A symbolic multilevel method with sparse submatrix representation for memory-speed-tradeoff

Johann Schuster and Markus Siegle

Universität der Bundeswehr München, Institut für Technische Informatik
{johann.schuster,markus.siegle}@unibw.de

Abstract. This paper is about the numerical analysis of Markov chains, employing a multilevel algorithm for computing the vector of steady-state probabilities. As a basic data structure, multi-terminal binary decision diagrams (MTBDD) are used, which are known to provide very space-efficient symbolic representations of the rate matrix, even for very large Markov chains. In the approach presented here, the original Markov chain and several aggregated chains (used during the multilevel cycles) are stored within a single MTBDD. It is also discussed how the problem of non-contiguous encodings of the reachable states (of both the original and the aggregated chains) is dealt with by the concept of multi-offset-labelling. Furthermore, as the major innovation of the paper, a modification of the symbolic data structure is presented, in which parts of the MTBDD are replaced by a new type of enhanced sparse-matrix format, thereby speeding up access to the matrix elements during iteration. This new memory layout, specially tailored for multilevel algorithms, follows the recursive block structuring which is characteristic for both the multilevel algorithm and the MTBDD-based representation. The proposed data structure and algorithm are evaluated on the basis of three benchmark models. The empirical results exhibit considerable improvements in speed at very low memory cost, when compared to other methods.

Keywords: CTMC, numerical solution, aggregation, multilevel method, MTBDD, sparse matrices.

1 Introduction

Markov chains are an important and powerful mathematical concept for the model-based performance and dependability analysis of computer and communication systems. Usually some sort of high-level modelling formalism, such as stochastic Petri nets or stochastic process algebra, is employed for model specification, and the underlying Markov chain is generated automatically with the help of a tool which implements the semantics of the high-level model. The most important step during Markov chain analysis is the computation of the vector of stationary or transient state probabilities, from which most performance and dependability measures of interest can be derived by simple computations. A large range of numerical algorithms exists for computing these state probabilities: Simple iterative techniques such as Jacobi, Gauss-Seidel or their over-relaxed variants, block iterative methods such as block Gauss-Seidel, projection methods such as GMRES or BiCGStab, and methods based on aggre-

gation/disaggregation. Uniformisation is the method of choice for computing transient state probabilities. A classical overview of these methods is given in [18].

If the state space of the Markov chain is very large (which is often the case when modelling complex distributed systems with concurrent behaviour), its handling becomes problematic. Powerful approaches have been developed for generating and storing very large Markov chains in an efficient manner, of which a range of techniques which are based on compositional modelling formalisms and different forms of decision diagrams have proved to be particularly successful. These can be grouped into those approaches which are based on binary decision diagrams (BDD) and extensions thereof [12, 17, 8], and those approaches which are based on multi-valued decision diagrams and matrix diagrams [10, 11, 5]. A well-known feature of such symbolic representations is that state enumeration is non-contiguous, i.e. there exists a potential state space of which only a certain fraction is actually reachable. The reachable portion, however, can be determined efficiently with the help of symbolic reachability algorithms. While Markov chain generation and representation based on symbolic data structures is well understood, their numerical solution remains the major bottleneck and may even become infeasible in practice, due to memory and CPU time restrictions.

In this paper, we consider the computation of the vector of steady-state probabilities of Continuous Time Markov Chains (CTMC) which stem from a compositional modelling formalism and are represented in symbolic form using the multi-terminal BDD (MTBDD) data structure. For the numerical solution, we employ the so-called multilevel method, an iterative aggregation/disaggregation method which was inspired by multigrid methods, as proposed by Horton and Leutenegger [7], and even earlier considered by Schweitzer [16]. In our approach, the rate matrix of the original CTMC, as well as the rate matrices of all aggregated chains, are represented in a memory-efficient fashion within a single, augmented MTBDD. All probability vectors for the original and the aggregated systems are stored as arrays of the appropriate sizes, and the mapping between reachable states and their encoding in the potential state space is achieved with the help of an offset labelling of the MTBDD in the style of [12], but extended here to multiple offsets.

The contribution of this paper consists in the combination of the following three ingredients:

1. MTBDDs are used as a symbolic data structure for representing very large Markov chains in a compact way. In contrast to standard approaches, the basic MTBDD structure is extended in several ways: Instead of encoding only one Markov chain, multiple (aggregated) chains, together with their respective offset information, are represented by a single MTBDD.
2. As an efficient numerical solution method, the multilevel method is used, working directly on the symbolic Markov chain representation, thereby exploiting the natural recursive block structuring of the matrix as induced by the MTBDD.
3. For speeding up the traversal of the MTBDD, parts of it are replaced by a new type of sparse matrix data structures. In addition to our earlier approach described in [15], we do not only replace the very bottom parts of the MTBDD, but also blocks which are located in the interior of the MTBDD between any two aggregation levels, leading to improved runtimes.

1.1 Organisation of the paper

In Sec. 2, the multilevel solution algorithm is sketched. Sec. 3.1 gives a short introduction to the symbolic encoding of matrices, used for representing Markov chains. Sec. 3.2 recalls the multi-offset-labelling scheme, needed to keep track of different equation systems within one MTBDD, and Sec. 3.3 introduces the intermediate sparse matrices, used to speed up the symbolic multilevel algorithm. The applicability of the presented concepts is shown by some empirical results in Sec. 4. Sec. 5 concludes the paper.

2 The multilevel method

As an adaptation of multigrid techniques for the solution of Markov chains, the so-called multilevel algorithm has been presented in [7]. It is the idea of this algorithm to successively reduce large Markov chains to smaller ones. From the solutions of the smaller Markov chains, the current iteration vector of the original Markov chain is corrected.

For a given CTMC $\mathcal{M}$, let R denote its transition rate matrix and Q its generator matrix, respectively. Let π denote the vector of steady-state probabilities of the reachable states of $\mathcal{M}$. A certain level, i.e. a horizontal dotted line in Fig. 1, of the multilevel solution scheme is denoted by an integer $l \in \{0, \ldots, N\}$. The original system of linear equations (also called the *fine system*) will be labelled with level 0, i.e. $Q^{(0)} := Q$, $\pi^{(0)} := \pi$. The system of linear equations obtained by the aggregation of the system of level l is denoted by level $l+1$.

For a given Markov chain of level l, the aggregated chain of level $l+1$ is calculated as follows: Let $\mathcal{S}$ be the finite state space of the chain at level l, and $\tilde{\pi}^{(l),pre}$ be an approximation of the solution-vector of the steady-state equation $\pi^{(l)} \cdot Q^{(l)} = 0$ with $\sum_i \pi_i^{(l)} = 1$. Let the partition $\{\mathcal{S}_i | i \in \{0, \ldots, M-1\}\}$ of $\mathcal{S}$ define the next aggregation step. Then the initial probability vector $\tilde{\pi}^{(l+1),pre}$ of the aggregated Markov chain is given by the partial sums $\tilde{\pi}_i^{(l+1),pre} = \sum_{j \in S_i} \tilde{\pi}_j^{(l),pre}$. Using $\tilde{\pi}^{(l),pre}$, the $M \times M$ transition matrix of the ideal aggregate [14] is defined as follows:

$$Q_{ij}^{(l+1)} \overset{i \neq j}{=} \frac{\sum_{v \in S_i} \tilde{\pi}_v^{(l),pre} \cdot \sum_{w \in S_j} q_{vw}^{(l)}}{\sum_{v \in S_i} \tilde{\pi}_v^{(l),pre}},$$

i.e. consists of conditional probabilities of moving from one aggregate to another. As usual, the diagonal elements are given as negative row sums. After solving

$$\pi^{(l+1)} \cdot Q^{(l+1)} = 0 \text{ with } \sum_i \pi_i^{(l+1)} = 1$$

by $\pi^{(l+1),post}$, the correction

$$\tilde{\pi}_j^{(l),post} \overset{j \in S_i}{=} \frac{\pi_i^{(l+1),post}}{\tilde{\pi}_i^{(l+1),pre}} \cdot \tilde{\pi}_j^{(l),pre} \tag{1}$$

is applied, i.e. all the states belonging to a certain aggregate are scaled by the same factor. Instead of directly solving level $(l+1)$ by $\pi^{(l+1),post}$, it is a canonical extension to approximate the solution of level $(l+1)$ by $\tilde{\pi}^{(l+1),post}$ through further recursive

aggregation of level $(l+1)$. Then $\tilde{\pi}^{(l+1),post}$ is used instead of $\pi^{(l+1),post}$ in the disaggregation equation (1). The successive application of this approximation concept leads to the multilevel algorithm as indicated by a V-cycle in Fig. 1. On every level a certain number of *smoothing steps*, i.e. steps of an ordinary iterative solution algorithm, are performed.

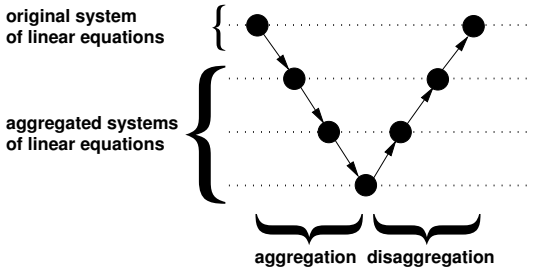


Fig. 1: Scheme of a multilevel V-cycle

3 MTBDD with sparse submatrix representation

3.1 Notation and MTBDD basics

Without loss of generality, it is assumed that all diagonal elements of the transition rate matrix R are zero (the row sums needed for the generator matrix Q are stored separately as a vector). The number of reachable states of the Markov chain is denoted by $S := |\mathcal{S}|$.

For the symbolic representation, symbols $\mathtt{s}_i$ and $\mathtt{t}_i$ $(1 \leq i \leq n)$ denote the Boolean variables encoding the source and target states of the transition rate matrix. Following a commonly accepted heuristics for obtaining small MTBDD sizes, these variables are ordered in an interleaved fashion, $\mathtt{s}_1 \prec \mathtt{t}_1 \prec \ldots \prec \mathtt{s}_n \prec \mathtt{t}_n$, and each pair $(\mathtt{s}_i, \mathtt{t}_i)$ is called a level of the MTBDD. Integers $a_1 > \ldots > a_N$ (where $1 \leq a_i \leq n$) denote the aggregation levels of the MTBDD, which means that a_i is the index of the s-variable that corresponds to the i-th aggregation level in the sense of Sec. 2. For an MTBDD node (or for a sparse submatrix), N_{below} denotes the number of aggregation levels below the node (or below the anchor position of the submatrix), not including the node itself.

Symbolic matrix encoding: An element r_{ij} of matrix R is represented in the MTBDD by a path from the root to a terminal node. The terminal node carries the real value r_{ij}, the s-variables on the path encode the source index i, and the t-variables encode the target index j.

As an example, consider the matrix R and its MTBDD representation as shown in Fig. 2 (this will be used as a running example throughout the paper). The matrix has $S = 7$ reachable states. However, the state indices to be encoded are from the set $\{0, 1, 2, 4, 5, 6, 8\}$, since reachable states are numbered non-contiguously, as is common in Markov chains stemming from compositional modelling formalisms. Since 8 is the highest index to be encoded, $n = 4$ s-variables and 4 t-variables are required. This means that the MTBDD in fact encodes a $2^4 \times 2^4$ matrix, of which in Fig. 2 (a) only the

top left portion is shown since all remaining entries are zero. As an example, consider the encoding of $r_{51} = 5$. Based on the binary encodings $5 = 0101_2$ and $1 = 0001_2$, the interleaved transition encoding is 00100011, which corresponds to the leftmost path of the MTBDD of Fig. 2 (b).

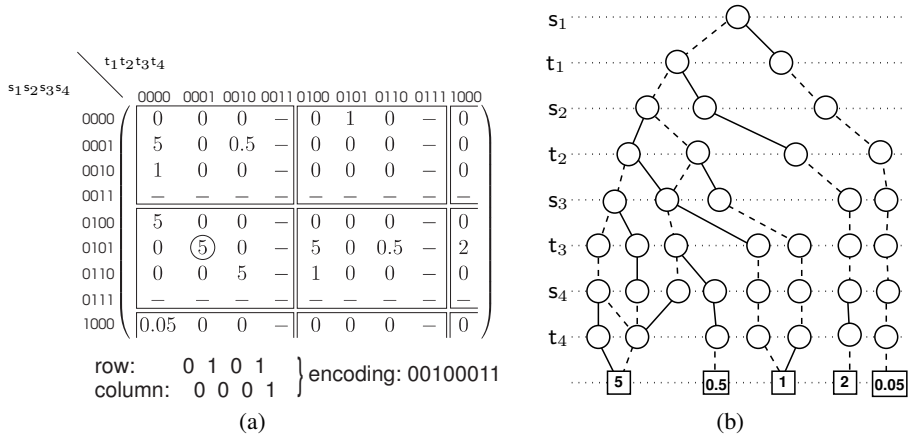


Fig. 2: (a) A transition matrix R and (b) its MTBDD representation

Recursive block structuring of the matrix: A well-known feature of MTBDD-based matrix representation is its recursive block structuring (which is of particular importance for the multilevel approach): In a first step, fixing the value of the $\mathtt{s}_1$-variable and of the $\mathtt{t}_1$-variable leads to a sub-MTBDD which represents one of the four quadrants of the original matrix (e.g., in Fig. 2, the setting $\mathtt{s}_1 = 0,\ \mathtt{t}_1 = 0$ leads to the top left 8×8 quadrant). In a second step, fixing the $\mathtt{s}_2$-variable and the $\mathtt{t}_2$-variable leads to a quadrant within the top-level quadrant, and so on. This phenomenon is of course due to the interleaved ordering of the s- and t-variables. Each block indicated in Fig. 2 (a) is addressed by a particular combination of the variables $\mathtt{s}_1,\ \mathtt{t}_1,\ \mathtt{s}_2,\ \mathtt{t}_2$. These blocks will be aggregated later on in the running example.

3.2 Multi-offset-labelling

The concept of multi-offset-labelling was introduced in [15], where also an algorithm for computing the multi-offsets was presented. For the symbolic multilevel method, the BDD reach (encoding the set of reachable states) and the MTBDD trans (representing the rate matrix of the original (fine) system) are extended in several ways:

BDD reach: This is an ordinary BDD (not an MTBDD) used to encode the set of reachable states. It depends on the Boolean variables $\mathtt{s}_1, \ldots, \mathtt{s}_n$. This BDD is augmented by offset information for the various (i.e. original and aggregated) systems. Each non-terminal node carries multiple offsets, i.e. a list of offset values, whose length is determined by the level of the node: A given node carries offset values for the fine

system and for all systems whose aggregation level lies below this node (which number equals $N_{below} + 1$).

Fig. 3 (a) shows the reachability structure of the original vector and of the aggregated vector for the running example (shaded fields denote unreachable states). Using s_3 as aggregation level, four states of the original system are aggregated into a single state. A state of the aggregated system is reachable, if and only if at least one of the corresponding states of the original system is reachable.

BDD reach, shown in Fig. 3 (b), is augmented by offset information for the fine and all aggregated systems, as will now be explained: Firstly, the complete BDD reach encodes the set of reachable states of the original system, i.e. the set $\{0, 1, 2, 4, 5, 6, 8\}$. This set needs to be mapped to the set of contiguous indices $\{0, 1, 2, 3, 4, 5, 6\}$ with the help of the offset values for the fine system, which are the leftmost values inside the BDD nodes. Offset values are additive, where a particular value is counted only on such paths where the corresponding node is left via the 1-valued edge. As an example, consider the example path 0110 in Fig. 3 (b). The corresponding potential state index $0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = 6$, is mapped to the offset $0+3+2+0 = 5$, as indicated in Fig. 3 (c), bottom. Secondly, regarding aggregation level s_3, variables s_1 and s_2 encode the reachable states of the aggregated system, which happens to be the set $\{0, 1, 2\}$. This set is mapped to the set of contiguous indices $\{0, 1, 2\}$ (i.e. in this case to itself!) with the help of the offset values for the aggregated system, which are the values on the right-hand side inside the doubly indexed BDD nodes. For the example path $01xx$ in Fig. 3 (b) the potential state index $0 \cdot 2^1 + 1 \cdot 2^0 = 1$ is mapped to the offset $0 + 1 = 1$, as indicated in Fig. 3 (c), top.

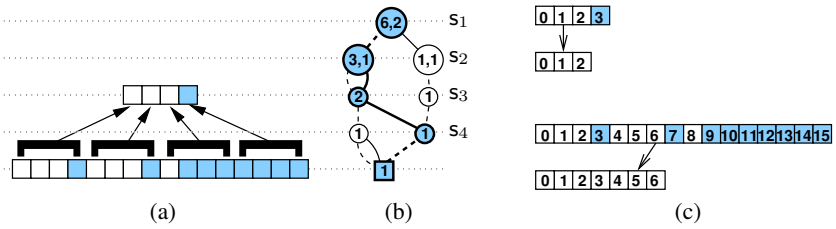
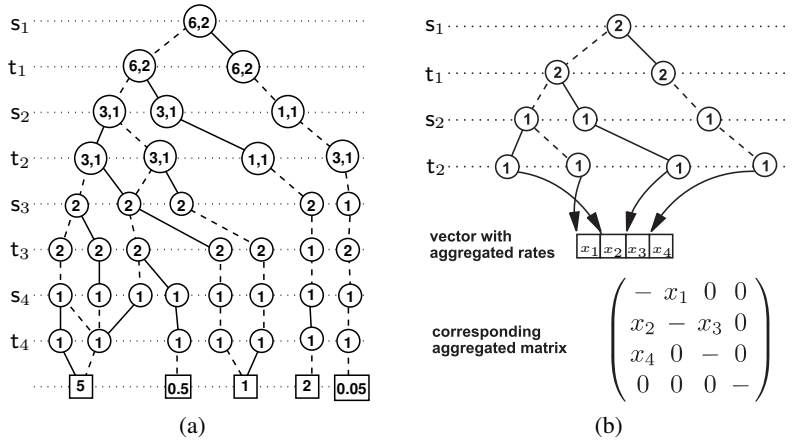


Fig. 3: (a) Reachability structure of original and aggregated vector for the running example, (b) corresponding multi-offset-labelled BDD and (c) mapping of potential to reachable state space

MTBDD trans: This symbolic data structure represents both the transition rate matrix of the fine system and of all aggregated systems. Rates of an aggregated system are stored within separate vectors in the aggregation levels.

Fig. 4 (a) illustrates this concept of multi-offest-labelling, referring again to the running example (the transition rates of the aggregated system are not shown in the figure). Offset values inside s-nodes contribute to the row index, whereas offset values inside t-nodes contribute to the column index. As an example for the fine system, consider again the binary encodings 0101 and 0001, which combine to the interleaved transition encoding 00100011. Looking at the offset values in the figure, this results in the row

offset $0+3+0+1=4$ and in the column offset $0+0+0+1=1$. As an example for the aggregated system, consider the binary encodings 01 and 00, which combine to the path 0010. This results in the row offset $0+1=1$ and in the column offset $0+0=0$. Fig. 4 (b) (top) shows how the aggregation information is stored within the MTBDD of Fig. 4 (a). For this illustration, variables $\{\mathtt{s}_3, \mathtt{s}_4, \mathtt{t}_3, \mathtt{t}_4\}$ as well as all the fine system offsets are omitted, as they are not used for the aggregated matrix. Again, diagonal elements of the matrix are stored separately. Pointers are used to keep track of the aggregated values. For the storage of the aggregation information, an iteration vector of the original system is reused, for details we refer to [15]. The aggregated system as a matrix is given in Fig. 4 (b), (bottom). As an example, for $\pi = (\pi_0, \pi_1, \pi_2, \pi_3, \pi_4, \pi_5, \pi_6)$ the aggregation process gives $x_2 = \frac{5\pi_3+5\pi_4+5\pi_5}{\pi_3+\pi_4+\pi_5}$.



(a) (b)

Fig. 4: (a) Multi-offset-labelled MTBDD for the running example, (b) representation of the aggregated system

3.3 Sparse matrix representation

In order to speed up the traversal of the symbolic data structure trans, parts of it (as much as memory allows) may be replaced by sparse matrix structures. Replacing sub-MTBDDs below the first aggregation level a_1 by sparse matrices is the most straightforward (following the replacement scheme originally described in [12]), and has already been described in [15]. However, the achievable gain in speed is quite limited, if only the lowest parts of the MTBDD can be replaced. Therefore, as an important contribution, this paper also describes the replacing of parts of the MTBDD that lie between two aggregation levels, which is more complicated, since offset information for more than one system has to be included in the sparse matrix representation. This paper does not consider sparse replacements of MTBDD-subgraphs that cross any aggregation level.

The following briefly recalls the principle of sparse matrix data structures [9, 18]: A common sparse representation for a square $S \times S$ matrix with NZ non-zero entries uses three arrays: Vals, Cols and RStart. The array Vals (of size NZ) contains all non-zero entries of the matrix, ordered by row. Array Cols is also of size NZ, its position i contains the column index of the corresponding entry in Vals. The array RStart of size S contains pointers into Vals and Cols, such that RStart$[i]$ denotes the beginning of row i.

In the context of the multilevel method, every non-zero value of a sparse matrix block must be associated with an array of row offsets and an array of column offsets. These arrays may not coincide, since for non-diagonal blocks the reachability structure for rows and columns may be different. Therefore, the sparse matrix storage scheme is extended by additional data structures, whose identifiers and roles are as follows:

- ROff is a list of arrays of row offsets. The length of the list equals the number of reachable rows in the current block. Each array is of length $N_{below} + 1$, i.e. the number of relevant aggregation levels.
- COff is a list of arrays of column offsets. Its length is the number of pairwise different combinations of column indices for the different relevant aggregated systems. Each array has the length $N_{below} + 1$.
- RStart is used in the standard way. It is an array of indices into the array ColsVals. Its length is the number of reachable rows in the current block.
- ColsVals combines the roles of the above mentioned Cols and Vals, but concerning the column index, there is one more degree of redirection than in the standard case. ColsVals is an array of index-pointer pairs. Its length is the number of non-zero entries in the current block. The index does not directly store the column index, it is an index into the array COff. The pointer is a pointer to a rate value or to the anchor node of a sub-block.

This general scheme is now explained by the running example (Fig. 4 (a)): In this example, MTBDD levels $\mathtt{s}_2$ and $\mathtt{t}_2$ are replaced by sparse multi-offset matrices (i.e. these matrices are of dimension 2×2). For these matrices we have $N_{below} = 1$ (since aggregation level $a_1 \sim \mathtt{s}_3$ lies below their anchor nodes). This means that the matrices have to contain offset information for two systems, the fine system and the first (and in this case only) aggregated system. For the discussion, we focus on the part of the MTBDD shown in Fig. 5 (a). The replacement of $\mathtt{s}_2$ and $\mathtt{t}_2$ is given in Fig. 5 (b) and reads as follows: ROff $= (0, 0); (3, 1)$ means that there are two non-zero rows. For the fine system, the row offsets are 0 and 3, whereas for the aggregated

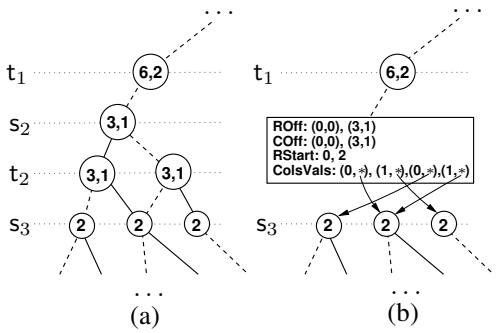


Fig. 5: Part of matrix substitution process for the running example

system the row offsets are 0 and 1. RStart $= 0, 2$ means that the first non-zero row contains 2 entries, and the second non-zero row contains the remaining (in this case also 2) entries. ColsVals $= (0, *), (1, *), (0, *), (1, *)$ means that the matrix contains 4 non-zero entries. The column offsets for a particular entry are determined by looking up the corresponding element of COff, i.e. for the value 0 the column offsets for the fine/aggregated system are 0/0, while for the value 1 the column offsets are 3/1. The actual entries can be found by following the corresponding arrows emanating from the $*$ symbol. In Fig. 6, the resulting overall data structure is shown.

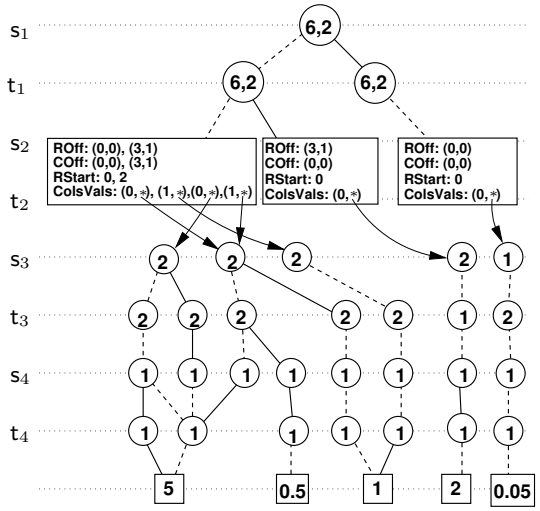


Fig. 6: Multi-offset-labelled MTBDD with sparse submatrices for the running example

4 Experimental results

In this section, we give examples of multilevel speedup due to the new sparse matrix approach. A comparison of runtimes will be given between the Jacobi Overrelaxation Algorithm, which is used as a smoother for the multilevel algorithm, the multilevel algorithm without intermediate sparse matrices and the new version with sparse matrices between some aggregation levels.

As suggested in [2], special emphasis is placed on the aggregation according to the submodel-structure of the high-level model specification from which the given Markov chain is derived. However, in contrast to [2], the MTBDD-based multilevel algorithm is not restricted to submodel-wise aggregation. This is very useful in cases where only a few submodels with small aggregated systems exist. Additional aggregation levels can be introduced in order to benefit from the finer structure of the aggregated matrix. Vice versa, for aggregation matrices which are too large, aggregation levels may be skipped or raised in order to obtain smaller aggregated systems.

All measurements presented here were performed on an Intel Xeon 3.0 GHz processor with 2 GByte of main memory under the Linux operating system. As a software platform for experimentation, we used the tool PRISM [13]. We implemented our multilevel algorithms (ML and the new MLS, see below) within the open source framework of PRISM. The following numerical algorithms are compared:

- *JOR*, Jacobi Overrelaxation, as provided by PRISM, with relaxation parameter 0.9.
- *ML* Multilevel algorithm using 4 pre- and 4 post-smoothing steps, respectively, on the fine system and 8 pre- and 8 post-smoothing steps, respectively, for the aggregated systems using V-cycles.
- *MLS* Multilevel algorithm using 4 pre- and 4 post-smoothing steps, respectively, on the fine system and 8 pre- and 8 post-smoothing steps, respectively, for the aggregated systems using V-cycles and intermediate sparse levels.

We did not consider Pseudo-Gauss-Seidel (*PGS*), as in [15] it is shown that for the FMS model JOR performs better than PGS and for the tandem queueing network only for the small scaling parameter PGS was able to outperform the ML algorithm.

For the multilevel smoothing steps of the fine system and all the aggregated systems, ordinary JOR steps with overrelaxation parameter 0.9 are used. For every multilevel experiment, on which we report, the aggregation levels for each aggregate will be given in parentheses, starting with the first and ending with the last aggregate. With the notation introduced in Sec. 3.1, an experiment is given by $ML(a_1, \ldots, a_N)$ or $MLS(a_1, \ldots, a_N)$, respectively.

Our ML and MLS implementations aim to achieve memory efficiency by using the second JOR iteration vector of the fine system for storing all aggregation information during the V-cycle [15]. The Jacobi Overrelaxation algorithm has been measured twice. Once with the PRISM standard setting which limits the memory for the sparse matrix structures to one megabyte, and the other that uses pure sparse matrices (all MTBDD levels are converted to sparse matrices).

The stopping criterion for all algorithms is a relative element-wise error smaller than $1.0 \cdot 10^{-6}$. For JOR this is measured between two consecutive iterations, whereas for the ML and MLS algorithms, convergence is tested during the post-smoothing step in the fine system.

Tables 1, 2 and 3 show the results of the experiments. In the first three columns the model characteristics are shown: The *scaling* parameter of the model (e.g. number of tokens for every machine in the FMS system, see 4.1), the number of reachable *states* and the number of *transitions* between the reachable states. Column *algorithm* specifies the numerical method used. Column *ML-cycles* shows the number of multilevel cycles until the convergence criterion is satisfied (which is only applicable to the ML and MLS algorithms). Column *steps* gives the number of iteration steps until convergence for JOR, whereas in the multilevel cases only the smoothing steps on the fine system are counted (to show the reduced smoothing effort for the fine system). In *variable levels* the number of `s`-variables is shown. The *sparse levels* column gives the number of `s`- (and `t`-) variables substituted by sparse matrices beginning from the bottom of the MTBDD. In column *residual* the maximum norm of the final vector $\tilde{\pi}Q$ is given. The column *memory* shows the total memory consumption of the different algorithms in kilobyte and finally the column *time* shows the consumed time (including both setup

phase of the data structure and time for the iterations) in seconds until convergence was achieved.

4.1 Flexible Manufacturing System (FMS)

This example is one of the standard benchmark case studies which are available from the PRISM web page [13] and is based on the model published in [3]. The model consists of three machines where one machine produces one certain part (denoted by part1), the second machine can produce two different parts (denoted by part2 and part3), and the third machine produces a new part (denoted by part12) out of part1 and part2 provided by the first and the second machine. The scaling parameter is the initial number of raw parts for each machine.

The current version of the multilevel algorithm uses a fixed ordering of the aggregations. For the experiments, the ordering of submodels is always (part2, part1, part12, part3) for the generation of the MTBDD. Therefore, the aggregation levels for scaling $5-7$ are (47,34,15), scalings 8 and 9 use (59,43,19).

The results in Table 1 show that the intermediate sparse matrix version (MLS) accelerates the symbolic multilevel algorithm (ML) by the factors (2.52, 2.05, 1.85, 1.76, 1.40). This is due to the 32 (scalings 5-7) or 40 (scaling 8 and 9) additionally replaced MTBDD levels. The MLS algorithm always outperforms the JOR algorithm, even in the case where all MTBDD variable levels were substituted by sparse matrices. The speedup factors from fully-sparse JOR to MLS are (1.52, 1.75, 2.02, 2.02, 1.88), which shows the superiority of the multilevel principle. The memory consumption of ML and MLS is not higher than the memory consumption of JOR. This is due to the fact, that in our implementation, aggregation information is stored within the second iteration vector of the JOR smoother of the fine system, therefore memory for storing the aggregated systems is "for free". Overall, the experiments show that the FMS model can be solved efficiently by the multilevel algorithm.

4.2 Tandem Queueing Network

This example is also one of the standard benchmark case studies which are available from the PRISM web page [13] and is based on the model published in [6]. The model consists of two queues, the first one is a $M/Cox_2/1$, the second one a $M/M/1$ queue, both of the same capacity. The scaling parameter is the capacity of the queues.

The PRISM specification consists of two submodels, namely the two queues. If only submodel-wise aggregation were applied, the multilevel algorithm would degenerate to a two-level algorithm. In this case, the aggregated matrix would be very small and the multilevel corrections could not efficiently contribute to the solution phase. Therefore, for every experiment with the tandem model, two aggregation levels were used.

The results in Table 2 show that the speedup from (ML) to the intermediate sparse matrix version (MLS) is (1.13, 1.08, 1.03, 1.03), Such low speedup had to be expected, as there are only four MTBDD levels additionally replaced by the MLS algorithm.

The total speedup of the MLS algorithm compared to the fully-sparse JOR algorithm is (1.10, 1.90, 2.11, 2.46). For scaling parameters 200 and 400, the aggregation

scaling	states	transitions	algorithm	ML-cycles	steps	variable levels	sparse levels	residual	memory (kB)	time (s)
5	152712	1111482	ML(47,34,15)	28	224	55	9	$3.3086 \cdot 10^{-11}$	4279.2	38.99
			MLS(47,34,15)	28	224	55	41	$3.3086 \cdot 10^{-11}$	4437,4	15.46
			JOR	-	996	55	27	$5.5554 \cdot 10^{-10}$	4789.8	34.87
			JOR	-	996	55	55	$5.5554 \cdot 10^{-10}$	8523.2	23.45
6	537768	4205670	ML(47,34,15)	31	248	55	9	$4.3858 \cdot 10^{-11}$	11985.5	123.46
			MLS(47,34,15)	31	248	55	41	$4.3858 \cdot 10^{-11}$	12285.0	60.21
			JOR	-	1189	55	24	$3.9511 \cdot 10^{-10}$	12586.2	147.74
			JOR	-	1189	55	55	$3.9511 \cdot 10^{-10}$	28554.3	105.66
7	1639440	13552968	ML(47,34,15)	34	272	55	9	$1.7333 \cdot 10^{-11}$	32838.8	362.97
			MLS(47,34,15)	34	272	55	41	$1.7333 \cdot 10^{-11}$	33365,9	195.71
			JOR	-	1385	55	22	$3.0842 \cdot 10^{-10}$	33017.4	528.45
			JOR	-	1385	55	55	$3.0842 \cdot 10^{-10}$	86777.4	394.76
8	4459455	38533968	ML(59,43,19)	37	296	70	12	$3.2604 \cdot 10^{-11}$	85696.0	1127.32
			MLS(59,43,19)	37	296	70	52	$3.2604 \cdot 10^{-11}$	86567.4	641.90
			JOR	-	1582	70	24	$2.4266 \cdot 10^{-10}$	85035.1	1759.28
			JOR	-	1582	70	70	$2.4266 \cdot 10^{-10}$	239493.2	1297.75
9	11058190	99075405	ML(59,43,19)	41	328	70	12	$1.3347 \cdot 10^{-11}$	205011.8	2813.31
			MLS(59,43,19)	41	328	70	52	$1.3347 \cdot 10^{-11}$	206017.9	2012.56
			JOR	-	1782	70	24	$1.9119 \cdot 10^{-10}$	204209.8	4834.75
			JOR	-	1782	70	70	$1.9119 \cdot 10^{-10}$	601256.2	3775.74

Table 1: Empirical results for the FMS model

information does not fit into the space provided by the second JOR iteration vector of the fine system, so slightly more memory is consumed than for the JOR with moderate sparse matrix level. For scaling parameters 600 and 800 the aggregation information fits in the second iteration vector, so the memory requirement is lower than for the JOR variants. In every case fully sparse JOR has the highest memory consumption.

4.3 Multi server multi queue model (MSMQ)

This third case study follows exactly the model published in [1]. A similar model had been used in the context of a multilevel algorithm for hierarchical Kronecker structures in [2]. It consists of five clients which are served by two servers in a round robin manner. We use the same parameter set as used in [2], that is, $\lambda = (0.075, 0.075, 0.225, 0.75, 1.2)$ for the arrival rates to the queues, service rate $\omega = 10.0$ and walk rate $\mu = 1.0$. Service and walk are of infinite server type, whereas the arrivals are of single server type.

The results are shown in Table 3, where the "scaling parameter" column has a special meaning: One configuration uses some extra MTBDD variables for the high-level structure, that is, the distribution of the two servers to the clients, as proposed in [2] (denoted by HLM). The other configuration uses only the synchronised queuing components without additional MTBDD variables for the high-level structure (denoted by STD). The ordering of submodels is (high-level, client1, client2, client3, client4, client5) for HLM and (client1, client2, client3, client4, client5) for STD. For all experiments we use submodel-wise aggregation of one or more submodels per aggregation step. The high-level submodel uses 10, all the clients use 7 MTBDD variables. For both

scaling	states	transitions	algorithm	ML-cycles	steps	variable levels	sparse levels	residual	memory (kB)	time (s)
200	80601	280599	ML(16,12)	170	1360	17	2	$3.4694 \cdot 10^{-18}$	2414.2	87.22
			MLS(16,12)	170	1360	17	6	$3.4694 \cdot 10^{-18}$	2417.4	76.99
			JOR	-	3670	17	2	$2.2204 \cdot 10^{-16}$	1426.4	114.4
			JOR	-	3670	17	17	$2.2204 \cdot 10^{-16}$	2601.1	84.36
400	321201	1121199	ML(17,13)	339	2712	19	3	$2.2204 \cdot 10^{-16}$	6366.9	476.17
			MLS(17,13)	339	2712	19	7	$2.2204 \cdot 10^{-16}$	6370.1	442.46
			JOR	-	7389	19	4	$5.5511 \cdot 10^{-17}$	5657.2	949.0
			JOR	-	7389	19	19	$5.5511 \cdot 10^{-17}$	10350.1	841.76
600	721801	2521799	ML(17,13)	695	5560	21	5	$1.1102 \cdot 10^{-16}$	12715.6	1463.42
			MLS(17,13)	695	5560	21	9	$1.1102 \cdot 10^{-16}$	12718.7	1421.25
			JOR	-	11130	21	15	$2.2204 \cdot 10^{-16}$	13517.5	3055.40
			JOR	-	11130	21	21	$2.2204 \cdot 10^{-16}$	23255.7	2998.17
800	1282401	4482399	ML(17,13)	842	6736	21	5	$6.9388 \cdot 10^{-18}$	22569.0	3064.44
			MLS(17,13)	842	6736	21	9	$6.9388 \cdot 10^{-18}$	22573.2	2979.68
			JOR	-	14880	21	15	$1.1102 \cdot 10^{-16}$	23443.7	7465.32
			JOR	-	14880	21	21	$1.1102 \cdot 10^{-16}$	41315.7	7343.33

Table 2: Empirical results for the Tandem model

STD and HLM, we experimented with different sets of aggregation levels, as indicated by the "algorithm" column of Table 3.

In the STD case, MLS accelerates ML by (1.63, 1.63). For HLM, the speedup of MLS over ML amounts to (1.85, 1.90, 1.79), as there are more levels to traverse. In both cases, when each submodel is aggregated separately, the memory consumption of the multilevel algorithms is about one megabyte higher than the memory consumption of JOR, as for this model the aggregation information cannot be "hidden" within the second iteration vector of the JOR smoother.

In our experiments, we found no parameter set where the multilevel algorithms performed better than the standard Jacobi Overrelaxation method. It is remarkable that the multilevel algorithms require fewer iterations than JOR (as expected), but due to the multilevel overhead, they still perform more slowly than JOR. However, the positive results published in [2] are probably due to the fact that there a cyclic or dynamic change of the multilevel aggregation ordering was employed, a feature which is not currently available with our own implementation.

The fewest multilevel cycles are required by the (39,25,11) aggregation level set for HLM, where only the high-level submodel is not aggregated in the coarsest system. It seems that this model is too small to apply the MTBDD-based multilevel algorithm successfully.

5 Conclusion and future work

This paper demonstrated how an improvement of the data structure can considerably speed up the numerical solution of Markov chains. Working with a symbolic MTBDD-based representation of the rate matrix and employing the well-known multilevel solution method, the new method which we presented consists of replacing parts of the

scaling	states	transitions	algorithm	ML-cycles	steps	variable levels	sparse levels	residual	memory (kB)	time (s)
HLM	358560	2135160	ML(39,32,25,18,11)	51	408	45	7	$1.9872 \cdot 10^{-9}$	8131.7	120.79
			MLS(39,32,25,18,11)	51	408	45	35	$1.9872 \cdot 10^{-9}$	8174,7	65.12
			ML(39,25,11)	44	352	45	7	$3.1295 \cdot 10^{-9}$	7671.8	99.52
			MLS(39,25,11)	44	352	45	35	$3.1295 \cdot 10^{-9}$	7845,4	52.37
			ML(39,25)	49	392	45	7	$2.2471 \cdot 10^{-9}$	7671.8	112.03
			MLS(39,25)	49	392	45	21	$2.2471 \cdot 10^{-9}$	7756,3	62.62
			JOR	-	663	45	26	$8.9223 \cdot 10^{-9}$	7162.5	33.05
			JOR	-	663	45	45	$8.9223 \cdot 10^{-9}$	15134.2	29.87
STD	358560	2135160	ML(29,22,15,8)	51	408	35	7	$2.0104 \cdot 10^{-9}$	7888.3	105.07
			MLS(29,22,15,8)	51	408	35	28	$2.0104 \cdot 10^{-9}$	7914.0	64.19
			ML(29,15)	48	384	35	7	$1.9999 \cdot 10^{-9}$	7544.2	95.98
			MLS(29,15)	48	384	35	21	$1.9999 \cdot 10^{-9}$	7620.8	59.05
			JOR	-	663	35	25	$8.9223 \cdot 10^{-9}$	6971.7	32.66
			JOR	-	663	35	35	$8.9223 \cdot 10^{-9}$	15047.2	30.09

Table 3: Empirical results for the MSMQ model

symbolic data structure by a specialised type of sparse matrices. Contrary to previous approaches, the new method does not only replace boundary, i.e. bottom-most or top-most parts of the MTBDD, but also interior parts, thereby speeding up the traversal of the graph-based data structure and thus leading to reduced runtimes.

We found that the new method works very well on most, but not on all examples. The degree of improvement depends, of course, on the portion of the MTBDD to be replaced by sparse structures. This is limited on the one hand by the available memory. On the other hand, since we only replace parts of the MTBDD that are located *between* two aggregation levels, the success of the approach depends on a careful combined choice of aggregation levels and levels to be replaced.

We are currently developing a parallelised version of the symbolic multilevel code, targeted at multicore architectures, which promises further speedup while being widely available on modern microprocessor architectures. Furthermore, in addition to building the multilevel algorithm on the basis of Jacobi (or its overrelaxed variant), we plan to also employ the pseudo-Gauss-Seidel scheme, which is known to converge faster in many cases. However, the memory overhead is expected to be larger in that case. Furthermore, it seems that some models (such as MSMQ) that have been investigated in the literature, require a dynamic variation of the multilevel cycles in order to exhibit good performance. Our current implementation is not yet capable of performing such dynamic cycles, but this issue is also on our agenda for future work.

Acknowledgements: The authors would like to thank Cristian Cioriia from the University of Craiova, Romania, who did part of the implementation work during his stay as an exchange student at the Universität der Bundeswehr München [4].

The authors would also like to thank Deutsche Forschungsgemeinschaft (DFG) which supported this work under grants SI 710/2 and SI 710/5.

References

1. M. Ajmone-Marsan, S. Donatelli, and F. Neri. GSPN models of Markovian multiserver multiqueue systems. *Performance Evaluation*, 11:227–240, May 1990.
2. P. Buchholz and T. Dayar. Comparison of Multilevel Methods for Kronecker-based Markovian Representations. *Computing*, 73(4):349–371, 2004.
3. G. Ciardo and K. Trivedi. A decomposition approach for stochastic reward net models. *Performance Evaluation*, 18(1):37–59, 1993.
4. C. Cioriia. Analysis of large Markov models based on symbolic data structures. Graduation project, Universität der Bundeswehr München, Institut für Technische Informatik, and University of Craiova, June 2007.
5. S. Derisavi, P. Kemper, and W. H. Sanders. Symbolic state-space exploration and numerical analysis of state-sharing composed models. In *Proc. of the 4th Int. Conf. on the Numerical Solution of Markov Chains (NSMC '03)*, pages 167–189, Urbana, IL, September 2003.
6. H. Hermanns, J. Meyer-Kayser, and M. Siegle. Multi terminal binary decision diagrams to represent and analyse continuous-time Markov chains. *Proc. 3rd Int. Workshop on the Num. Sol. of Markov Chains*, pages 188–207, 1999.
7. G. Horton and S. Leutenegger. A Multi-Level Solution Algorithm for Steady-State Markov Chains. *ACM Performance Evaluation Review*, 22(1):191–200, May 1994.
8. K. Lampka and M. Siegle. Analysis of Markov Reward Models using Zero-suppressed Multi-terminal BDDs. In *1st. Int. Conf. on Performance Evaluation Methodologies and Tools (Valuetools)*, Pisa, Italy, ACM press, ISBN 1-59593-504-5 (CD edition), 10 pages, 2006.
9. R. Mehmood. Serial disk-based analysis of large stochastic models. In C. Baier, B. Haverkort, H. Hermanns, J-P. Katoen, and M. Siegle, editors, *Validation of Stochastic Systems: A Guide to Current Research*, volume 2925 of *Lecture Notes in Computer Science (Tutorial Volume)*, pages 230–255. Springer, 2004.
10. A. Miner and G. Ciardo. Efficient reachability set generation and storage using decision diagrams. In H. Kleijn and S. Donatelli, editors, *Application and Theory of Petri Nets 1999*, pages 6–25, Williamsburg, VA, USA, 1999. Springer, LNCS 1639.
11. A. Miner and D. Parker. Symbolic representations and analysis of large probabilistic systems. In C. Baier, B. Haverkort, H. Hermanns, J-P. Katoen, and M. Siegle, editors, *Validation of Stochastic Systems: A Guide to Current Research*, volume 2925 of *Lecture Notes in Computer Science (Tutorial Volume)*, pages 296–338. Springer, 2004.
12. D. Parker. *Implementation of symbolic model checking for probabilistic systems*. PhD thesis, School of Computer Science, Faculty of Science, University of Birmingham, 2002.
13. PRISM website. `http://www.prismmodelchecker.org/`.
14. G. Rubino and B. Sericola. Sojourn Times in Finite Markov Processes. *Journal of Applied Probability*, 26(4):744–756, Dec 1989.
15. J. Schuster and M. Siegle. A Multilevel Algorithm based on Binary Decision Diagrams. In K. Al-Begain, A. Heindl, and M. Telek, editors, *14th Int. Conf. on Analytical and Stochastic Modelling Techniques and Applications (ASMTA'07)*, pages 129–136, June 2007.
16. P. Schweitzer. Aggregation Methods for Large Markov Chains. In G. Iazeolla, P. Courtois, and A. Hordijk, editors, *Math. Computer Performance and Reliability*. Elsevier, 1984.
17. M. Siegle. Advances in model representation. In L. de Alfaro and S. Gilmore, editors, *Process Algebra and Probabilistic Methods, Joint Int. Workshop PAPM-PROBMIV 2001*, pages 1–22. Springer, LNCS 2165, September 2001.
18. W.J. Stewart. *Introduction to the numerical solution of Markov chains*. Princeton University Press, 1994.