies examined by the PRISM group [18, 19] and constructed LDD representations of these matrices using three different LDD minimization criteria. The particular PRISM case studies we used were the Cyclic Server Polling System [16], the Tandem Queueing Network [15], the Flexible Manufacturing System [10], the Kanban Manufacturing System [9] and the Workstation Cluster [14]. These models have been used regularly as benchmarks in the performance analysis literature.

The three LDD minimization criteria we used were:

1. Full minimization. The dimension (number of nodes) of the LDD is the minimum possible.
2. Partial minimization. The LDDs were minimized according to criterion (3) of Section 3.5.
3. Deterministic. Deterministic LDDs were produced that had zero/one entries at all levels except the last. As noted in Section 3.4, such LDDs are essentially the same as quasi-reduced MTBDDs.

The example matrices we used were obtained from PRISM using an export feature present in the current version of that tool. Unfortunately, we were unable to make a direct comparison with the MTBDD size information that is reported for these examples on the PRISM web site. This is because although PRISM does represent these matrices using a variant (offset-labeled MTBDDs) of quasi-reduced MTBDDs, the internal representation of the matrices in PRISM uses more dimensions, and hence more MTBDD levels, than the matrices obtained via the PRISM export feature. So, although the figures we report for deterministic LDDs should give an idea of the relative compactness of MTBDDs versus that of partially and fully minimized LDDs, the numbers cannot be compared directly to those published by the PRISM group for the same benchmark examples.

Figure 1 summarizes our experimental results. Each table gives the results for various instances of a particular benchmark example. The parameter N, whose exact meaning depends on the example, determines the size of the instance. For the polling system, N is the number of stations, for the tandem queueing network, N is the capacity of the queue, for the flexible manufacturing system and kanban manufacturing system, N refers to the number of tokens, and for the workstation cluster, N denotes the number of workstations. To avoid complicating the results with issues related to the accumulation of near-zero floating point values, we used exact rational arithmetic in constructing the LDDs. The columns under "model" give the number of states and the number of nonzero entries in the rate matrix. The columns under "fully minimized" and "partially minimized" give the number of nodes (*i.e.* the dimension) of the LDD representation of the rate matrix and the number of edges (*i.e.* the total number of nonzero values in the input vector, matrices and output vector) of the LDD. In the "deterministic" case, we also report under "terminal nodes" the dimension d_n of the final LDD level. This value equals the number of distinct nonzero entries in the matrix represented by the LDD.

In general, the experimental results show that fully minimized and even partially minimized LDDs often provide substantial reductions in the numbers of nodes and edges as compared to determinized LDDs (*i.e.* quasi-reduced MTBDDs). The reductions derive from the ability of LDDs to exploit patterns that are more general than just repeated instances of identical submatrices. Fully minimized LDDs, while providing additional reduction (in some cases significant amounts) in the number of nodes over partially reduced LDDs, typically do so at the cost of an increase in the number of edges. In addition, though the tables do not show timing information, the cost of fully minimizing LDDs grows at a rate greater than linear in the number of dimensions, which eventually negates the benefits of the additional reduction.

5. Conclusions

LDDs are a data structure that generalizes some other BDD variants that have been used in verification and performance analysis. LDDs, in turn, are themselves a special case of a more general structure, namely linear representations for formal power series, for which an associated minimization theory exists. Fully node-minimized LDDs are generally more compact than MTBDDs, but the minimization algorithm can be costly to execute. Partial minimization of LDDs can be done faster, and produces results that can still be significantly more compact than MTBDDs. Viewing MTBDDs and the like as special cases of linear representations might suggest various useful constructions and algorithms that would otherwise not be evident. A systematic treatment of numerical issues related to LDDs remains to be performed.

References

[1] S. Akers. Binary decision diagrams. *IEEE Transactions on Computers*, 27(6):509–516, 1978.

[2] R. I. Bahar, E. A. Frohm, C. M. Gaona, G. D. Hachtel, E. Macii, A. Pardo, and F. Somenzi. Algebraic decision diagrams and their applications. In *IEEE /ACM International Conference on CAD*, 1993.

[3] J. Berstel and C.Reutenauer. Rational series and their languages. *EATCS Monographs on Theoretical Computer Science*, 12, 1984.

[4] R. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 35(8):677–691, 1986.

[5] R. E. Bryant and Y.-A. Chen. Verification of arithmetic functions with binary moment diagrams. Technical report, Carnegie-Mellon University, 1994.

[6] P. Buchholz. Exact performance equivalence: An equivalence relation for stochastic automata. *Theoretical Computer Science*, 215:263–287, 1999.

[7] A. Cardon and M. Crochemore. Determination de la représentation d'une série reconnaissable. *R.A.I.R.O. Informatique Théorique*, 14(4):371–379, 1980.

[8] G. Ciardo and A. Miner. A data structure for the efficient Kronecker solution of GSPNs. In P. Buchholz and M. Silva, editors, *Proc. 8th International Workshop on Petri Nets and Performance Models*, pages 22–31. IEEE Computer Society Press, 1999.

[9] G. Ciardo and M. Tilgner. On the use of kronecker operators for the solution of generalized stochastic petri nets. Technical Report 96-35, ICASE Report, 1996.

[10] G. Ciardo and K. Trivedi. A decomposition approach for stochastic reward networks. *Performance Evaluation*, 18(1):37–59, 1993.

[11] E. Clarke, M. Fujita, P. McGeer, K. McMillan, J. Yang, and X. Zhao. Multi-terminal binary decision diagrams: An efficient data structure for matrix representation. In *Proc. International Workshop on Logic Synthesis (IWLS'93)*, pages 1–15, 1993. Also available in *Formal Methods in System Design*, 10(2/3):149-169, 1997.

[12] E. Clarke, K. McMillan, X. Zhao, M. Fujita, and J. Yang. Spectral transforms for large boolean functions with applications to technology mapping. In *Proc. 30th Design Automation Conference (DAC'93)*, pages 54–60. ACM Press, 1993. Also available in *Formal Methods in System Design*, 10(2/3):137-148, 1997.

[13] S. Eilenberg. *Automata, Languages and Machines, Vol. A*. Academic Press, 1974.

[14] B. Haverkort, H. Hermanns, and J.-P. Katoen. On the use of model checking techniques for dependability evaluation. In *In Proc. 19th IEEE Symposium on Reliable Distributed Systems (SRDS'00)*, pages 228–237, 2000.

[15] H. Hermanns, J. Meyer-Kayser, and M. Siegle. Multi-terminal binary decision diagrams to represent and analyse continuous time markov chains. *In Proc. 3rd International Workshop on the Numerical Solution of Markov Chains*, pages 188–207, 1999.

[16] O. Ibe and K. Trivedi. Stochastic petri net models of polling systems. *IEEE Journal on Selected Areas in Communications*, 8(9):1649–1657, 1990.

[17] T. Kam, T. Villa, R. K. Brayton, and A. Sangiovanni-Vincentelli. Multi-valued decision diagrams: theory and applications. *Multiple-Valued Logic*, 4(1-2):9–62, 1998.

[18] M. Kwiatkowska, G. Norman, and D. Parker. PRISM: Probabilistic symbolic model checker. In T. Field, P. Harrison, J. Bradley, and U. Harder, editors, *Proc. 12th International Conference on Modeling Techniques and Tools for Computer Performance Evaluation (TOOLS'02)*, volume 2324 of *LNCS*, pages 200–204. Springer-Verlag, 2002.

[19] M. Kwiatkowska, G. Norman, and D. Parker. Probabilistic symbolic model checking with PRISM: a hybrid approach. *International Journal on Software Tools for Technology Transfer (STTT)*, ?(?), 2004. (to appear).

[20] Y.-T. Lai, M. Pedran, and S. Vrudhula. Evbdd-based algorithms for linear integer programming, spectral transformation and function decomposition. *IEEE Transactions on CAD*, 13(8):959–975, 1994.

[21] C. Lee. Representation of switching circuits by binary-decision programs. *Bell System Technical Journal*, 38:985–999, 1959.

[22] A. Miner and D. Parker. Symbolic representations and analysis of large probabilistic systems. In *Validation of Stochastic Systems*. Springer-Verlag, 2003.

[23] B. Plateau. On the stochastic structure of parallelism and synchronization models for distributed algorithms. *Performance Evaluation Review*, 13:147–154, 1985.

[24] A. Salomaa and M. Soittola. *Automata-Theoretic Aspects of Formal Power Series*. Springer-Verlag, 1978.

[25] M. P. Schützenberger. On the definition of a family of automata. *Information and Control*, 4:245–270, 1961.

[26] E. Stark and G. Pemmasani. Implementation of a compositional performance analysis algorithm for probabilistic I/O automata. In *Proceedings of 1999 Workshop on Process Algebra and Performance Modeling (PAPM99)*. Prensas Universitarias de Zaragoza, Sept. 1999.

[27] E. W. Stark. Compositional performance analysis using probabilistic I/O automata. In C. Palamidessi, editor, *CONCUR 2000 - Concurrency Theory, 11th International Conference, University Park, PA, USA, August 22-25, 2000, Proceedings*, volume 1877 of *Lecture Notes in Computer Science*, pages 25–28. Springer-Verlag, 2000. Abstract of invited talk.

[28] E. W. Stark and S. A. Smolka. Compositional analysis of expected delays in networks of probabilistic I/O automata. In *Proceedings of the 13th Annual IEEE Symposium on Logic in Computer Science (LICS '98)*, pages 466–477, Indianapolis, IN, June 1998. IEEE Computer Society Press.

[29] P. Tafertshofer. Factored edge-valued binary decision diagrams and their application to matrix representation and manipulation. *Master thesis, Institute of Electronic Design Automation, Technical University of Munich*, 1994.

[30] S.-H. Wu, S. A. Smolka, and E. W. Stark. Compositionality and full abstraction for probabilistic I/O automata. In *Proceedings of CONCUR '94 — Fifth International Conference on Concurrency Theory*, Uppsala, Sweden, Aug. 1994.

[31] S.-H. Wu, S. A. Smolka, and E. W. Stark. Composition and behaviors of probabilistic I/O automata. *Theoretical Computer Science*, 176(1-2):1–38, 1997.

[32] D. Zhang, R. Cleaveland, and E. W. Stark. The integrated CWB-NC/PIOATool for functional verification and performance analysis of concurrent systems. In H. Garavel and J. Hatcliff, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 9th International Conference, TACAS 2003, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2003, Warsaw, Poland, April 7-11, 2003, Proceedings*, volume 2619 of *Lecture Notes in Computer Science*, pages 431–436. Springer-Verlag, 2003.

Cyclic Server Polling System

	model		fully minimized		partially minimized		deterministic		
	states	non-zeros	nodes	edges	nodes	edges	nodes	edges	terminal nodes
4	96	272	111	224	118	177	221	322	4
5	240	800	165	385	175	257	356	549	4
6	576	2,208	246	612	259	370	731	1137	4
7	1,344	5,824	324	883	340	482	1121	1823	4
8	3,072	14,848	408	1191	427	606	2082	3387	4
9	6,912	36,864	514	1561	536	754	3209	5481	4

Tandem Queueing Network

	model		fully minimized		partially minimized		deterministic		
	states	non-zeros	nodes	edges	nodes	edges	nodes	edges	terminal nodes
5	66	189	128	259	133	229	243	341	6
7	120	363	68	106	68	106	118	159	6
31	2015	6819	116	178	116	178	218	295	6
63	8128	27971	140	214	140	214	268	363	6

Flexible Manufacturing System

	model		fully minimized		partially minimized		deterministic		
	states	non-zeros	nodes	edges	nodes	edges	nodes	edges	terminal nodes
1	54	155	132	282	137	277	314	444	9
2	810	3699	1155*	4216*	1166	4222	3880	5956	14

* For these cases, nearly degenerate set(s) of vectors were encountered during minimization, and to avoid precision blowup, partial minimization was used instead of full minimization for the levels where the problem occurred.

Kanban Manufacturing System

	model		fully minimized		partially minimized		deterministic		
	states	non-zeros	nodes	edges	nodes	edges	nodes	edges	terminal nodes
1	160	616	133	226	134	254	387	524	14
2	4600	28120	2039	18063	2487	8315	7570	11447	14

Workstation Cluster

	model		fully minimized		partially minimized		deterministic		
	states	non-zeros	nodes	edges	nodes	edges	nodes	edges	terminal nodes
6	1652	7520	621	4716	865	2875	2643	3894	13
7	2176	10000	685	4990	907	3100	3135	4564	14
8	2772	12832	824	7556	1166	4443	4316	6337	15
9	3440	16016	596	1401	659	1185	4164	5984	16
10	4180	19552	1056	9205	1506	6466	5045	7323	17

Figure 1. Comparison of LDD Sizes for Benchmark Examples