# On the Representation of Probabilities over Structured Domains*

Marius Bozga and Oded Maler

Verimag, Centre Equation, 2, av. de Vignate, 38610 Gières, France,
**bozga@imag.fr maler@imag.fr**

**Abstract.** In this paper we extend one of the main tools used in verification of discrete systems, namely Binary Decision Diagrams (BDD), to treat probabilistic transition systems. We show how probabilistic vectors and matrices can be represented canonically and succinctly using probabilistic trees and graphs, and how simulation of large-scale probabilistic systems can be performed. We consider this work as an important contribution of the verification community to numerous domains which need to manipulate very large matrices.

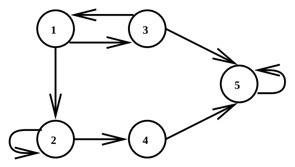
## 1 Introduction

Many problems in discrete verification can be reduced to the the following one: *given a non-deterministic finite-state automaton $\mathcal{A} = (Q, \delta)$ and a set $P \subseteq Q$ of states, find the set $P^*$ of all the states reachable from $P$.* One common way to do this calculation is to let $P^0 = P$ and $P^{i+1} = \delta(P^i)$ until $P^i$ is included in the union $P^0 \cup \ldots \cup P^{i-1}$. Here $P^i$ is the set of states reachable from $P$ after exactly $i$ steps.

This method can be formulated using Boolean state-vectors and transition matrices. Each subset $P$ of an $n$-element set of states can be written as an $n$-dimensional Boolean row vector $p$ (a function from $Q$ to $\{0,1\}$) and any transition relation $\delta$ as an $n \times n$ Boolean matrix $A_\delta$ (a function from $Q \times Q$ to $\{0,1\}$). Thus, the calculation step $P^{i+1} = \delta(P^i)$ is equivalent to the multiplication of a vector by a matrix: $p^{i+1} = p^i \cdot A_\delta$. For example, consider Figure 1 where a 5-state automaton is depicted along with its corresponding $5 \times 5$ matrix $A_\delta$. The reader can verify that calculating the states reachable in one step from $P = \{1,2\}$ is done via the multiplication $[1,1,0,0,0] \cdot A_\delta = [0,1,1,0,1]$ where logical conjunction and disjunction replace multiplication and addition, respectively.

Probabilistic transition systems, such as *discrete Markov chains*, operate in a similar but different fashion. At any given stage of the system's evolution the state is given by a probability function $p : Q \to [0,1]$ such that $\sum_{q \in Q} p(q) = 1$. The transition structure is probabilistic as well and is represented by a function

* This work was partially supported by the European Community Esprit-LTR Project 26270 VHS (Verification of Hybrid systems) and the French-Israeli collaboration project 970MAEFUT5 (Hybrid Models of Industrial Plants).



$$\begin{matrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{matrix}$$

**Fig. 1.** A non-deterministic automaton and its transition matrix.

$\delta : Q \times Q \to [0,1]$ where $\delta(q,q')$ denotes the *conditional probability* of being in $q'$ in the next-state *given* that the current state is $q$. The evolution from one probabilistic state vector to another is captured by the vector by matrix multiplication $p^{i+1} = p^i \cdot A_\delta$, this time over the reals.

The *state-explosion problem*, also known as *the curse of dimensionality*, arises when the system under consideration is composed of many sub-systems. The size of the global state-space is exponential in the number of components and verification by explicit enumeration of states and transitions becomes impossible. Symbolic methods provide an alternative to explicit state enumeration. They are based on the following observation: the global state-space of a composed system can be encoded naturally using *state-variables* (a variable for the local state of each component). The evolution of each variable usually depends on a small subset of the other variables and the corresponding transition law can be written concisely as a formula in some adequate formalism (e.g. propositional logic when the variables are Boolean) and the global transition relation is a conjunction of such formulae. Similarly, sets of states can be written down as formulae. With the aid of appropriate data-structures, a symbolic version of the basic computation $P^{i+1} = \delta(P^i)$ can be performed, calculating a (hopefully concise) representation of $P^{i+1}$ from a representation of $P^i$ and $\delta$.

In verification of systems modeled as automata this technique is called *symbolic model-checking* [McM93,BCM$^+$93] and it had a great success. In fact it can be seen as one of the breakthroughs in verification, facilitating the analysis of systems with hundreds of state variables, far beyond the capabilities of explicit enumeration on current and future computers. The most popular representation scheme used in symbolic verification is the *binary decision diagram* (BDD), which is a formalism for representing Boolean functions, admitting the following properties [B86,MT98]:

1. It is canonic – given an ordering of the variables, a unique BDD corresponds to every Boolean function.
2. There are relatively-efficient algorithms for manipulating BDDs, in particular for the operations needed to compute $P^{i+1} = \delta(P^i)$.
3. It performs well in the analysis of many structured systems: the size of the BDD remains small relative to the size of the state-space.

The goal of the paper is to apply this recipe to probabilistic systems, that is, to define a representation formalism for probabilistic vectors and transition functions such that the operation $p^{i+1} = p^i \cdot A_\delta$ could be performed for systems for which it is impossible to do so using currently existing methods. To this end we define *probabilistic decision graphs* (PDG)[1] , a data-structure for representing probabilities over structured domains which enjoys the nice properties of BDDs.

The rest of the paper is organized as follows. In section 2 we present probabilistic decision trees and graphs and show they constitute a canonic representation for probabilities. In section 3 we rephrase the basic definitions of Markov chains. Section 4 is devoted to the representation of probabilistic transition functions by *conditional* probabilistic graphs and sketch the PDG structure of some generic classes of probabilistic systems. The calculation of next-state probabilities on PDGs via the projection operation described in section 5 and some preliminary experimental results are reported in section 6. Finally we discuss the significance of this work and mention some of the previous relevant applications of BDD technology outside the Boolean realm.

## 2 Probabilistic Decision Graphs

Let $\mathbb{B} = \{0, 1\}$. We assume an underlying set $Q = \mathbb{B}^n$, and a probability distribution on $Q$, i.e. a function $p : Q \to [0, 1]$ such that $\sum_{q \in Q} p(q) = 1$. Such a function can be extended naturally to subsets of $Q$ by letting $p(Q') = \sum_{q \in Q'} p(q)$ for every $Q' \subseteq Q$. We will abuse strings from $\mathbb{B}^{\leq n}$ (the set of binary strings of length not greater than $n$) to denote certain subsets of $\mathbb{B}^n$. A string $u = x_1 x_2 \cdots x_n$ will stand for the singleton $\{(x_1, \ldots, x_n)\}$ while a string $x_1 x_2 \cdots x_i$, $i < n$ will stand for the set $\{(x_1, \ldots, x_i, x_{i+1}, \ldots, x_n) : (x_{i+1}, \ldots, x_n) \in \mathbb{B}^{n-i}\}$. This can be defined recursively by associating with $u$ the union of the sets associated with $u0$ and $u1$. Note that the empty string $\varepsilon$ denotes the whole $\mathbb{B}^n$. To avoid additional symbols we use the same notation for a string and for the set it denotes. The set $\mathbb{B}^{\leq n}$ has a binary tree structure and every level $\mathbb{B}^i$ corresponds to a partition of $\mathbb{B}^n$. The next definition is the essence of this paper.

**Definition 1 (Probabilistic Decision Trees).** *A probabilistic decision tree* (PDT) *of depth $n$ is a tuple $P = (S, 0, 1, v)$ where $S = \mathbb{B}^{\leq n}$, 0 and 1 are respectively the* left-successor *and* right-successor *partial functions on $S$, and $v : S \to [0, 1]$ is a function satisfying $v(\varepsilon) = 1$ and for every non-leaf node $s$, $v(s0) + v(s1) = 1$.*

**Theorem 1 (Unique Representation).** *There is a one-to-one*[2] *correspondence between probabilities on $\mathbb{B}^n$ and PDTs.*

**Proof**: First we assign probabilities to nodes by letting $p(\varepsilon) = 1$ and

$$p(sx) = p(s) \cdot v(sx) \qquad x \in \mathbb{B} \tag{1}$$

It is not hard to see that all $p$ values are in $[0, 1]$ and that their sum at each level of the tree is 1. Conversely, given a probability on the leaves, it is straightforward to

[1] We say "graphs" instead of "diagrams" to avoid yet another xDD acronym.

[2] In our definition there is an implicit ordering on the "variables".

calculate the probability of the sets associated with the upper nodes by letting $p(s) = p(s0) + p(s1)$ and then compute $v$ via normalization, i.e. the inverse of (1): $v(sx) = p(sx)/p(s)$. In the case when $p(s) = 0$ we can put any number in $v(sx) = 0/0$, and a convention such as $1/2$ can be used. ∎

PDTs are nothing but the presentation of probabilities using the so-called "chain-rule", the probabilistic analogue of Shannon factorization of Boolean functions which underlies BDDs:

$$p(x_1 x_2 \cdots x_n) = p(x_1) \cdot p(x_1 x_2 | x_1) \cdots p(x_1 x_2 \cdots x_n | x_1 \cdots x_{n-1})$$

where $p(r|s)$ is the conditional probability of $r$ given $s$. We will replace this unfortunate (but very common) notation with $p_s(r)$ such that the above rule will be written as

$$p(x_1 x_2 \cdots x_n) = p(x_1) \cdot p_{x_1}(x_1 x_2) \cdots p_{x_1 \cdots x_{n-1}}(x_1 \cdots x_n).$$

Decision trees are exponential in the number of variables and, by themselves, do not solve the state explosion problems. However, when there is some structure in the objects they represent, different nodes may have identical sub-trees and the tree can be represented concisely by a directed acyclic graph (DAG) carrying the same information. The transformation of a tree into a DAG is a variation of the classical procedure for minimizing automata, and can be phrased as follows.

**Definition 2 (Probabilistic Decision Graphs).** *Let $P = (S, 0, 1, v)$ be a* PDT *and let $\sim$ be a congruence relation*[3] *on $S$ defined as $s \sim s'$ if $v(s) = v(s')$ and both $s0 \sim s'0$ and $s1 \sim s'1$. The associated probabilistic decision graph* (PDG) *is $G = (S/\sim, 0, 1, v)$.*

In other words, the nodes of G are the equivalence classes of $\sim$. Graphically speaking, the process starts from the bottom of the tree by merging leaves $sx$ and $s'x'$ which have identical $v$'s. Then the edge from $s$ labeled by $x$ and the edge from $s'$ labeled by $x'$ are redirected toward the merged node and the process continues recursively upward. Note that $sx = \bot$ for a leaf $s$, hence $s \sim s'$ only if both belong to the same level of the tree.
**Example**: Consider the following probability function over $\mathbb{B}^3$:

| 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 |
|---|---|---|---|---|---|---|---|
| $\frac{1}{6}$ | 0 | $\frac{2}{15}$ | $\frac{1}{30}$ | $\frac{4}{15}$ | $\frac{1}{15}$ | $\frac{1}{15}$ | $\frac{4}{15}$ |

Figure 2-(a) shows the probabilities of all subsets in $\mathbb{B}^{\leq 3}$. The PDT in Figure 2-(b) is obtained via the normalization $v(sx) = p(sx)/p(s)$. The reduction modulo $\sim$ into a PDG starts in Figure 2-(c) by merging identical leaves and terminates in Figure 2-(d) by merging some of their parents.[4] Like in BDDs, when there is

[3] Congruence with respect to the 0 and 1 operations.
[4] Unlike BDDs we do not go further and eliminate nodes whose left and right successors are identical: we restrict ourselves to *balanced* DAGs where all paths from the root to the leaves are of the same length, otherwise we cannot satisfy the requirement that the sum of the leaves at every level is 1.

a lot of independence between the variables, the size of the PDG is much smaller than the size of $Q$. In the rest of the paper we describe algorithms in terms of full trees, bearing in mind that the actual implementation reduces every tree into its corresponding minimal DAG.

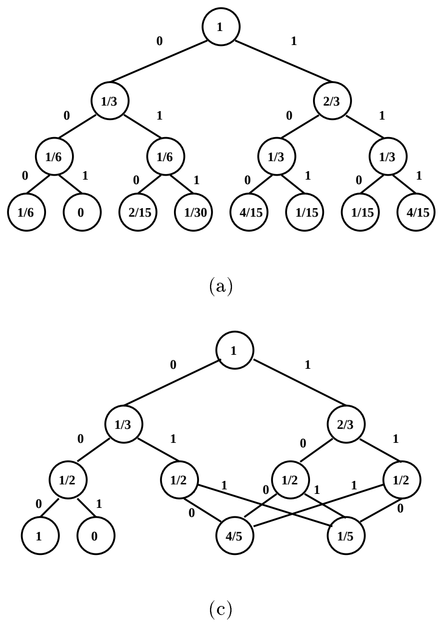


**Fig. 2.** Transforming a probability function (a) into a PDT (b) and successively via (c) into a PDG (d).

## 3 Markov Transition Functions

Having defined a canonical representation for probabilistic state vectors, we now move to the representation of transition matrices. In a non-probabilistic setting there is not much difference between sets (subsets of $\mathbb{B}^n$) and relations (subsets of $\mathbb{B}^{2n}$) and both can be represented by BDDs of the same type. For probabilistic systems, we must be more careful.

**Definition 3 (Markov Transition Function).** *A Markov transition function on $Q$ is a function $\delta : Q \to (Q \to [0,1])$ such that for every $q \in Q$, $\delta_q : Q \to [0,1]$ is a probability function on $Q$.*

In 20th century mathematics, such functions used to be written as $|Q| \times |Q|$ matrices such as

$$A_\delta \;=\; \begin{array}{llll} \delta_1(1) & \delta_1(2) & \ldots & \delta_1(n) \\ \delta_2(1) & \delta_2(2) & \ldots & \delta_2(n) \\ \ldots & \ldots & \ldots & \ldots \\ \delta_n(1) & \delta_n(2) & \ldots & \delta_n(n) \end{array}$$

where each line represents a particular $\delta_q$. The action of $\delta$ on a probabilistic state-vector $p$ can be decomposed into two stages. The first can be viewed as applying a function $\hat{\delta} : (Q \to [0,1]) \to (Q \times Q \to [0,1])$ where $\hat{p} = \hat{\delta}(p)$ if for every $q, q' \in Q$, $\hat{p}(q,q') = p(q) \cdot \delta_q(q')$. In other words, given that the current state probability is $p$, $\hat{\delta}(p)$ denotes the probability of any *transition* to happen. Matrix-wise, when $p$ is written as a vector $[p_1, \ldots, p_n]$, calculating $\hat{\delta}(p)$ amounts to multiplying every element of $p$ by the elements of its corresponding row in $\delta$ to obtain

$$A_{\hat{\delta}(p)} \;=\; \begin{array}{llll} p_1 \cdot \delta_1(1) & p_1 \cdot \delta_1(2) & \ldots & p_1 \cdot \delta_1(n) \\ p_2 \cdot \delta_2(1) & p_2 \cdot \delta_2(2) & \ldots & p_2 \cdot \delta_2(n) \\ \ldots & \ldots & \ldots & \ldots \\ p_n \cdot \delta_n(1) & p_n \cdot \delta_n(2) & \ldots & p_n \cdot \delta_n(n) \end{array}$$

Note that unlike $\delta$, $\hat{\delta}(p)$ *is* a probability function on $Q \times Q$.

The probability of being in the next step at a state $q'$ is then the sum of the probabilities of the form $\hat{p}(q,q')$, i.e. those leading to $q'$. This can be captured by a function: $w : (Q \times Q \to [0,1]) \to (Q \to [0,1])$ defined as $w(\hat{\delta}) = \sum_i \hat{\delta}_i$. Matrixly speaking, this is equivalent to summing up every column of $A_{\hat{\delta}(p)}$ to obtain a vector $p'$. Hence the composition $w \circ \hat{\delta} : (Q \to [0,1]) \to (Q \to [0,1])$ gives the evolution of the system as the action of a probabilistic transition matrix on a probabilistic state vector.[5]

Next we define a data-structure for representing $\delta$ when $Q = \mathbb{B}^n$ and a natural way to transform it, given a PDG-represented probability $p$, into a PDG of depth $2n$ for $\hat{\delta}(p)$. After that we define the basic operation on PDGs, the *projection* which is used in the calculation of $w$.

## 4 Conditional Probabilistic Decision Graphs

The basic idea is to extend PDTs such that nodes at certain levels of the tree are empty (with $v$ undefined) to denote undetermined variables.[6] To this end we will use somewhat more elaborate notations.

Let $X = \{1^x, 2^x, \ldots, n^x\}$ and $Y = \{1^y, 2^y, \ldots, n^y\}$ be two copies of $\{1, \ldots, n\}$. An order relation $\prec$ on $X \cup Y$ can be written as a bijection $J : \{1, 2, \ldots, 2n\} \to$

[5] For those familiar with BDDs, we mention that these operations resemble the non-probabilistic ones: $\hat{\delta}(q,q') = p(q) \wedge \delta(q,q')$ and $w(q') = \exists q\ \hat{\delta}(q,q') = \bigvee_q\ \hat{\delta}(q,q')$.

[6] In fact we could have started the paper by defining data-structures for conditional probability functions, with a partition of variables into two types. This way we could obtain probability functions as the special case where all the variables are determined, and Markov transition functions as a special case where the sizes of the two sets of variables are the same and certain restrictions are imposed on variable dependencies. However, we prefer clarity over generality.

$X \cup Y$. Without loss of generality we assume that $\prec$ is compatible with the natural ordering of $X$ and of $Y$, i.e. $1^x \prec 2^x \prec \ldots \prec n^x$. Given $J$, any binary string $s \in \mathbb{B}^{\leq 2n}$ can be mapped into a pair of strings $J^x(s)$ and $J^y(s)$ from $\mathbb{B}^{\leq n}$. For example, if $J = 1^x \prec 2^x \prec 1^y \prec 3^x \prec 2^y \prec 3^y$ then for a string $s = x_1 x_2 y_1 x_3 y_2 y_3$, $J^x(s) = x_1 x_2 x_3$ and $J^y(s) = y_1 y_2$. We also extend our string notation for sets: a string of the form $x_{i_1} x_{i_1} \cdots x_{i_m}$ with $0 < i_1 < i_2 < \ldots < i_m \leq n$ will denote a subset of $\mathbb{B}^n$ with the obvious meaning, i.e. the set of $n$-tuples such that the value of every $i_j$-coordinate is $x_{i_j}$.

A Markov transition function over $\mathbb{B}^n$ is a function $\delta : \mathbb{B}^n \to (\mathbb{B}^n \to [0,1])$ whose instances are written as $\delta_{x_1 \cdots x_n}(y_1 \cdots y_n)$. For every $x_1 \cdots x_n$, $\delta_{x_1 \cdots x_n}$ is a probability function which can be written using the chain rule just as as any other probability:

$$\delta_{x_1 \cdots x_n}(y_1 \cdots y_n) = \delta_{x_1 \cdots x_n}(y_1) \cdot \delta_{x_1 \cdots x_n y_1}(y_1 y_2) \cdots \delta_{x_1 \cdots x_n y_1 \cdots y_{n-1}}(y_1 \cdots y_n).$$

We restrict our attention to Markov chains in which every coordinate of the state-space behaves *causally*, i.e. it depends only on the *previous* values of the state variables.[7] This means that for every $x_1 \ldots x_n$ and every $y_i$, $y_j$ we have $\delta_{x_1 \cdots x_n y_i}(y_j) = \delta_{x_1 \cdots x_n}(y_j)$. Hence $\delta$ can be written as:

$$\delta_{x_1 \cdots x_n}(y_1 \cdots y_n) = \delta_{x_1 \cdots x_n}(y_1) \cdot \delta_{x_1 \cdots x_n}(y_2) \cdots \delta_{x_1 \cdots x_n}(y_n). \tag{2}$$

We say that $j^y$ is *independent* of $i^x$ if for every $x_1, \ldots, x_{i-1}, x_{i+1}, \ldots x_n$,

$$\delta_{x_1 \cdots x_{i-1} 0 x_{i+1} \cdots x_n}(y_j) = \delta_{x_1 \cdots x_{i-1} 1 x_{i+1} \cdots x_n}(y_j).$$

In this case we can use the notation $\delta_{x_1 \cdots x_{i-1} x_{i+1} \cdots x_n}(y_j)$. When this is not the case we say that $i^x$ *influences* $j^y$ and denote it by $i^x \rightharpoondown j^y$.

An order relation $\prec$ on $X \cup Y$ is *compatible* with a Markov transition function $\delta$ iff for every $i^x \in X, j^y \in Y$, $i^x \rightharpoondown j^y$ implies $i^x \prec j^y$. The *default ordering* $1^x \prec \ldots \prec n^x \prec 1^y \prec \ldots \prec n^y$ is compatible with *any* $\delta$ and is the only one compatible with a $\delta$ for which every $j^y$ depends on all $X$.

**Definition 4 (Conditional PDT and PDG).** *A conditional probabilistic decision tree* (CPDT) *of depth $n$ is a tuple $P = (S, 0, 1, J, v)$ where $S = \mathbb{B}^{\leq 2n}$,* 0 *and* 1 *are as in a PDT, $J$ is the ordering bijection and $v : S \to [0,1]$ is a partial function, defined only on nodes $s$ such that $J(|s|) \in Y$, satisfying $v(\varepsilon) = 1$ and for every node $s$, $v(s0) + v(s1) = 1$ whenever it is defined. A conditional probabilistic decision graph* (CPDG) *is $G = (S/\sim, 0, 1, J, v)$ where $\sim$ is the congruence relation of Definition 2.*

**Theorem 2 (CPDT=Markov Transition Function).** *There is a one-to-one correspondence between Markov transition functions and CPDTs.*

[7] Note that one can write Markov transition functions over $Q$ which *do not* admit such a causal decomposition, and this observation might be a source of interesting investigations in the theory of stochastic processes. In fact, the above implies that every $m$-state Markov chain which admits a causal decomposition can be represented in space $O(m \log m)$ instead of $O(m^2)$.

**Sketch of Proof**: Similar to that of Theorem 1. We assume a fixed ordering bijection $J$ compatible with $\delta$. For every $Y$-node $sy_i$ of the CPDT we associate $v(sy_i)$ with the conditional probability $\delta_{J^x(s)}(y_i)$, for example $v(x_1x_2y_1x_3y_2) = \delta_{x_1x_2x_3}(y_2)$. To reconstruct $\delta$ from a tree we go down the tree until we calculate $\delta$ for the lowest $Y$-nodes. To build a CPDT from $\delta$ we climb-up starting from the $Y$-leaves and construct the tree. ∎

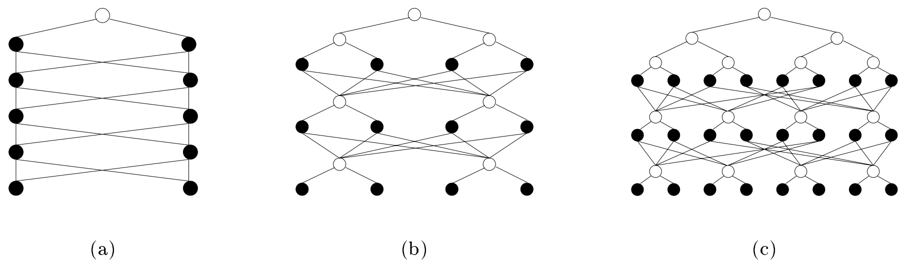


**Fig. 3.** Schematic CPDGs for Markov transition function which consist of: (a) Independent Bernoulli trials (b) Independent Markov chains (c) A cascade with $k = 2$. The dark nodes indicate $Y$-nodes.

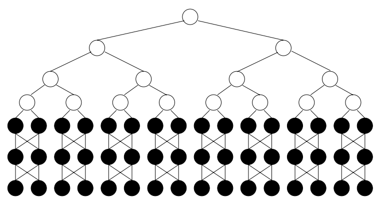


**Fig. 4.** A schematic CPDG for an arbitrary (but causal) Markov transition function.

We mention some classes of probabilistic transition systems such that the pattern of interaction between their components alone suffices for giving an upper-bound on the size of their CPDGs. Consider first the degenerate case of $n$ independent Bernoulli trials. It can be modeled as a direct product of $n$ memory-less automata, for which the probability of the next state is independent of the current state. Thus, $\delta_{x_1 \ldots x_n}(y_1 \ldots y_n)$ can be written as $\delta(y_1) \cdots \delta(y_n)$ and represented by a CPDG without empty nodes, which is in fact a PDG, like in Figure 3-(a).

As a slightly less trivial example consider a direct product of $n$ independent 2-state Markov chains. In this case each $i^y$ depends only on $i^x$ and the transition function can be represented by the CPDG of Figure 3-(b). More generally, consider a *cascade* $\mathcal{A}_1, \ldots, \mathcal{A}_n$ of probabilistic automata where the transition probabilities of each automaton $\mathcal{A}_i$ depends on the states of its $k$ predecessors (including itself) $\mathcal{A}_{i-k+1}, \ldots, \mathcal{A}_{i-1}, \mathcal{A}_i$. Such systems will have a CPDG of size $O(n2^k)$ similar to the one appearing in Figure 3-c for $k = 2$.

When there are no such constraints on variable dependencies, the default order needs to be used and no a-priori lower-bound better than $n2^n$ can be stated (although some independencies might make the corresponding CPDG smaller). We repeat that even this bound is better than the $2^{2n}$ size implied by a straightforward encoding of the transition matrix. The general structure of such a CPDG is depicted in Figure 4.

Going from $p$ and $\delta$ to $\hat{\delta}(p)$ is straightforward: take $v(s)$ from the PDT for $p$ and put it in any nodes $s'$ of the CPDT of $\delta$ such that $J^x(s') = s$. This way the whole tree becomes full and represents the probability $\hat{\delta}(p)$ over $\mathbb{B}^{2n}$.

## 5 Projection

The basic operation on probabilities (and PDGs) is the probabilistic analogue of the elimination of a quantified variable in Boolean functions (and BDDs). This is what is needed to transform $\hat{\delta}(p)$ into $\delta(p)$.

**Definition 5 (Projection).** *Let $p : \mathbb{B}^n \to [0,1]$ be a probability. The $k$-projection of $p$, is a function $p_{\downarrow k} : \mathbb{B}^{n-1} \to [0,1]$ defined as*

$$p_{\downarrow k}(x_1 \cdots x_{k-1} x_{k+1} \cdots x_n) = \begin{array}{c} p(x_1 \cdots x_{k-1} 0 x_{k+1} \cdots x_n) \\ + \\ p(x_1 \cdots x_{k-1} 1 x_{k+1} \cdots x_n) \end{array} \tag{3}$$

Using conditional probabilities, (3) can be rewritten as

$$p(x_1 \ldots x_{k-1}) \cdot \begin{bmatrix} p_{x_1 \cdots x_{k-1}}(0 x_{k+1} \cdots x_n) \\ + \\ p_{x_1 \cdots x_{k-1}}(1 x_{k+1} \ldots x_n) \end{bmatrix}$$

and further as

$$p(x_1 \cdots x_{k-1}) \cdot \begin{bmatrix} v(x_1 \cdots x_{k-1} 0) \cdot p_{x_1 \cdots x_{k-1} 0}(x_{k+1} \cdots x_n) \\ + \\ v(x_1 \cdots x_{k-1} 1) \cdot p_{x_1 \cdots x_{k-1} 1}(x_{k+1} \cdots x_n) \end{bmatrix}$$

As one can see, performing a $k$-projection on the PDT representation of $p$ consists of copying the first $k-1$ levels of the tree and then plugging at each branch $x_1 \cdots x_{k-1}$ a sub-tree which encodes the *weighted sum* of the functions $p_{x_1 \cdots x_{k-1} 0}$ and $p_{x_1 \cdots x_{k-1} 1}$. This is the main computational burden in the manipulation of PDGs. The transformation of a PDT $\mathrm{P} = (\mathrm{S}, 0, 1, v)$ for $p$ with $\mathrm{S} = \mathbb{B}^n$ into a PDT $\mathrm{P}_{\downarrow k} = (S_{\downarrow k}, 0, 1, v_{\downarrow k})$ for $p_{\downarrow k}$ with $S_{\downarrow k} = \mathbb{B}^{n-1}$ is performed as follows. For any node $s \in \mathbb{B}^{\leq k-1}$ we have $p_{\downarrow k}(s) = p(s)$. For the other nodes we have

$$p_{\downarrow k}(x_1 \cdots x_{k-1} s) = p(x_1 \cdots x_{k-1} 0 s) + p(x_1 \cdots x_{k-1} 1 s)$$

These values are calculated from the top down and every calculation of $p_{\downarrow k}(sx)$ is followed by calculating $v_{\downarrow k}(sx)$ as $v_{\downarrow k}(sx) = p_{\downarrow k}(sx)/p_{\downarrow k}(s)$, which in the first $k-1$ levels reduces simply to $v_{\downarrow k}(s) = v(s)$. Aplying this procedure $n$ times[8] we transform a probability on $\mathbb{B}^{2n}$ to a probability on $\mathbb{B}^n$ and complete the computation of $p' = p \cdot A_\delta$. While working with PDGs, one can avoid part of the computation whenever there is an equivalence of the form $p_{s0} = p_{s1}$. In that case the weighted sum $r \cdot p_{s0} + (1-r)p_{s1}$ is equal to both.

## 6 Implementation and Experimental Results

The treatment of the mathematical *real* numbers by computer involves an additional dimension of problematics absent from traditional applications of verification methodology. The continuum is approximated by a very large (but finite) subset of the rationals, the *floating point* numbers. Practitioners seem to be satisfied with this approximation. It turns out that for exploiting the advantages of PDGs we had to go further and round node values to multiples of $2^{-m}$ (for $m$ ranging between 3 to 10), otherwise the size of non-trivial PDGs becomes exponential after few iterations because of the low probability of two nodes having *exactly* the same floating-point value. With this discretization, systems with limited interaction among variables usually converge to vectors with a small PDG description. As for the semantic price of the approximation, if we reflect a bit on the empirical source of probability estimations in models, we realize that these numbers are not sacred and an initial "imprecision" of $2^{-m}$ does not make any difference.

We have implemented these data-structures and algorithms and tested their performance on some generic examples. The implementation is preliminary and does not yet employ all the optimizations one can find in BDD packages. Let us first mention the trivial cases. For $n$ randomly-generated mutually-independent Markov chains we can treat almost any $n$. This is, of course, not so impressive if one realizes that each chain could be simulated separately. Yet someone unaware of BDDs will be rather surprised to see how fast you can multiply a $2^{15} \times 2^{15}$ transition matrix void of any apparent structure or sparseness (see table 1). A slightly less trivial example is a chain of noisy communication channels where each component copies the value of its predecessor with probability $1-\epsilon$. Such a chain converges to a uniform probability vector where $p(q) = 1/2^n$ for every state. Here again we could iterate for very large $n$ with a linear growth in the size of the PDGs.

Next, we have tested randomly-generated cascades of communication depth 2, which using the previously mentioned discretization, usually converge to vectors with small PDGs, although exponential ones are, of course, still possible. We demonstrate the time and space behavior of the algorithm on a family consisting of a cascade of noisy AND gates such that each component becomes the conjunction of its previous value and that of its predecessors (Figure 5) with

[8] Like in BDDS, this procedure can be extended naturally to a procedure that eliminates several variables in a single pass.

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 0.000564 | 0.000093 | 0.000412 | 0.000068 | 0.000094 | 0.000015 | 0.000068 | 0.000011 | 0.000727 | 0.000120 | ... |
| 0.000653 | 0.000003 | 0.000477 | 0.000002 | 0.000108 | 0.000001 | 0.000079 | 0.000000 | 0.000842 | 0.000004 | ... |
| 0.000823 | 0.000135 | 0.000153 | 0.000025 | 0.000137 | 0.000022 | 0.000025 | 0.000004 | 0.001061 | 0.000175 | ... |
| 0.000953 | 0.000005 | 0.000177 | 0.000001 | 0.000158 | 0.000001 | 0.000029 | 0.000000 | 0.001229 | 0.000006 | ... |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |

**Table 1.** An initial fragment of a $2^{15} \times 2^{15}$ matrix which can be iterated until convergence within less than a second.

probability 0.9. The performance results are depicted in Figure 6 and although space behaves nicely, computation time still grows exponentially, reaching almost 4 hours for $n = 54$. The reason lies in the fundamental difference between BDDs and PDGs: in the former, when an algorithms encounters a node, it does not need to remember via which branch the node is reached, and thus the hashing mechanism prevents duplicate calls. On the other hand, in PDGs, each time the projection procedure is called with a node, it has, as an additional parameter, the probability associated with its parent. Hence procedure calls with identical arguments are rather rare and the current implementation needs to do exponential work on linear-sized PDGs. We are currently investigating improvements of the implementation.

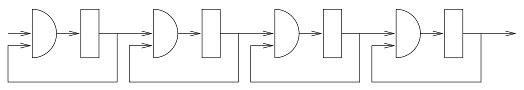


**Fig. 5.** A chain of noisy AND gates.

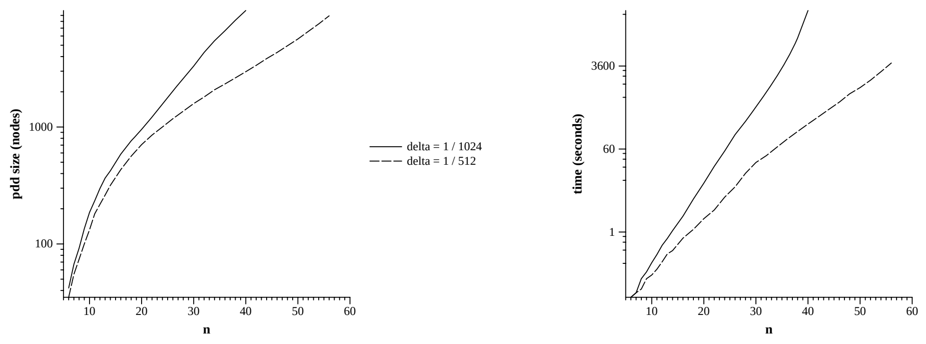


**Fig. 6.** The PDG size and time until convergence as a function of the number of variables, for discretizations of 1/1024 and 1/512.

## 7 Discussion

We have introduced and implemented a new method for manipulating large probabilistic transition systems. We hope that this technique will improve the performance of probabilistic simulation tools. In addition, the investigation of the structure of PDGs might contribute to a better understanding of the structure of probabilistic functions. The application domains which might benefit from such a technique are numerous and include performance and reliability analysis, probabilistic verification, planning under uncertainty [P94,BDH99], calculation of equilibria in economics, statistical mechanics and more.

This work is built on what we consider to be the main insight of the BDD experience: in many situations the indices of rows and columns in matrices are the outcome of "flattenning" of much more structured domains. This flattening, which is unavoidable if one wants to draw a matrix on a two-dimensional sheet of paper, hides the structure of the problem, or at least makes it very hard to retrieve.[9] BDDs and PDGs suggest a way of maintaining this structural information and exploiting it in efficient computations.

Among previous extensions of BDD technology to represent functions from $\mathbb{B}^n$ to $\mathbb{N}$ (motivated chiefly by arithmetical circuits), $\mathbb{R}$ and other domains we mention the structure called Multi-terminal BDDs (MTBDD) in [CFM$^+$93] and Algebraic Decision Diagrams (ADD) in [BFG$^+$93]. This is a straightforward extension of BDDs with leaves having values in non-Boolean domains. Algorithms for performing matrix multiplications and other operations on these representations have been proposed and applied, for example, to probabilistic verification [BCG$^+$97]. The main drawback of MTBDDs/ADDs is that they yield a succint representation only if the corresponding vectors and matrices have a lot of *identical* entries, e.g. sparse matrices having many zeros. In contrast many generic examples of functions with no interaction between the variables will lead to exponential MTBDDs: for example it is not hard to create probabilities on $\mathbb{B}^n$ with all variables mutually-independent, and yet no two elements will have the same probability. In fact, the ability to represent functions concisely as decision graphs *without* putting any information on the non-leaf nodes is a special property of Boolean algebra.

The above observation has led some researchers in the hardware verification community [VPL96,TP97] to consider extending BDD with values on their edges (which is practically the same as putting values on the nodes, as we do here). This structure is called Edge-valued BDD (EVBDD) and it has been used to encode the so-called Pseudo-Boolean functions which are essentially functions from $\{0,1\}^n$ to $\mathbb{N}$. EVBDDs contain both additive and multiplicative constants and in some cases overcome the limitations of MTBDDs. However, since the class of functions treated by EVBDDs is much less constrained than the class of

[9] Just compare the non-intuitive definition of the Kronecker product (also known as Tensor product) of two matrices with the straightforward Cartesian product of automata.

probabilistic functions, normalization and matrix multiplication are much more complicated than the ones reported in this paper.

Finally, let us mention another formalism, related to PDGs, the *Bayesian Networks* which are used extensively in AI [P88,J96]. Like PDGs, Bayesian networks consist of a graphical representation of variables and their probabilistic dependencies. The comparison between the two formalisms is outside the scope of this paper, but it seems that PDGs can be viewed as a constrained and well-behaving sub-class of networks, with a special emphasis on the *dynamic* aspects (next-state probabilities) which makes them, perhaps, more suitable for treating large-scale Markov decision processes.

**Acknowledgements**: We are grateful to Moshe Tennenholz for raising the possibility of applying some verification techniques to AI problems of planning under uncertainty. His visit in Grenoble, in fact, triggered this work. We thank Amir Pnueli for many fruitful discussions at various stages of this work, and in particular for the observations concerning causal Markov chains and weighted sum of identical sub-trees. Eugene Asarin reminded us of certain facts concerning the convergence of probabilistic matrix multiplications.

## References

[B86] R.E. Bryant, Graph-based Algorithms for Boolean Function Manipulation, *IEEE Trans. on Computers* C-35, 677-691, 1986.

[BCM$^+$93] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang, Symbolic Model-Checking: $10^{20}$ States and Beyond, *Information and Computation* 98, 142-70, 1992.

[BDH99] C. Boutilier, T. Dean and S. Hanks, Decision Theoretic Planning: Structural Assumptions and Computational Leverage, *J. of AI Research* (to appear).

[BFG$^+$93] R.I. Bahar, E.A. Frohm, C.M. Ganoa, G.D. Hachtel, E. Macii, A. Pardo and F. Somenzi, Algebraic Decision Diagrams and their Applications, *Proc. ICCAD'93*, 188-191, 1993.

[BCG$^+$97] C. Baier, E. Clarke, V. Garmhausen-Hartonas, M. Kwiatkowska and M. Ryan, Symbolic Model Checking for Probabilistic Processes, in P. Degano, R. Gorrieri and A. Marchetti-Spaccamela (Eds.), *Proc. ICALP'97*, 430-440, LNCS 1256, Springer, 1997.

[CFM$^+$93] E. M. Clarke, M. Fujita, P. C. McGeer, K. L. Mcmillan and J. C.-Y. Yang, Multi-terminal Binary decision Diagrams: An Efficient Data-structure for Matrix Representation, *Proc. ILWS'93*, 1-15, 1993.

[J96] F.V. Jensen, *An Introduction to Bayesian Networks*, Springer, 1996.

[McM93] K.L. McMillan, *Symbolic Model-Checking: an Approach to the State-Explosion problem*, Kluwer, 1993.

[MT98] C. Meinel and T. Theobald, *Algorithms and Data Structures in VLSI Design: OBDD - Foundations and Applications*, Springer, 1998.

[P88] J. Pearl, *Probabilistic Reasoning in Intelligent Systems*, Morgan Kaufmann, 1988.

[P94] M.L. Puterman, *Markov Decision Processes*, Wiley, 1994.

[TP97] P. Tafertshofer and M. Pedram, Factored Edge-Valued Binary Decision Diagrams, *Formal Methods in system Design* 10, 137-164, 1997.

[VPL96] S. B. K. Vrudhula, M. Pedram and Y.-T. Lai, Edge-valued Binary Decision Diagrams, in T. Sasao and M. Fujita (Eds.), *Representations of Discrete Functions*, 109-132, Kluwer, 1996.