# Multi Terminal Binary Decision Diagrams to Represent and Analyse Continuous Time Markov Chains

H. Hermanns[1] and J. Meyer-Kayser[2] and M. Siegle[2]

[1] Systems Validation Centre, Dept. of Computer Science, University of Twente, The Netherlands
[2] IMMD 7, Dept. of Computer Science, University of Erlangen-Nürnberg, Germany

**Abstract.** Binary Decision Diagrams (BDDs) have gained high attention in the context of design and verification of digital circuits. They have successfully been employed to encode very large state spaces in an efficient, *symbolic* way. Multi terminal BDDs (MTBDDs) are generalisations of BDDs from Boolean values to values of any finite domain. In this paper, we investigate the applicability of MTBDDs to the symbolic representation of continuous time Markov chains, derived from high-level formalisms, such as queueing networks or process algebras. Based on this data structure, we discuss iterative solution algorithms to compute the steady-state probability vector that work in a completely symbolic way. We highlight a number of lessons learned, using a set of small examples.

## 1 Introduction

*State space explosion* is a notorious problem in the context of continuous time Markov chains (CTMC) derived from high-level formalisms such as queueing networks, stochastic automata, stochastic Petri nets or stochastic process algebras. The inherent concurrency in such a high-level representation is translated into an interleaving of all possible moves. As a consequence, the number of states of a CTMC tends to grow exponentially in the number of parallel components from which it is derived. In addition, using approximations of non-exponential distributions by phase-type distributions leads to an enormous growth of the state space. Various approaches to attack the state space explosion problem have been pursued so far. Without aiming to be exhaustive we mention *product-form* solutions [BCMP75], *decomposition-based solution* [SA61,CS85], *tensor-based* representations [PA91,Buc94,Don93,Sie94b], *symmetry exploitation* [CDFH93,HR98,Sie94a] and *partial order* representations [BKLL95].

In the field of design and verification of digital circuits, state space explosion is a similarly omnipresent phenomenon. During the last decade, however, *binary decision diagrams* (BDDs) [Bry86] have moved the border between manageable and unmanageable sizes of verification problems upwards by many orders of magnitude. Papers have been published presenting verification results of state spaces larger than the number of atoms in the whole universe, see for instance

[BCL91,BCM+92]. BDDs are compact and canonical representations of Boolean functions as directed acyclic graphs, where the representation of redundant information is completely avoided. A proper assessment of BDDs, however, has to mention that much of their success is based on *heuristics*, specifically concerning the encoding of state spaces as Boolean functions and the avoidance of redundancies in the representation of these functions as BDDs. Without applying these heuristics, BDDs do not necessarily behave better than ordinary state space encodings, such as linked lists, for instance.

Multi terminal BDDs [FM97] (MTBDDs, also called algebraic decision diagrams) extend BDDs such that they can represent functions of an arbitrary range, while their domain is still a multidimensional Boolean space. Other BDD extensions with the same capability are edge-valued BDDs [LS92], and decision-node BDDs [HS99]. In this way, state spaces can be encoded where transitions between states are labelled by some value, such as a non-negative real number. Hence, MTBDDs are capable of representing discrete and continuous time Markov chains. [HMPS96] contains an investigation of the benefits when representing discrete time Markov chains (DTMCs) derived from digital circuits by MTBDDs. In a nutshell, it describes how steady-state probabilities of DTMCs of up to $10^{27}$ states can be computed by applying iterative numerical solution methods, namely the power method and a variant of the method of Jacobi. Direct solution (such as LU decomposition) of the Markov chain turns out not to be well-suited for the MTBDD framework. Apart from the size of the problem itself, a major limitation of direct methods is that each step modifies the MTBDD structure, and hence involves serious overhead to keep the representation canonical [BFG+97].

In this paper, we investigate the application of MTBDDs to the iterative solution of continuous time Markov chains and report on a number of decisive fine points in order to derive small MTBDD representations for large Markov chains, a topic that is not discussed in [HMPS96]. We give reformulations of classical iterative solution methods for CTMCs in the setting of MTBDDs which can be implemented in a straight-forward manner. The main contribution of this paper is a set of *heuristics* that are an essential part of a successful application of MTBDDs, in the same way as the success of BDDs is grounded on heuristics. The essential observation underlying these heuristics is that there is usually a large degree of freedom when encoding states as bit vectors, and that small differences in encodings can have tremendous effects on the size of the MTBDD that (still) has to be kept in memory during the iterative solution of the Markov chain. Optimal encodings exist, but unfortunately this issue is known to be NP-hard [THY93,BW96]. For practical purposes, we introduce heuristics in the form of *rules of thumb* that exploit the structure of the Markov chain based on the structure of the high-level specification at hand. We consider high-level specifications in the form of failure-repair models, stochastic process algebra models and queueing networks. For some cases, these heuristics have just a modest effect, while for others the result is *far beyond* any other data structure used to represent Markov chains so far. In particular, we present a class of queueing models

that can be encoded in such a way that an exponential increase in the number of queue positions only *linearly* increases the memory requirements. For instance, we show how a tandem queueing network with queue lengths of $10^{877}$ results in a memory requirement of not more than 1MB.

The paper is organised as follows: In Sec. 2, we summarise the theoretical background needed for our work. Sec. 3 describes the encoding of CTMCs and symbolic algorithms to compute steady-state probabilities. In Sec. 4, we provide examples and discuss various subtle points when deriving a symbolic representation from a high-level description. Sec. 5 concludes the paper.

## 2 Preliminaries

In this section, we briefly recall the essentials of continuous time Markov chains and the computation of their steady-state probabilities. Furthermore, we introduce multi-terminal BDDs and their crucial features.

**Continuous time Markov chains:** A CTMC is a tuple[1] $\mathcal{M} = (S, R)$ where $S$ is a finite set of *states*, and $R : S \times S \to \mathbb{R}_{\geq 0}$ is the *rate matrix* satisfying $R(s, s) = 0$ for all $s$. The CTMC is called *homogeneous* if the rates are time-independent. An infinitesimal generator matrix $Q$ is derived from $R$ by setting $Q(s, s') = R(s, s')$ for $s \neq s'$ and replacing the diagonal elements of $R$ by $Q(s, s) = -\sum_{s' \neq s} R(s, s')$.

Given a CTMC $\mathcal{M} = (S, R)$, one of the main goals is the computation of its steady-state probability vector $\boldsymbol{\pi} = \lim_{t \to \infty} \boldsymbol{\pi}(t)$. In this paper, we only consider finite, homogeneous, irreducible CTMCs. The steady-state probability vector $\boldsymbol{\pi} = (\pi_s)_{s \in S}$ is obtained by solving the linear system of equations

$$\boldsymbol{\pi} \cdot Q = 0$$

under the additional constraint that $\sum_{s \in S} \pi_s = 1$. This equation can be solved by direct methods such as Gaussian elimination. In order to obtain iterative solution schemes, the equation can also be transformed into the following common fixed point equation with appropriate iteration matrix $M$:

$$\boldsymbol{\pi} = \boldsymbol{\pi} \cdot M .$$

**Multi terminal binary decision diagrams:** MTBDDs are an extension of BDDs [Bry86], representing functions from a multidimensional Boolean domain to an arbitrary finite range $\mathcal{D}$. For instance, $\mathcal{D}$ can be a finite subset of the real numbers, or the set $\{0, 1\}$. In the latter case the MTBDD actually reduces to a BDD, representing a Boolean function. In the former case, the MTBDD represents a function of the type $f : \{0, 1\}^n \to \mathbb{R}$. The main idea behind the MTBDD representation of real-valued functions is the use of rooted directed acyclic graphs for a more compact representation of the *binary decision tree* which results from the *Shannon expansion*

$$f(\mathsf{v}_1, \ldots, \mathsf{v}_2) \;=\; \mathsf{if}\ \mathsf{v}_1\ \mathsf{then}\ f(1, \mathsf{v}_2, \ldots, \mathsf{v}_n)\ \mathsf{else}\ f(0, \mathsf{v}_2, \ldots, \mathsf{v}_n),$$

or, in terms of arithmetics, where $+$ and $\cdot$ denote ordinary addition and multi-

[1] Since we only consider steady-state behaviour for strongly connected chains, we do not fix a particular initial distribution.

| $r_1$ | $c_1$ | $r_2$ | $c_2$ | value |
|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 3 |
| 0 | 1 | 0 | 1 | 3 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 4 |
| 1 | 0 | 1 | 0 | 2 |
| 1 | 0 | 1 | 1 | 2 |
| *else* | | | | 0 |

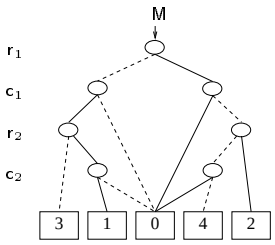



**Fig. 1.** MTBDD M, representing a function $f_{\mathsf{M}}\{0,1\}^4 \mapsto \{0,\ldots,4\}$

plication,

$$f(\mathsf{v}_1,\ldots,\mathsf{v}_2) \;=\; \mathsf{v}_1 \cdot f(1,\mathsf{v}_2,\ldots,\mathsf{v}_n) \;+\; (1-\mathsf{v}_1)\cdot f(0,\mathsf{v}_2,\ldots,\mathsf{v}_n).$$

**Definition 1.** *Let $\mathcal{D}$ be a finite set, and* Var *be a finite set of Boolean variables, equipped with a total ordering $\prec \subset \mathsf{Var}\times\mathsf{Var}$. A* multi-terminal binary decision diagram *(MTBDD) over $\langle \mathsf{Var},\prec\rangle$ is a rooted acyclic directed graph with vertex set $V$ and the following labelling: Each terminal vertex $v$ is labelled by an element of $\mathcal{D}$, denoted by $\mathsf{value}(v)$. Each non-terminal vertex $v$ is labelled by a variable $\mathsf{var}(v) \in \mathsf{Var}$ and has two children $\mathsf{then}(v)$, $\mathsf{else}(v) \in V$. In addition, the labelling of the non-terminal vertices by variables respect the given ordering $\prec$, i.e. $\mathsf{var}(\mathsf{then}(v)) \succ \mathsf{var}(v) \prec \mathsf{var}(\mathsf{else}(v))$ for all non-terminal vertices $v$.*

The edge from $v$ to $\mathsf{then}(v)$ represents the case where $\mathsf{var}(v)$ is true; conversely, the edge from $v$ to $\mathsf{else}(v)$ the case where $\mathsf{var}(v)$ false. A BDD is an MTBDD with $\mathsf{value}(v) \in \{0,1\}$ for all terminal vertices $v$. Each MTBDD M over $(\mathsf{v}_1,\ldots,\mathsf{v}_n)$ represents a function $f_{\mathsf{M}} : \{0,1\}^n \mapsto \mathcal{D}$ and has two *cofactors* $\mathsf{M}^{\mathsf{then}}$ and $\mathsf{M}^{\mathsf{else}}$, resulting from a top-level Shannon expansion, i.e. $\mathsf{M}^{\mathsf{then}}$ ($\mathsf{M}^{\mathsf{else}}$) represents $f_{\mathsf{M}}(1,\mathsf{v}_2,\ldots,\mathsf{v}_n)$ ($f_{\mathsf{M}}(0,\mathsf{v}_2,\ldots,\mathsf{v}_n)$), respectively.[2]

Fig. 1 shows a simple MTBDD M over $(\mathsf{r}_1,\mathsf{c}_1,\mathsf{r}_2,\mathsf{c}_2)$ together with the function $f_{\mathsf{M}}$ it represents. In the graphical representation, vertices are grouped into four levels, and all vertices on the same level are assumed to be labelled with the variable denoted on the left. Furthermore, we adopt the convention that edges from vertex $v$ to $\mathsf{then}(v)$ are drawn solid, while edges to $\mathsf{else}(v)$ are drawn dashed.

**Definition 2.** *A MTBDD* M *is called* reduced *iff*

- *for each non-terminal vertex $v$ the two children are distinct, i.e.* $\mathsf{then}(v) \neq \mathsf{else}(v)$. *Each terminal vertex $v$ has a distinct label* $\mathsf{value}(v)$.
- *for all vertices $v$, $v'$ with the same labelling, if the subgraphs with root $v$ and $v'$ respectively are isomorphic (i.e. coincide up to the names of the vertices) then $v = v'$. Formally: If* $\mathsf{var}(v) = \mathsf{var}(v')$ *and* $\mathsf{else}(v) = \mathsf{else}(v')$ *and* $\mathsf{then}(v) = \mathsf{then}(v')$, *then $v = v'$.*

[2] Note that an MTBDD over $\langle \mathsf{Var},\prec\rangle$ is also an MTBDD over $\langle \mathsf{Var}',\prec'\rangle$ for any superset $\mathsf{Var}'$ of Var and total ordering $\prec'$ on $\mathsf{Var}'$ such that $\mathsf{v}_1 \prec \mathsf{v}_2$ iff $\mathsf{v}_1 \prec' \mathsf{v}_2$ for all $\mathsf{v}_1$, $\mathsf{v}_2 \in \mathsf{Var}$. If $\mathsf{Var} = \{\mathsf{v}_1,\ldots,\mathsf{v}_n\}$ and $\mathsf{v}_1 \prec \mathsf{v}_2 \prec \ldots \prec \mathsf{v}_n$ then we also speak about MTBDDs over $(\mathsf{v}_1,\ldots,\mathsf{v}_n)$ rather than MTDDs over $\langle \mathsf{Var},\prec\rangle$.

For a fixed ordering of Boolean variables, reduced MTBDDs form a *canonical* representation of (real valued, in our setting) functions, i.e. if $\mathsf{M}$, $\mathsf{M}'$ are two reduced MTBDDs over the same ordered set $\mathsf{Var}$ such that $f_{\mathsf{M}} = f_{\mathsf{M}'}$, then $\mathsf{M}$ and $\mathsf{M}'$ are isomorphic. Bryant [Bry86] has proposed a recursive procedure to reduce BDDs that can be applied to MTBDDs as well. Unless otherwise stated, we work with MTBDDs that are reduced by convention.

MTBDD $\mathsf{M}$ of Fig. 1 satisfies Definition 2, it is *reduced*. Note that the valuations of some variable levels are irrelevant on certain paths through the MTBDD. For instance, for function $f_{\mathsf{M}}$ to return the values 3 or 2, the truth value of variable $\mathsf{c}_2$ is irrelevant. Hence the vertices on these paths are skipped, a consequence of the first clause of Definition 2. Variable $\mathsf{c}_2$ is called a *don't-care variable* for the respective paths.

## 3 Symbolic representation and analysis

In this section, we discuss how matrices (and hence CTMCs) can be encoded as MTBDDs and present iterative solution algorithms to compute steady-state probabilities that completely work on MTBDDs, extending work by [BFG+97]. The algorithms make use of a set of basic operations on MTBDDs taken from the literature, and described in Appendix A.

**Encoding of matrices:** A matrix whose entries are from a finite domain $\mathcal{D}$, say a finite subset of $\mathbb{R}$, can be represented by an MTBDD whose leaf vertices are labelled by elements from $\mathcal{D}$. For the moment we consider square matrices whose dimension is a power of two, $2^n$. In the case of general dimensions, the matrix will be padded with an appropriate number of rows and columns of zeroes. By construction, these additional entries do not contribute to the size of the MTBDD representing the matrix.

The basic idea behind the use of BDDs for representing non-stochastic systems is to use an *encoding* of the states by bit vectors $(\mathsf{v}_1, \ldots, \mathsf{v}_n)$ of some fixed length $n$ and to create a *symbolic representation* of the transition relation. The BDD corresponds to the Boolean function $f : \{0,1\}^{2n} \to \{0,1\}$ that is 1 if and only if a transition between two states exists. In the case of Markov chains, the rate matrix or transition probability matrix can be represented similarly, using the same basic idea.

As an example, consider the simple CTMC $\mathcal{M} = (\{0,1,2,3\}, R)$ (an example taken from [Ste94, p. 21]) with transition rate matrix $R$ as shown on the left of Fig. 2. Since the dimension of this matrix is $2^n = 4$, we need $n = 2$ bits for addressing its rows and 2 bits for addressing its colums. We use Boolean variables $\mathsf{r}_1, \mathsf{r}_2$, $(\mathsf{c}_1, \mathsf{c}_2)$ for encoding the row (column) position. Now, two crucial decisions have to be made, each of which influences the size of the MTBDD encoding matrix $R$. (1) It is necessary to define a total ordering on the Boolean variables involved, and (2) the state identifiers have to be mapped on bit vectors of lenght $n = 2$.

Concerning the first issue, it might seem natural to order all row bits on top (or below) of the column bits, grouping the bits together that belong together.

$$R = \begin{pmatrix} - & 4 & - & - \\ 3 & - & 3 & - \\ - & 2 & - & 2 \\ - & - & 1 & - \end{pmatrix}$$

| transition | $r_1$ | $c_1$ | $r_2$ | $c_2$ | value |
|---|---|---|---|---|---|
| $0 \xrightarrow{4} 1$ | 0 | 0 | 0 | 1 | 4 |
| $1 \xrightarrow{3} 0$ | 0 | 0 | 1 | 0 | 3 |
| $1 \xrightarrow{3} 2$ | 0 | 1 | 1 | 0 | 3 |
| $2 \xrightarrow{2} 1$ | 1 | 0 | 0 | 1 | 2 |
| $2 \xrightarrow{2} 3$ | 1 | 1 | 0 | 1 | 2 |
| $3 \xrightarrow{1} 2$ | 1 | 1 | 1 | 0 | 1 |
| | *else* | | | | 0 |

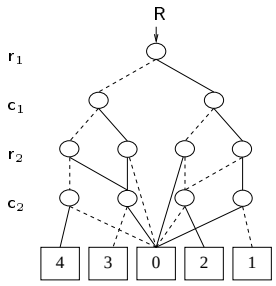



**Fig. 2.** Rate matrix $R$, transition encoding, and MTBDD $\mathsf{R}$ for a simple CTMC

However, experience has shown that in order to obtain compact symbolic representations, the Boolean variables should *not* be ordered in this way. Instead, it is recommended to choose an ordering such that variables specifying row and column positions are *interleaved* [EFT93]. This is the most prominent heuristics known in the context of BDDs. We will focus on heuristics in detail in Sec 4. For the moment we simply choose the following interleaved variable ordering:

$$\mathsf{r}_1 \prec \mathsf{c}_1 \prec \mathsf{r}_2 \prec \mathsf{c}_2$$

Concerning the second issue, we choose the straight-forward encoding for the set of states $\{0,1,2,3\}$, i.e. $0 \mapsto 00$, $1 \mapsto 01$, $2 \mapsto 10$, and $3 \mapsto 11$. Fig. 2 shows the resulting encoding of the transitions of the CTMC, and the (reduced) MTBDD $\mathsf{R}$ over $(\mathsf{r}_1, \mathsf{c}_1, \mathsf{r}_2, \mathsf{c}_2)$. In fact, this is not the smallest possible MTBDD representing $R$; a more efficient mapping from states to bit vectors exists. Using $0 \mapsto 10$, $1 \mapsto 00$, $2 \mapsto 11$, and $3 \mapsto 01$ instead results in a slightly smaller representation, namely the MTBDD $\mathsf{M}$ shown in Fig. 1. The latter MTBDD has only 12 instead of 16 vertices, basically because some *don't care variables* are skipped on paths from the $\mathsf{r}_2$ level to the terminal vertices labelled 3 and 2. Even in this simple example we can see the influence which the state encoding may have on the space efficiency of the symbolic representations.

The cofactors of an MTBDD representing a matrix correspond to submatrices. For instance, in our example $\mathsf{R}^{\mathsf{else}}$ yields the upper half and $\mathsf{R}^{\mathsf{then}}$ the lower half of matrix $R$, while $\mathsf{R}^{\mathsf{else}^{\mathsf{else}}}$ yields the upper left quarter of matrix $R$.

Since matrices derived from CTMCs are usually rather sparse, it is worth to briefly compare MTBDD representations with explicit sparse matrix data structures. As shown in [FMY97], the order of space requirements of MTBDDs is no worse than that of sparse matrix data structures, but it may (in the worst case) require a linear overhead. In the best case, MTBDD storage is by far superior. The *particular* strength of MTBDD lies in the ability to combine all matrix entries with the same numerical value in a single terminal vertex. This sharing is especially well-suited for CTMC applications, where usually just a handful of different rates are present, each of them occupying many entries in the generator. The crucial challenge is to encode the sparse matrix in such a way that sharing of terminal vertices is extended to sharing of subgraphs in the MTBDD. As we will see in Sec. 4, this can be achieved in particular for block-structured matrices with repetitive blocks.

**Matrix operations:** To give an impression of how basic logic and arithmetic operations are performed on MTBDDs, we discuss the realisation of matrix multiplication. All operations on (MT)BDDs follow the exemplified way. Details on other operations needed in this paper (such as APPLY) can be found in the appendix.

Consider two square matrices $M_1$ and $M_2$, represented as (non-reduced, for simplicity) MTBDDs $\mathsf{M}_1$ and $\mathsf{M}_2$ over variables $(\mathsf{r}_1, \mathsf{c}_1 \ldots, \mathsf{r}_n, \mathsf{c}_n)$ and $(\mathsf{c}_1, \mathsf{c}'_1 \ldots, \mathsf{c}_n, \mathsf{c}'_n)$. MMULT$(\mathsf{M}_1, \mathsf{M}_2)$ produces an MTBDD $\mathsf{M}$ over $(\mathsf{r}_1, \mathsf{c}'_1 \ldots, \mathsf{r}_n, \mathsf{c}'_n)$, representing the matrix product $M = M_1 \cdot M_2$. This MTBDD is generated by *recursive descent*. The four quarters of $M$ corresponding to the cofactors $\mathsf{M}^{\mathsf{else}\,\mathsf{else}}$, $\mathsf{M}^{\mathsf{else}\,\mathsf{then}}$, $\mathsf{M}^{\mathsf{then}\,\mathsf{else}}$, and $\mathsf{M}^{\mathsf{then}\,\mathsf{then}}$ are computed on the basis of the cofactors of $\mathsf{M}_1$ and $\mathsf{M}_2$. For instance:

$$\mathsf{M}^{\mathsf{else}\,\mathsf{else}} = \text{APPLY}\left(\text{MMULT}(\mathsf{M}_1^{\mathsf{else}\,\mathsf{else}}, \mathsf{M}_2^{\mathsf{else}\,\mathsf{else}}), \text{MMULT}(\mathsf{M}_1^{\mathsf{else}\,\mathsf{then}}, \mathsf{M}_2^{\mathsf{then}\,\mathsf{else}}), +\right)$$

is the MTBDD reformulation of the fact that the upper left quarter of $M_1 \cdot M_2$ equals the sum of (1) the product of the upper left quarters of $M_1$ and $M_2$, and (2) the product of the upper right quarter of $M_1$ and the lower left quarter of $M_2$. The products MMULT$(\mathsf{M}_1^{\mathsf{else}\,\mathsf{else}}, \mathsf{M}_2^{\mathsf{else}\,\mathsf{else}})$ and MMULT$(\mathsf{M}_1^{\mathsf{else}\,\mathsf{then}}, \mathsf{M}_2^{\mathsf{then}\,\mathsf{else}})$ are recursively computed in the same way. The recursion terminates when the operands of MMULT are terminal vertices $v_1$ and $v_2$, in which case a terminal vertex labelled by $\mathsf{value}(v_1) \cdot \mathsf{value}(v_2)$ is returned. Matrix-vector (and vector-matrix) multiplication MVMULT (VMMULT) is performed by the same strategy. If $\mathsf{M}_1$ is as above, and $\mathsf{P}$ over variables $(\mathsf{c}_1, \ldots, \mathsf{c}_n)$ represents a vector $\boldsymbol{p}$, then MVMULT$(\mathsf{M}_1, \mathsf{P})$ computes an MTBDD $\mathsf{Q}$ representing $\boldsymbol{q} = M_1 \cdot \boldsymbol{p}$ by means of the cofactors of its arguments. The upper half of the vector $\boldsymbol{q}$ is, for instance, obtained from

$$\mathsf{Q}^{\mathsf{else}} = \text{APPLY}\left(\text{MVMULT}(\mathsf{M}_1^{\mathsf{else}\,\mathsf{else}}, \mathsf{P}^{\mathsf{else}}), \text{MVMULT}(\mathsf{M}_1^{\mathsf{else}\,\mathsf{then}}, \mathsf{P}^{\mathsf{then}}), +\right).$$

In order to extend this scheme to the case of *reduced* MTBDDs, additional integer parameters have to be passed to function MMULT, MVMULT, and VMMULT, basically to take care of variable levels that are skipped (don't care variables) [FMY97]. Their implementation works on reduced MTBDDs and returns MTBDDs that are reduced (and hence canonical) by construction. This highly relies on the use of hash tables for bookkeeping of functions for which the MTBDDs are already computed, together with pointers to the vertices of the (already generated) MTBDDs. These tables make it possible to generate reduced MTBDDs without an explicit call of Bryant's reduction procedure. As a side result, the hashing strategy avoids that the same arithmetic or logic operation with the same arguments is performed twice. We refer the reader to [Bry86,BRB90,CMZ+97,BFG+97,FMY97] for elaborated expositions of these details.

The above matrix algorithms have the same time complexity as their conventional sparse-matrix counterparts, but some overhead (in computation time) is needed, for instance to keep MTBDDs reduced, which on the other hand allows sophisticated hashing techniques to avoid recalculations of intermediate results. Furthermore, it should be noted that the look-up of a matrix entry on MTBDDs is logarithmic in the dimension of the matrix.

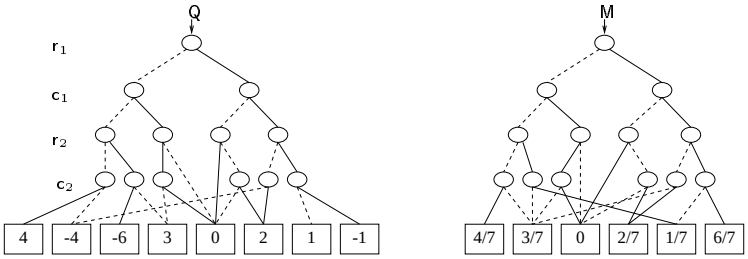



**Fig. 3.** MTBDD representation of matrices $Q$ and $M$ for the CTMC of Fig. 2

**Preparations for symbolic iteration:** Now we are ready to discuss symbolic (i.e. MTBDD-based) algorithms for calculating the steady-state distribution vector $\boldsymbol{\pi}$ of a CTMC, i.e. for the solution of the linear system of equations $\boldsymbol{\pi} \cdot Q = 0$ under the constraint $\boldsymbol{\pi} \cdot 1 = 1$. Direct methods such as Gaussian elimination have proved to be unsuitable for symbolic implementation [BFG+97, p. 201] because of the changing structure of the coefficient matrix caused by every elimination step. Modifications of the coefficient matrix are expensive in a symbolic setting, because canonicity of the MTBDD has to be maintained at every step. This is why we consider iterative methods where an iteration matrix $M$ is derived from the generator (or rate) matrix. Matrix $M$ remains unmodified during iteration which takes the form $\boldsymbol{\pi}^{(k+1)} := \boldsymbol{\pi}^{(k)} \cdot M$.

As a first step towards building the symbolic iteration matrix $\mathsf{M}$, the infinitesimal generator matrix $Q$ must be derived from the rate matrix $R$. Assuming that $f_{\mathsf{R}}$ over $(\mathsf{r}_1, \mathsf{c}_1, \ldots, \mathsf{r}_n, \mathsf{c}_n)$ represents matrix $R$, we compute the row sums of $R$ by setting:

$$\mathsf{D} := \text{MV}\textsc{mult}(\mathsf{R}, \mathbf{1})$$

where $\mathbf{1}$ is the reduced MTBDD (consisting of only a terminal vertex labelled 1) over $(\mathsf{c}_1, \ldots, \mathsf{c}_n)$ representing a vector of length $2^n$ with constant entries 1. From function $f_{\mathsf{D}}$ representing the vector of negative diagonal entries of $Q$, we can now construct the MTBDD $\mathsf{Q}$ (for the operator Diag, see the appendix)

$$\mathsf{Q} := \textsc{Apply}(\mathsf{R}, \textsc{Diag}(\mathsf{D}), -)$$

For the running CTMC example the MTBDD $\mathsf{Q}$ is shown on the left of Fig. 3.

**Power method:** In elementwise notation, the power method to iteratively compute the steady-state probabilities can be written as follows:

$$\pi_s^{(k+1)} = \pi_s^{(k)} + \textstyle\sum_{s' \in S} \pi_{s'}^{(k)} Q(s', s) \cdot \Delta t$$

which corresponds to the following matrix equation:

$$\boldsymbol{\pi}^{(k+1)} = \boldsymbol{\pi}^{(k)} \cdot (Q \cdot \Delta t + I)$$

where $I$ is the identity matrix of appropriate size. The scaling factor $\Delta t$ must be chosen such that $\Delta t < (\max_{s \in S} |Q(s,s)|)^{-1}$ in order to ensure that the iteration matrix $M = Q \cdot \Delta t + I$ is stochastic. A suitable value for $\Delta t$ can be chosen by means of $\textsc{Max}(\mathsf{D})$, since the MTBDD $\mathsf{D}$ is used to represent the row sums. In our running example, the biggest row sum is $f_{\mathsf{D}}(0,1) = 6$. Therefore, for sim-

```
algorithm IterativeSolve(M, n, diff_max)
(0)     P := 1/2^n
(1)     repeat
(2)         P' := VMmult(P, M)
(3)         P' := P'{c_1 ← r_1} ... {c_n ← r_n}
(4)         T  := Max(Apply(P, P', −))
(5)         P  := P'
(6)     until value(T) < diff_max
(7)     return P
```

**Table 1.** Symbolic iterative solution algorithm

plicity, we choose $\Delta t = 1/7$.[3] The actual scaling, i.e. calculating the product $Q \cdot \Delta t$ amounts to a simple scalar multiplication, afterwards the identity matrix (of dimension $2^n$) is added. Let $\mathsf{T}$ be an MTBDD consisting of a terminal vertex labelled $\Delta t$. Summarising the above steps, we define the iteration MTBDD

$$\mathsf{M} := \textsc{Apply}(\textsc{Smult}(\mathsf{Q}, \mathsf{T}), \mathsf{I}, +)$$

where $\mathsf{I} = \textsc{Diag}(\mathbf{1})$ is a BDD representing the identity matrix (actually a function $f_{\mathsf{I}} = \prod_{i=1}^{n}(\mathsf{c}_i \cdot \mathsf{r}_i + (1-\mathsf{c}_i)(1-\mathsf{r}_i))$ indicating that the row and column indices are identical.) Fig. 3 shows on the right the MTBDD $\mathsf{M}$ for the CTMC of Fig. 2. Having constructed the MTBDD representation of the iteration matrix $M$, two vectors are needed that will contain the state probability vectors $\boldsymbol{\pi}$, $\boldsymbol{\pi}'$ (of length $2^n$) before and after each iteration step. We represent them as MTBDD $\mathsf{P}$ and MTBDD $\mathsf{P}'$, over variables $(\mathsf{r}_1, \ldots, \mathsf{r}_n)$, and $(\mathsf{c}_1, \ldots, \mathsf{c}_n)$.

The symbolic iterative solution algorithm is given in Table 1. The algorithm has three parameters: iteration MTBDD $\mathsf{M}$, length of the state encoding $n$, and the maximal tolerated difference between successive iterates, $diff_{\max}$. To initialise the iteration, $\mathsf{P}$ is set to a reduced MTBDD consisting of a single terminal vertex labelled with $1/2^n$, i.e. all states are assumed to be equiprobable (alternatively, one could put probability 1 on a single state, or choose a known estimate as the initial probability vector). In line (2), the multiplication of the current estimate with the iteration matrix $M$ is performed. The variable renaming in line (3) basically transposes a column vector into a row vector (note that the ordering of variables respects the precondition required by renaming, cf. App. A). This is required in the subsequent subtraction in line (4), where the difference between the old and the new iterate is calculated, which is used in line (6) as a termination criterion. If the difference lies within the tolerated bound $diff_{\max}$ the algorithm terminates, returning the probability vector $\boldsymbol{\pi}$, represented as the MTBDD $\mathsf{P}$.

**Jacobi method:** In elementwise notation, the method of Jacobi can be written as follows:

$$\pi_s^{(k+1)} = -\sum_{\substack{s' \in S \\ s' \neq s}}^{n} \boldsymbol{\pi}_{s'}^{(k)} Q(s', s) \cdot \tfrac{1}{Q(s,s)}.$$

[3] Note that in practice $\Delta t$ should be chosen very close to $(\max_{s \in S} |Q(s,s)|)^{-1}$, for instance $(\max_{s \in S} |Q(s,s)|)^{-1} \cdot (1-\epsilon)$ for a small value of $\epsilon$, in order to achieve good convergence [Ste94, p. 31, 124]

Using $D$ to refer to the diagonal matrix of the row sums of $R$, the corresponding matrix formulation is

$$\boldsymbol{\pi}^{(k+1)} = \boldsymbol{\pi}^{(k)} \cdot R \cdot D^{-1}.$$

This means that the iteration matrix $M$ is defined as $M = R \cdot D^{-1}$. This form is suitable for symbolic implementation since inverting $D$ is trivial (componentwise inversion) and the product $R \cdot D^{-1}$ can be calculated by the matrix multiplication algorithm sketched in Sec. 3. More precisely, let $\mathsf{R}$ represent the rate matrix $R$ and $\mathsf{D}$ be the MTBDD encoding the row sums of $R$ (see above). Then we set

$$\mathsf{M} := \textsc{Mmult}(\mathsf{R}, \textsc{InvDiag}(\textsc{Diag}(\mathsf{D}))).$$

The MTBDD operation InvDiag is implemented by means of a single update of each terminal vertex $v$ of the argument MTBDD, replacing each non-zero $\mathsf{value}(v)$ by $\mathsf{value}(v)^{-1}$. For similar (time) efficiency reasons, an MTBDD-based algorithm for matrix multiplication can be devised for our special case where the right argument is a diagonal matrix. This multiplication simply amounts to column-wise scaling of the matrix $R$.[4] With this MTBDD $\mathsf{M}$, the iteration proceeds as in Table 1.

**Gauss-Seidel and Successive Overrelaxation:** The iteration scheme of Gauss-Seidel is similar to the method of Jacobi, but for computing the new iterate $\pi_{s_i}^{(k+1)}$ the already updated values for $\pi_{s_j}^{(k+1)}$, $j < i$ are used *immediately*, instead of at the next iteration. Since the specific strength of our symbolic computation is the simultaneous computation of common parts of a matrix-vector product, at first sight this strategy does not seem to be well-suited for symbolic implementation. But – for the sake of argument – suppose one wished to perform Gauss-Seidel by straight-forward matrix multiplication. One would have to explicitly calculate the iteration matrix $L \cdot (D - U)^{-1}$ where $L$ and $U$ are the lower (upper) triangular portions of $R$. A recursive MTBDD-based algorithm InvTri for inverting triangular matrices can easily be devised. The inversion causes a lot of fill-in, and therefore this approach is counterproductive in the setting of sparse matrices. However, in the MTBDD setting, the fill-in can be tolerated as long as many of the newly computed entries of $(D - U)^{-1}$ are identical, because this induces a sharing of subgraphs in the resulting MTBDD. For specific cases, such as the simple M/M/1 queue (cf. Sec. 4.3) with finite capacity $c$, this is the case indeed: Its matrix $D - U$ possesses $2 \cdot c + 1$ non-zeroes. The inverse has $(c + 1) \cdot (c + 2)/2$ non-zeroes (fill-in). However, the number of *distinct* non-zeroes is only $2 \cdot c$. Therefore, in specific cases, the straight-forward vector-matrix realisation of Gauss-Seidel might turn out to be rather efficient in a symbolic setting.

A symbolic version of the Successive Overrelaxation method (SOR) raises essentially the same issues as with Gauss-Seidel: Performing SOR by a simple matrix-vector multiplication scheme (which is not what is usually done) involves the inversion of a triangular matrix.

[4] A matrix column of $\mathsf{R}$ is addresssed with the help of restriction $\mathsf{R}\big|_{\mathsf{c}_1=b_1} \ldots \big|_{\mathsf{c}_1=b_n}$ of the $\mathsf{c}$ labelled vertices, where $b_1 \ldots b_n$ encodes the column index.

## 4 Compact encodings

In this section, we discuss a number of heuristics that are essential to get compact MTBDD representations for CTMCs derived from high-level formalisms. The heuristics address the ordering of Boolean variables, and the encoding of state identifiers as bit vectors. They are presented as *rules of thumb*, and discussed in the context of different high-level formalisms. The first, and most common rule of thumb has already been used in previous sections.

> *Rule of thumb 1:* It is recommended to use an *interleaved ordering* of the bit vectors $(\mathsf{r}_1, \ldots, \mathsf{r}_n)$ and $(\mathsf{c}_1, \ldots, \mathsf{c}_n)$ encoding row and column indices, i.e. to use $(\mathsf{r}_1, \mathsf{c}_1, \ldots, \mathsf{r}_n, \mathsf{c}_n)$ as the variable ordering for the Markov chain.

The benefits of this rule can be visualised by an inspection of the function $f_{\mathsf{I}} = \prod_{i=1}^{n} (\mathsf{c}_i \cdot \mathsf{r}_i + (1 - \mathsf{c}_i)(1 - \mathsf{r}_i))$ that can be interpreted as the identity matrix of dimension $2^n$. Using the variable ordering $(\mathsf{r}_1, \mathsf{c}_1, \ldots, \mathsf{r}_n, \mathsf{c}_n)$ leads to an MTBDD $\mathsf{I}$ with $3 \cdot n + 2$ vertices, whereas the naïve ordering $(\mathsf{r}_1, \ldots, \mathsf{r}_n, \mathsf{c}_1, \ldots, \mathsf{c}_n)$ blows up exponentially in $n$, it requires $3 \cdot 2^n - 1$ vertices. In our matrix setting, this phenomenon implies that everything that 'happens' on the main diagonal, or on a diagonal of some submatrix represented by a cofactor (or obtained by restriction of some arbitrary bits of the encoding) profits from this encoding. Furthermore, the interleaved ordering makes it possible to exploit structural information of high-level formalisms in the encoding. Clearly, if any kind of structural insight is used in the encoding of states, this insight will be reflected in the encoding of both row and column positions. Reasons why this is exploited best with the interleaved ordering will be given in the next sections.

### 4.1 Failure-repair models

In this section, we wish to exemplify that (and how) high-level structure can be turned into space-efficient, low-level MTBDD encodings. We do this by example, using a simple failure-repair model taken from [Ste94, p. 135]. The model describes two classes of subsystems, each class consisting of two identical components that are subject to failures and successive repairs. A component in class $i \in \{1, 2\}$ fails with rate $\lambda_i$ and is subsequently repaired with a rate $\mu_i$. Taking into account that equally behaving components can be lumped [KS76], we obtain the following CTMC $\mathcal{M} = (\{0, \ldots, 8\}, R)$ with transition rate matrix $R$ as follows.

$$R = \begin{pmatrix}
- & 2\lambda_2 & - & 2\lambda_1 & - & - & - & - & - \\
\mu_2 & - & \lambda_2 & - & 2\lambda_1 & - & - & - & - \\
- & 2\mu_2 & - & - & - & 2\lambda_1 & - & - & - \\
\mu_1 & - & - & - & 2\lambda_2 & - & \lambda_1 & - & - \\
- & \mu_1 & - & \mu_2 & - & \lambda_2 & - & \lambda_1 & - \\
- & - & \mu_1 & - & 2\mu_2 & - & - & - & \lambda_1 \\
- & - & - & 2\mu_1 & - & - & - & 2\lambda_2 & - \\
- & - & - & - & 2\mu_1 & - & \mu_2 & - & \lambda_2 \\
- & - & - & - & - & 2\mu_1 & - & 2\mu_2 & -
\end{pmatrix}$$

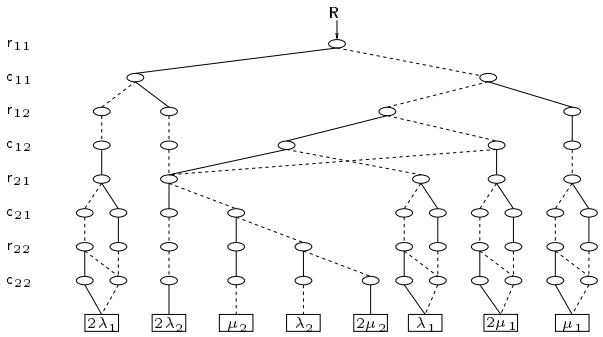



**Fig. 4.** MTBDD R for the structured encoding of the failure-repair model

The nine states can be encoded in the straight-forward way, mapping them onto bit vectors of length 4, such that $0 \mapsto 0000, \ldots, 8 \mapsto 1000$. Using interleaved variable ordering, the MTBDD R based on this encoding has 66 vertices in total. In fact, this straight-forward encoding of states turns out to be suboptimal.

> *Rule of thumb 2:* Compressing $m$ states into $\lceil \log_2 m \rceil$ bits does not necessarily lead to the most space efficient encoding. It is always more promising to exploit the structure of the high-level specification.

To illustrate this rule, we exploit the information that the above system consists of two classes, and that each class has two components that may fail independently. So we may view the states as tuples $(w_1, w_2)$, where each of the elements of this tuple ranges from 0 to 2 and indicates how many components of each class are currently running. We need two bits to encode each of the tuple elements, and choose to encode these elements in the straight-forward way. For instance, $(1, 2)$ (corresponding to state 3) is encoded as a bit vector 0110. The resulting MTBDD is depicted in Fig. 4. It has fewer vertices than the one obtained with the straight-forward encoding, 59 vertices in total. (Here, and in the sequel, we omit the terminal vertex 0 and its incident edges from the graphical representation.) The example does not completely cover the claim in *rule of thumb 2*, since both the naïve and the structured encoding require $\lceil \log_2 m \rceil = 4$ bits to encode the $m = 9$ states of the example. We will return to this issue later, in Sec. 4.2.

The next rule of thumb is quite an interesting one. In particular we are not aware of similar rules even in the non-stochastic setting[5].

> *Rule of thumb 3:* Established techniques to compress state spaces, such as lumping, can be counterproductive in the MTBDD setting, since structure gets lost.

[5] A corresponding rule in the non-stochastic setting would say that the application of bisimulation equivalence [Mil89] tends to be counterproductive in the BDD setting. This is our experience indeed, but we are not aware of any publication reporting this effect.

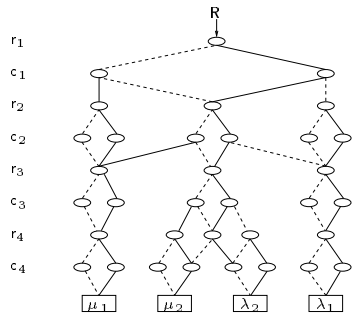



**Fig. 5.** MTBDD R for the non-lumped failure-repair model

To illustrate this rule, we consider the above example without applying lumpability beforehand. As a consequence, we now deal with a CTMC $\mathcal{M}' = (S', R')$ with nearly twice the number of states as before, namely $|S'| = 2^4 = 16$. The rate matrix $R'$ clearly contains more non-zero entries than the lumped variant, but only four different values ($\lambda_1$, $\lambda_2$, $\mu_1$, $\mu_2$) appear in this matrix. As a consequence, we can obtain an again more efficient MTBDD representation of this CTMC. It is depicted in Fig. 5, and has only 39 vertices, although the underlying state space is much bigger. The encoding used for this example is simple. Each of the four bits represents the status of one of the components. 1 is used to represent a running component, and 0 for a failed component, The first two bits represent components of class 1, the latter two class 2.

One may argue that a compression of the state space, even if it does increase the memory requirements, results in a reduction of the solution effort in time, since less computations have to be performed. However, since lumpable states will by the definition of lumpability (and our choice of the start vector in Table 1) be involved in the same arithmetic operations during the iterative solution, the solution effort is not substantially increased in the MTBDD setting, if lumpable states are kept distinct. The reason is that by means of efficient hashing (citing from [FMY97, p. 154]) "... virtually every operation has its result remembered for later re-use." Thus, the application of lumpability can be counterproductive, as it destroys the structure of the MTBDD without saving solution time.

### 4.2 Process algebras

Stochastic process algebras, or stochastic automata, are an excellent source for structure, and therefore well-suited to be used together with MTBDDs. [DB95] describes a general scheme to efficiently generate reduced BDDs from process algebraic descriptions. In particular, parallel composition on BDDs outperforms other ways to represent the composition, admittedly except *tensor-based* approaches: As a direct consequence of the interleaved variable ordering (*rule of thumb 1*) it is shown for non-stochastic process algebras in [EFT93] that the symbolic representation grows additive in the size of the representation of its components if the components are loosely coupled (i.e. have few synchronisation

points).[6] This result carries over from the non-stochastic to the CTMC case [HS99,Sie98]. The same additive growth is known from tensor approaches, but has to be contrasted to the usually observed multiplicative growth, due to interleaving of independent moves. Note that, as opposed to tensor approaches, every transition between two states is *explicitly* encoded in the MTBDD $\mathsf{R}$ by means of the corresponding $(\mathsf{r}_1, \mathsf{c}_1, \ldots, \mathsf{r}_n, \mathsf{c}_n)$ path leading to its rate. So, the look-up of a matrix entry can be done in (worst-case) logarithmic time in the size of the matrix (respectively additive in the size of the component MTBDDs), without involving arithmetic operations. The fact that symbolic parallel composition alleviates the explosion of the state space caused by interleaving can be intensified by applying the following rule.

> *Rule of thumb 4:* If the specification formalism has composition operators, it is wise to invest into an optimal encoding of the lowest level component state spaces. Avoid to touch the encodings that result after composition.

This rule proposes to optimise the component encodings, either by means of the exact algorithm[7] [THY93] or by means of adaptions of Rudell's sifting algorithm [Rud93] or other heuristic methods for BDDs, e.g. [FMK91,BMS95]. Furthermore, the above rule says that the structure gained by applying composition operators, such as parallel composition, should not be sacrificed, even though it might be tempting to shorten the bit vector encoding the states. This is essentially a re-statement of *rule of thumb 2*. In the context of parallel composition this rule implies that even though synchronisation constraints can induce that considerable parts of of the composed state space are actually unreachable, we propose *not* to construct an MTBDD over a Boolean space of a smaller dimension taking into account the reachability information. Our experience has shown that it is more efficient to keep the state encoding of the composed MTBDD unchanged, and to construct an additional BDD that encodes a reachability predicate. This BDD can be used to restrict the original MTBDD to the reachable subset of the state space, i.e. to remove all transitions originating from unreachable states. Reachability analysis can be realised efficiently on MTBDDs [BdS92]. It is in the nature of a *rule of thumb* that it is not always valid. In fact, for very strongly coupled systems, the additional computational burden imposed by the large number of unreachable states can justify to sacrifice the MTBDD structure.

### 4.3 Queueing networks

In this section, we are aiming to show the effect which (rather obvious) structurings of the state space encoding can have on the size of the MTBDD rep-

[6] In short, the condition that only a single component has moved requires a predicate in the style of the above function $f_|$, to check whether row and column indices of all the other components are identical, i.e. unchanged.

[7] Since the lowest level components are not likely to have large MTBDD representations, NP-completeness of the exact algorithm is not a problem in practice.

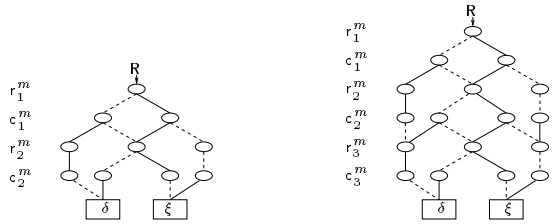



**Fig. 6.** MTBDD R of the M/M/1 example with queue capacity 3 and 7

resentation. We start with a simple M/M/1 queue with finite capacity $c$; for simplicitiy, we assume that $c = 2^k - 1$ for some natural $k$, since this ensures that the state space size is a power of two. Enumerating the states in the usual way from 0 to $2^k - 1$, we obtain that the rate matrix $R$ is non-zero at $R(i, i+1) = \delta$, $R(i+1, i) = \xi$, for $0 \leq i \leq 2^k - 2$, where $\delta$ is the arrival rate, and $\xi$ is the departure rate. For $k = 2$ we get

$$R = \begin{pmatrix} - & \delta & - & - \\ \xi & - & \delta & - \\ - & \xi & - & \delta \\ - & - & \xi & - \end{pmatrix}$$

For this queue, the MTBDD R over $(r_1^m, c_1^m, r_2^m, c_2^m)$ is shown in Fig. 6, together with the MTBDD for $k = 3$, i.e. for an M/M/1 queue with 7 places (the superscript '$m$' is used to distinguish the variables from other variables that are added later). The encoding of states as bit vectors is done in the 'natural' way, where 0 is encoded as a vector of zeroes, and $2^k - 1$ is encoded as a vector of ones (both of length $k$). The reader is invited to inspect the structure of these two decision diagrams. The crucial observation is that doubling the state space size (and hence essentially the queue capacity) does *not* double the memory requirements of the MTBDD needed to symbolically represent the rate matrix. In contrast, the MTBDD increases only linearly, by a constant of 7 non-terminal vertices. This is true in general, the M/M/1 with queue capacity $2^k - 1$ requires $7 \cdot k - 1$ vertices to be represented as an MTBDD. As far as we know, this feature is not present in any other method to store Markov chains explicitly.

> *Rule of thumb 5:* Structure exploitation is the key. If repetitive sub-blocks of a matrix are encoded 'close' to each other, the result can be an exponential saving.

Since the M/M/1 system is the simplest queueing system at all, its efficient encoding is not really a convincing argument to exemplify *rule of thumb 5*. We wish to highlight that the same effect, a logarithmic increase of the memory requirements in the size of the queue lengths, can be obtained for more complex queueing models as well. To illustrate this fact, we extend the above result to the broad class of single queues with phase-type arrival and service time distribution. Consider, for instance, an M/Cox$_p$/1 queue with finite capacity $c = 2^k - 1$

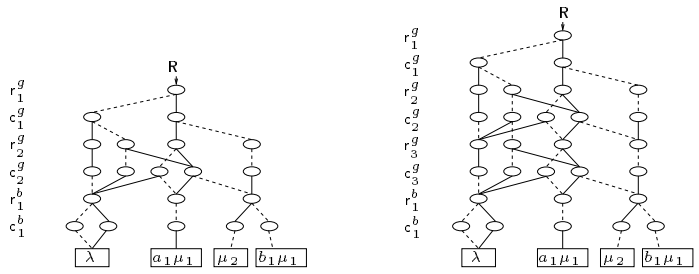


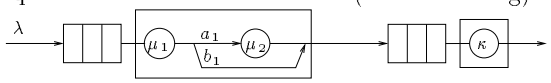

**Fig. 7.** MTBDD R of the M/$\text{Cox}_2$/1 example with queue capacity 3 and 7

and $p = 2^l$ phases.[8] Assuming $l = 1$ and $k = 2$, i.e. a queue capacity of $c = 3$ and a Coxian distribution with two phases, we end up with an example discussed in [Ste94, p. 237] (see there for the special treatment of the 'double' empty state). The rate matrix for this queue is given by

$$R = \left(\begin{array}{c|c|c|c} - & A & - & - \\ \hline B & C & A & - \\ \hline - & B & C & A \\ \hline - & - & B & C \end{array}\right), \text{ where } A = \begin{pmatrix} \lambda & - \\ - & \lambda \end{pmatrix}, B = \begin{pmatrix} b_1\mu_1 & - \\ \mu_2 & - \end{pmatrix}, \text{ and } C = \begin{pmatrix} - & a_1\mu_1 \\ - & - \end{pmatrix}.$$

Obviously, the matrix possesses a block tridiagonal structure, and again, for larger values of the queue capacity $c$, or the number of phases $p$ the matrix is simply extended in a regular fashion. In terms of $k$ and $l$, the matrix $R$ has dimension $2^{k+l}$, and hence it is quite natural to encode the global (diagonal) structure with $2k$ Boolean variables $(\mathsf{r}_1^g, \mathsf{c}_1^g, \ldots, \mathsf{r}_k^g, \mathsf{c}_k^g)$, and to represent the block matrices with $2l$ variables $(\mathsf{r}_1^b, \mathsf{c}_1^b, \ldots, \mathsf{r}_l^b, \mathsf{c}_l^b)$. For our example we get the MTBDDs shown in Fig. 7 (left). Note how the bottom variable levels $(\mathsf{r}_1^b, \mathsf{c}_1^b)$ directly encode the block matrices $A$, $B$ and $C$. In Fig. 7 we have also depicted the representation of the M/$\text{Cox}_2$/1 queue with capacity $c = 7 = 2^3 - 1$, in order to illustrate the logarithmic growth of the MTBDD. It turns out that the MTBDD corresponding to capacity $(2^k - 1)$ has $9 \cdot k + 7$ vertices, and a similar bound, linear in $l$, can be derived for an exponential increase of the number of phases of the Coxian distribution.

The same basic principle applies to any queue with block-structured generator matrix, and it is not restricted to single queues. If, for instance, we combine the two previous queues in a tandem network (with blocking) as shown below,




we can derive the corresponding MTBDD R from the operand's MTBDDs (using a variant of process algebra style parallel composition). The resulting MTBDD for the case where both queues have a finite capacity of $c = 7$ is shown in Fig. 8. The state space for this model is $16 \cdot 8 = 128$ states, and the MTBDD has 93 vertices. In general, if each queueing station has a capacity of $c = 2^k - 1$, the

[8] If the number of phases is not a power of 2, the same encoding applies, but some dummy rows and columns in the block matrices are filled with zero entries. The reduced MTBDD representation, however, has the same characteristics.

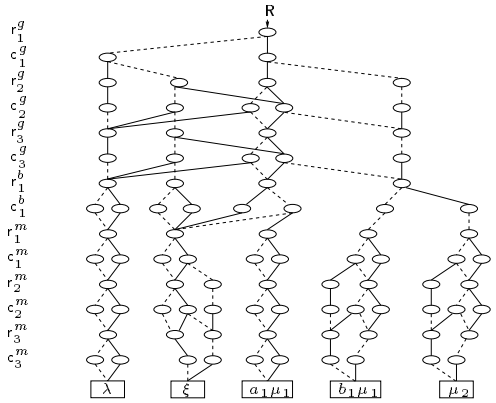



**Fig. 8.** MTBDD for the simple tandem queueing network

state space size is $2^{2k+1}$ while the MTBDD representation only requires $30 \cdot k + 3$ vertices. As a result, for exponentially growing state space, the growth of the MTBDD is linear.

To give an impression of the memory requirements of such a structure, assume that the memory requirement per MTBDD vertex is 12 byte (3 byte for the vertex identifier and the identifiers of the children, respectively, and an additional 3 byte for the labelling information). Note that with 3 byte $2^{24}$ vertices can be addressed. Using just 1 MB of memory, MTBDDs with up to 87381 vertices could be stored. In the tandem queueing network example we have seen that we needed $30 \cdot k + 3$ vertices to represent a state space of size $2^{2 \cdot k+1}$. For the 1 MB limit, the maximum value for $k$ is thus 2912, which corresponds to $2^{5825} = 10^{1753}$ states, generated by queue lengths of capacity $10^{877}$ for each of the two tandem queues.

## 5 Conclusion

In this paper we described symbolic representations of CTMCs based on multi-terminal binary decision diagrams. Adaptions of simple iterative schemes for the computation of the steady-state probabilities were discussed in this context, together with potential problems and benefits. We pointed out various crucial points concerning the encoding of state spaces stemming from different high-level formalisms and provided the reader with 5 useful *rules of thumb* which are intended as guidelines for the application of MTBDDs to CTMCs. These rules reflect our experience with symbolic representations in the context of Markov chains, and summarise the lessons we learned. In fact, the various success stories reported for (MT)BDDs usually neglect to highlight that their amazing space efficiency does not come for free. It is based on deep insight into the structure of the specification (formalism) at hand, together with heuristics to exploit them. Mechanised exploitation of such structure has been proposed for process algebra style specifications [DB95], but still some awkward phenomena can be observed,

for instance that state space compression (based on bisimulation or lumpability) can cause an increase of the size of the MTBDD (cf. *rule of thumb 3*). Nevertheless, we believe that it is promising to investigate symbolic representation of CTMCs much further, in particular because they have the *unique* potential of representing an exponential growth of the state space by means of an MTBDD that only grows linearly, as we have shown for the queueing systems discussed above. In the future, we are aiming to extend this result to broader classes of open networks with finite capacity and blocking.

Of course, a lot of work remains to be done concerning symbolic iterative solution of CTMCs. Although the prototypical implementation on which we are currently working is now capable of producing numerical results, it seems still too early to publish data on its performance. Among future work, we plan to investigate whether advanced numerical algorithms (e.g. projection techniques), of which it is known that they are superior to the simple schemes considered in this paper, have efficient symbolic variants.

**Acknowledgements:** We would like to thank Christel Baier for elucidating remarks concerning details of MTBDDs.

## References

[BCL91] J.R. Burch, E.M. Clarke, and D. Long. Symbolic Model Checking with partitioned transition relations. In *VLSI'91*, Edinburgh, Scotland, 1991.

[BCM+92] 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*, 98:142–170, 1992.

[BCMP75] F. Baskett, K.M. Chandy, R.R. Muntz, and F.G. Palacios. Open Closed and Mixed Networks of Queues with Different Classes of Customers. *Journal of the ACM*, 22(2):248–260, 1975.

[BdS92] A. Bouali and R. de Simone. Symbolic Bisimulation Minimisation. In *Proc. CAV'92*, pages 96–108, Springer LNCS 663, 1992.

[BFG+97] 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. *Formal Methods in System Design*, 10(2/3):171–206, 1997.

[BKLL95] H. Brinksma, J.-P. Katoen, R. Langerak, and D. Latella. A stochastic causality-based process algebra. *The Computer Journal*, 38(7):552–565, 1995.

[BMS95] J. Bern, C. Meinel, and A. Slobodova. Global rebuilding of OBDDs avoiding memory requirement maxima. In *Proc. CAV'95*, pages 4–15. Springer LNCS 939, 1995.

[BRB90] K.S. Brace, R.L. Rudell, and R.E. Bryant. Efficient Implementation of a BDD Package. In *27th ACM/IEEE Design Automation Conference*, pages 40–45, 1990.

[Bry86] R.E. Bryant. Graph-based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, 1986.

[BW96] B. Bollig and I. Wegener. Improving the Variable Ordering of OBDDs is NP-Complete. *IEEE Transactions on Computers*, 45(9):993-1006, 1996.

[Buc94] P. Buchholz. A class of hierarchical queueing networks and their analysis. *Queueing Systems*, 15:59–80, 1994.

[CDFH93] G. Chiola, C. Dutheillet, G. Franceschinis, S. Haddad. Stochastic Well-Formed Coloured Nets and Symmetric Modelling Applications. *IEEE Transactions on Computers*, 42(11):1343-1360, 1993.

[CS85] W.L. Cao and W.J. Stewart. Iterative Aggregation/Disaggregation Techniques for Nearly Uncoupled Markov Chains. *Journal of the ACM*, 32(3):702–719, 1985.

[CMZ+97] E.M. Clarke, K.L. Mcmillan, X. Zhao, M. Fujita, and J. Yang. Spectral Transforms for Large Boolean Functions with Applications to Technology Mapping. *Formal Methods in System Design*, 10(2/3):137–148, 1997.

[DB95] A. Dsouza and B. Bloom. Generating BDD models for process algebra terms. In *Proc. CAV'95*, pages 16–30. Springer LNCS 939, 1995.

[Don93] S. Donatelli. Superposed stochastic automata: a class of stochastic Petri nets with parallel solution and distributed state space. *Performance Evaluation*, 18(1):21–36, July 1993.

[EFT93] R. Enders, T. Filkorn, and D. Taubner. Generating BDDs for symbolic model checking in CCS. *Distributed Computing*, 6:155–164, 1993.

[FMK91] M. Fujita, Y. Matsunaga, and T. Kakadu. On variable ordering of binary decision diagrams for the application of multi-valued logic synthesis. In *Proc. EDAC'91*, pages 50–53, 1991.

[FM97] M. Fujita and P.C. McGeer, editors. Special Issue on Multi-Terminal Binary Decision Diagrams. *Formal Methods in System Design*, 10(2/3), 1997.

[FMY97] M. Fujita, P. McGeer, and J.C.-Y. Yang. Multi-terminal Binary Decision Diagrams: An efficient data structure for matrix representation. *Formal Methods in System Design*, 10(2/3):149–169, 1997.

[HMPS96] G.D. Hachtel, E. Macii, A. Pardo, and F. Somenzi. Markovian Analysis of Large Finite State Machines. *IEEE Transactions on CAD*, 15(12):1479–1493, 1996.

[HR98] H. Hermanns and M. Ribaudo. Exploiting Symmetries in Stochastic Process Algebras. In *Simulation – Past, Present and Future*, pages 763–770. SCS International, 1998.

[HS99] H. Hermanns and M. Siegle. Bisimulation Algorithms for Stochastic Process Algebras and their BDD-based Implementation. In *5th Int. AMAST Workshop on Real-Time and Probabilistic Systems*, Springer LNCS 1601, pages 244–264, J.P. Katoen, editor, 1999.

[KS76] J.G. Kemeny and J.L. Snell. *Finite Markov Chains*. Springer, 1976.

[LS92] Y.-T. Lai and S. Sastry. Edge-Valued Binary Decision Diagrams for Multi-Level Hierarchical Verification. In *29th Design Automation Conference*, pages 608–613. ACM/IEEE, 1992.

[Mil89] R. Milner. *Communication and Concurrency*. Prentice Hall, London, 1989.

[PA91] B. Plateau and K. Atif. Stochastic Automata Network for Modeling Parallel Systems. *IEEE Transactions on Software Engineering*, 17(10):1093–1108, 1991.

[Rud93] R. Rudell. Dynamic variable ordering for ordered binary decision diagrams. In *Proc. IEEE ICCAD'93*, pages 42–47, 1993.

[SA61] H.A. Simon and A. Ando. Aggregation of Variables in Dynamic Systems. *Econometrica*, 29:111–138, 1961.

[Sie94a] M. Siegle. Structured Markovian Performance Modelling with Automatic Symmetry Exploitation. In *Short Papers and Tool Descriptions of the 7th Int. Conf. on Modelling Techniques and Tools for Computer Performance Evaluation*, pages 77–81, Vienna, Austria, 1994.

[Sie94b] M. Siegle. Using Structured Modelling for Efficient Performance Prediction of Parallel Systems. In *Parallel Computing: Trends and Applications*, pages 453–460. North-Holland, 1994.

[Sie98] M. Siegle. Compact representation of large performability models based on extended BDDs. In *4th International Workshop on Performability Modeling of Computer and Communication Systems (PMCCS4)*, pages 77–80, Williamsburg, VA, 1998.

[Ste94] W.J. Stewart. *Introduction to the numerical solution of Markov chains*. Princeton University Press, 1994.

[THY93] S. Tani, K. Hamaguchi, and S. Yajima. The Complexity of Optimal Variable Ordering of a Shared Binary Decision Diagram. In *Proc. 4th ISAAC*, pages 389–398. Springer LNCS 762, 1993.

# A Operations on MTBDDs

In this appendix, we briefly sketch how the standard operators needed for our purposes can be realised on reduced MTBDDs. Let $\mathsf{M}$, $\mathsf{M}_1$, $\mathsf{M}_2$ be reduced MTBDDs over $(\mathsf{v}_1, \ldots, \mathsf{v}_n)$. In what follows, we write $v_1 \prec v_2$ if either $v_2$ is a terminal vertex while $v_1$ is non-terminal or both $v_1$, $v_2$ are non-terminal vertices and $\mathsf{var}(v_1) \prec \mathsf{var}(v_2)$.

**Combining two MTBDDs via binary arithmetic operators:** If OP is a binary operator (e.g. summation $+$ or multiplication $\cdot$) then APPLY$(\mathsf{M}_1, \mathsf{M}_2, \mathrm{OP})$ returns the MTBDD $\mathsf{M}$ over $(\mathsf{v}_1, \ldots, \mathsf{v}_n)$ where $f_{\mathsf{M}} = f_{\mathsf{M}_1}$ *op* $f_{\mathsf{M}_2}$. The basic idea behind this operator is as follows. APPLY$(\mathsf{M}_1, \mathsf{M}_2, \mathrm{OP})$ calls a recursive procedure $A_{\mathrm{OP}}(v_1, v_2)$ that takes a vertex $v_1$ of $\mathsf{M}_1$ and a vertex $v_2$ of $\mathsf{M}_2$ as its input and returns an MTBDDs with a root vertex $v$.

- If $v_1$ and $v_2$ are terminal vertices then $A_{\mathrm{OP}}(v_1, v_2)$ returns just the single terminal vertex $v$ labelled by $\mathsf{value}(v_1)$ OP $\mathsf{value}(v_2)$.
- If $v_1 \prec v_2$ then $\mathsf{var}(v) = \mathsf{var}(v_1)$, $\mathsf{else}(v) = A_{\mathrm{OP}}(\mathsf{else}(v_1), v_2)$ and $\mathsf{then}(v) = A_{\mathrm{OP}}(\mathsf{then}(v_1), v_2)$.
- Conversely, If $v_2 \prec v_1$ then $\mathsf{var}(v) = \mathsf{var}(v_2)$, $\mathsf{else}(v) = A_{\mathrm{OP}}(v_1, \mathsf{else}(v_2))$ and $\mathsf{then}(v) = A_{\mathrm{OP}}(v_1, \mathsf{then}(v_2))$.
- If $v_1$, $v_2$ are non-terminal vertices and $\mathsf{var}(v_1) = \mathsf{var}(v_2) = \mathsf{v}$ then $\mathsf{var}(v) = \mathsf{v}$, $\mathsf{else}(v) = A_{\mathrm{OP}}(\mathsf{else}(v_1), \mathsf{else}(v_2))$ and $\mathsf{then}(v) = A_{\mathrm{OP}}(\mathsf{then}(v_1), \mathsf{then}(v_2))$.

**Variable renaming:** Let $\mathsf{w} \notin \{\mathsf{v}_1, \ldots, \mathsf{v}_n\}$ and $i \in \{1, \ldots, n\}$ with $\mathsf{v}_{i-1} \prec \mathsf{w} \prec \mathsf{v}_{i+1}$. Then, $\mathsf{M}\{\mathsf{v}_i \leftarrow \mathsf{w}\}$ denotes the MTBDD over $(\mathsf{v}_1, \ldots, \mathsf{v}_{i-1}, \mathsf{w}, \mathsf{v}_{i+1}, \ldots, \mathsf{v}_n)$ that results from $\mathsf{M}$ where we change the variable labelling of any $\mathsf{v}_i$-labelled vertex into $\mathsf{w}$. For this, we put $\mathsf{var}(v) = \mathsf{w}$ for any $\mathsf{v}_i$-labelled vertex $v$ in $\mathsf{M}$.[9]

**Restriction:** Let $i \in \{1, \ldots, n\}$ and $b \in \{0, 1\}$. Then $\mathsf{M}\big|_{\mathsf{v}_i = b}$ denotes the MTBDD over $(\mathsf{v}_1, \ldots, \mathsf{v}_{i-1}, \mathsf{v}_{i+1}, \ldots, \mathsf{v}_n)$ that is obtained from $\mathsf{M}$ by replacing any edge from a vertex $v$ to a $\mathsf{v}_i$-labelled vertex $w$ by an edge from $v$ to $\mathsf{then}(w)$ if $b = 1$ ($\mathsf{else}(w)$ if $b = 0$), and by removing all $\mathsf{v}_i$-labelled vertices. Thus, for instance, $\mathsf{M}\big|_{\mathsf{v}_1 = 0}$ represents the partial function $(\mathsf{v}_2, \ldots, \mathsf{v}_n) \mapsto f_{\mathsf{M}}(0, \mathsf{v}_2, \ldots, \mathsf{v}_n)$, which is identical to the cofactor $\mathsf{M}^{\mathsf{else}}$.

**Scalar multiplication:** If $\mathsf{T}$ just consists of a terminal vertex $v'$ labelled with $k = \mathsf{value}(v')$, then SMULT$(\mathsf{M}, \mathsf{T})$ returns the unique reduced MTBDD over $(\mathsf{v}_1, \ldots, \mathsf{v}_n)$ representing the function $k \cdot f_{\mathsf{M}}$. Even though this operation can be easily realised using APPLY, it is more efficient to implement as an update of terminal vertices $v$ of $\mathsf{M}$, changing the value of each terminal vertex $v$ into $k \cdot \mathsf{value}(v)$.

**Maximum:** Let $\{v_1 \ldots v_h\}$ be the terminal vertices of $\mathsf{M}$. MAX$(\mathsf{M})$ is the MTBDD consisting of a single terminal vertex labelled with the $\max_{1 \le i \le h}(|\mathsf{value}(v_i)|)$, the absolute maximum of the function $f_{\mathsf{M}}$. This requires a simple traversal of the terminal vertices of $\mathsf{M}$.

**Matrix diagonal:** DIAG$(\mathsf{M})$ is the MTBDD over $(\mathsf{v}_1, \mathsf{v}'_1, \ldots, \mathsf{v}_n, \mathsf{v}'_n)$ representing $f_{\mathsf{M}}$ if $\mathsf{v}_i = \mathsf{v}'_i$ for $1 \le i \le n$ and 0 otherwise. So, it turns a vector into a diagonal matrix of the same dimension. It takes a vertex $v$ of $\mathsf{M}$ and introduces new vertices $v_0$ and $v_1$ with $\mathsf{var}(v_0) = \mathsf{var}(v_1) = \mathsf{var}(v)'$, $\mathsf{else}(v_0) = \mathsf{else}(v)$, $\mathsf{then}(v_1) = \mathsf{then}(v)$, and $\mathsf{then}(v_0) = \mathsf{else}(v_1) = 0$. Afterwards it sets $\mathsf{else}(v) = v_0$ and $\mathsf{then}(v) = v_1$, and (recursively) proceeds by taking the vertices $\mathsf{else}(v_0)$ and $\mathsf{then}(v_1)$.

[9] Note that $\mathsf{M}$ and $\mathsf{M}\{\mathsf{v}_i \leftarrow \mathsf{w}\}$ represent the same function (when viewed as MTBDDs over $(\mathsf{v}_1, \ldots, \mathsf{v}_{i-1}, \mathsf{v}_i, \mathsf{v}_{i+1}, \ldots, \mathsf{v}_n)$ and $(\mathsf{v}_1, \ldots, \mathsf{v}_{i-1}, \mathsf{w}, \mathsf{v}_{i+1}, \ldots, \mathsf{v}_n)$ respectively).