A Multilevel Algorithm based on Binary Decision Diagrams

Johann Schuster, Markus Siegle
Universität der Bundeswehr München
Institut für Technische Informatik
{Johann.Schuster, Markus.Siegle}@unibw.de

Abstract: In this paper, a new variant of the multilevel algorithm for computing the steady-state solution of a continuous-time Markov chain is proposed. The method is integrated into a symbolic framework, where the CTMC is represented in a compact way using multi-terminal binary decision diagrams (MTBDD). It is shown how to represent the original CTMC and several aggregated chains within the same decision diagram, where particular attention is devoted to the question of how to deal with unreachable states. Some preliminary empirical results are provided which indicate that the method has the potential to solve very large Markov chains in an efficient manner.

Keywords: Markov chain, numerical analysis, aggregation, multilevel algorithm, Binary Decision Diagram.

I. Introduction

Markov chains are very popular in the area of model-based performance and dependability evaluation of computer and communication systems. Today, several well-understood high-level modelling formalisms are available for specifying such models, and all phases of Markov chain generation and analysis are supported by powerful and user-friendly tools. State space explosion is an adverse phenomenon that occurs during the modelling of complex systems, especially of those consisting of several concurrent subsystems. In view of this situation, researchers have developed novel techniques for representing Markov chains with the help of decision diagram data structures, which make it possible to construct and manipulate Markov chains of immense size. However, even though the generation of billions of states is only a matter of seconds (using techniques such as the ones described in [5], [16]), calculating state probabilities by performing numerical analysis is still a serious bottleneck. In order to alleviate this bottleneck, this paper proposes a new version of the multilevel algorithm for calculating steady-state probabilities, which is based entirely on decision diagrams. Since data structures such as multi-terminal binary decision diagrams (MTBDD) [1], [9] offer a natural structuring of the transition matrix, they seem to be quite ideally suited for block aggregation methods.

A. Related work

Many different numerical methods exist for calculating the steady-state probability vector of a continuous-time Markov chain (CTMC). Apart from standard iterative methods such as Jacobi, Gauss-Seidel or SOR, methods based on the aggregation and disaggregation of the states of the Markov chain (so-called AD methods) were proposed already in the 1970ies. Among these "classical" AD methods are the method of Takahashi [24], the method of Courtois [8] (aimed at nearly completely decomposable Markov chains) and the method of Koury/McAllister/Stewart [13]. An overview of these techniques may be found in [23]. Recently, Bazan et al. [2] proposed a self-correcting aggregation technique, where several approximate first level aggregations are corrected by results from common second level aggregations.

In the 1990ies, Horton and Leutenegger developed a multilevel AD method, inspired by multigrid solvers which had turned out to be very effective for the solution of partial differential equations [12]. Buchholz proposed a multilevel solution method for very large structured Markov models whose generator matrix is represented compactly as a Kronecker expression [3]. This work was carried further in [4] where hierarchical Kronecker structures and different multigrid types were studied.

For Markov chains represented symbolically by binary decision diagrams, Parker was the first to develop numerical analysis techniques whose speed was competitive with those based on explicit data structures [20]. He introduced a hybrid approach, in which the matrix is represented symbolically while the vectors are stored as ordinary arrays, and showed how to best exploit the available memory by replacing parts of the decision diagram by explicit data structures (i.e. sparse matrices). Based on Parker's work, Mehmood developed a symbolic out-of-core method where only a small part of the vector resides in main memory while the remaining part is kept on disk [18]. Lampka et al. were able to speed up Parker's approach by employing a novel type of decision diagram [17].

To the best of our knowledge, multilevel analysis in connection with symbolic decision diagram data structures has not been investigated before.

B. Organisation

The rest of this paper is structured as follows: Sec. II provides the necessary background material on the multilevel method and on the symbolic representation of Markov chains. Sec. III, the core section of the paper, presents new data structures and algorithms which make it possible to integrate the multilevel aggregation approach smoothly into a symbolic framework. In Sec. IV, some empirical results are presented and discussed, and Sec. V concludes the paper.

II. Preliminaries

A. *The multilevel method*

Let Q be the generator matrix of an irreducible finite-state CTMC with n states. The unknown vector $\vec{\pi}$ of its steady-state probabilities is the unique solution of the linear system (further called *system*),

$$\vec{\pi} Q = \vec{0} \tag{1}$$

which satisfies the normalising condition $\sum_i \pi_i = 1$. The multilevel method is a method for solving the above equation by recursively aggregating the original system, thereby obtaining systems of smaller dimension. For the moment, only a single aggregation step is considered: Let the current (fine) state space with n states $i, j, \ldots$ be partitioned into N macro states $I, J, \ldots$, where $N < n$. Assuming that $\vec{\pi}^{(l)}$ is the current approximation to the steady-state vector at the fine level l, the transition rates of the aggregated (coarse) Markov chain at level $l-1$ are computed according to the following aggregation equation:

$$q_{IJ}^{(l-1)} = \frac{\sum_{i \in I} \pi_i^{(l)} \sum_{j \in J} q_{ij}^{(l)}}{\sum_{i \in I} \pi_i^{(l)}} \tag{2}$$

and the initial approximation to the steady-state vector of the coarse system is obtained by setting $\pi_I^{(l-1,in)} = \sum_{i \in I} \pi_i^{(l)}$. From $\vec{\pi}^{(l-1,in)}$ an improved approximation $\vec{\pi}^{(l-1,out)}$ is then computed by carrying out some smoothing iteration steps (of the Jacobi, Gauss-Seidel or SOR scheme, for example) at the coarse level and/or by possibly aggregating the Markov chain even further. In the following disaggregation step, the improved approximation on the coarse level is used to correct the previous approximation at the fine level. All micro states i belonging to the same macro state I are corrected by the same factor according to the disaggregation equation

$$\pi_i^{(l,new)} = \frac{\pi_I^{(l-1,out)}}{\pi_I^{(l-1,in)}} \pi_i^{(l,old)} \tag{3}$$

Fig. 1 shows the pseudo code of the basic multilevel algorithm. In line (2), if the lowest level has been reached the system is solved (approximately or exactly) without further recursion. Otherwise, in line (4), some smoothing steps are carried out before the aggregated system is computed in line (5). Line (6) contains the recursive call to the algorithm at the next lower (i.e. coarser) level. Its results are used in lines (7-8) to correct the solution at level l, which is followed by some further smoothing steps in line (9).

B. *MTBDD-based Markov chain representation*

This subsection introduces the MTBDD-based representation of CTMCs with the help of a simple example. For more detailed information on the use of MTBDDs for compactly encoding CMTCs see [20], [22], [19]. Consider the Markov chain $\mathcal{M}$ whose transition rate matrix $R_{\mathcal{M}}$ is shown in Fig. 2. This CTMC has four states (numbered 0,1,2,3) of which only three are actually reachable, i.e. state 2 is unreachable. Table I shows how the individual transitions of $\mathcal{M}$ can be encoded by bitstrings, where $\vec{\mathsf{s}}$ and $\vec{\mathsf{t}}$ are bit vectors (vectors of Boolean variables) of appropriate length $n_V = 2$. The s-variables encode the source state, and the t-variables encode the target state of a transition. Column headed "path" contains the shuffled concatenation of the s- and t-values, which corresponds to the variable ordering used in the decision diagram. Such an interleaved ordering is a commonly accepted heuristic which yields not necessarily optimal but provable compact decision diagrams [10]. The MTBDD M encoding CTMC $\mathcal{M}$ is shown in Fig. 3 (a). Its non-terminal nodes are labelled with Boolean variables from the ordered set $\{\mathsf{s}_1, \mathsf{t}_1, \mathsf{s}_2, \mathsf{t}_2\}$, and its terminal nodes carry the transition rates. Each path through the MTBDD, starting at the root and ending in a non-zero terminal node encodes a valid transition of $\mathcal{M}$, where dashed edges correspond to the Boolean value 0 and solid edges correspond to the Boolean value 1. Starting from a fixed node, the successor reached by the dashed line will be called the else-successor, the node reached by the solid line will be called the then-successor.

```
(1)  $ml(l, Q^{(l)}, \vec{\pi}^{(l,in)})$
(2)    if $l = 0$ then solve $\vec{\pi}^{(l,out)} Q^{(l)} = \vec{0}$
(3)    else
(4)      $\vec{\pi}^{(l,old)} := pre\_smoothing(\vec{\pi}^{(l,in)})$
(5)      compute $\vec{\pi}^{(l-1,in)}$ and $Q^{(l-1)}$ acc. to eq. (2)
(6)      $ml(l-1, Q^{(l-1)}, \vec{\pi}^{(l-1,in)})$
(7)      for all $I$
(8)        for all $i \in I$ compute $\pi_i^{(l,new)}$ acc. to eq. (3)
(9)      $\vec{\pi}^{(l,out)} := post\_smoothing(\vec{\pi}^{(l,new)})$
(10)   return
```

Fig. 1. The multilevel algorithm

$$R_{\mathcal{M}} = \begin{pmatrix} 0 & \lambda & 0 & \lambda \\ 0 & 0 & 0 & \nu \\ 0 & 0 & 0 & 0 \\ \mu & \nu & 0 & 0 \end{pmatrix}$$

Fig. 2. An example Markov chain

$\vec{\mathsf{s}}$	$\vec{\mathsf{t}}$	path	rate
00	01	0001	λ
00	11	0101	λ
01	11	0111	ν
11	00	1010	μ
11	01	1011	ν

TABLE I: Binary encoding of CTMC $\mathcal{M}$

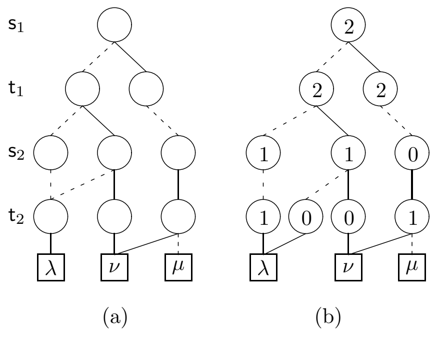

Fig. 3. MTBDD representation of CTMC $\mathcal{M}$ from Fig. 2

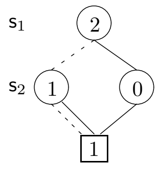

Fig. 4. Offset-labelled BDD *reach* representing the reachability set of $\mathcal{M}$ from Fig. 2

B.1 Offset-labelling

When generating a Markov chain from a high-level modelling formalism, thereby encoding states as bitstrings of length n_V, it is very common that certain such patterns correspond to unreachable states. In fact, symbolic algorithms are often employed in order to determine the set of reachable states, which may be only a very small subset of the so-called potential state space. Tools such as PRISM [15], [21], CASPA [14] and the Moebius pZDD-engine [17] contain such symbolic reachability schemes. During numerical analysis, in order not to waste memory space, the iteration vectors should be stored as arrays whose size corresponds to the number of reachable states n_r (not the number of potential states!). Therefore it is necessary to construct a mapping which maps bitstrings from the potential state space $\{0, 1, \ldots, 2^{n_V} - 1\}$ to the dense set $\{0, 1, \ldots, n_r - 1\}$. An order-preserving such mapping can be achieved by enriching the MTBDD with offsets which may be computed by an algorithm described in [20]. More precisely, first a BDD *reach* encoding the set of reachable states is computed and labelled by offsets. Using those offsets, the MTBDD $\mathcal{M}$ encoding the transitions of the CTMC is then enhanced by offsets in a double fashion, i.e. both for the s- and for the t-variables. Returning to the example, the offset-labelled BDD *reach* representing the reachability set of $\mathcal{M}$ is shown in Fig. 4. From this BDD, it is possible, for example, to calculate the index of state 11 (binary representation) as $2+0 = 2$. The offset-labelled version of MTBDD M is shown in Fig. 3 (b). (The t_2-labelled node at the bottom left has to be duplicated in this example since it can be reached via two different paths with different t_2-offsets.) During traversal of a path of the offset-labelled MTBDD, the index of the source state of the encoded transition may be determined by adding all offsets of s-nodes which are left via a 1-valued (solid) edge. Likewise, the index of the target state may be determined by adding all offsets of t-nodes which are left via a 1-valued edge.

III. Aggregation of the DD

In the following, the data structures and algorithms for the symbolic multilevel algorithm are presented. It will turn out that all the aggregated matrices can be added to the MTBDD without having to change the basic structure of the diagram, which will result in a small time and memory overhead for the aggregation process. Below, the term *fine system* will be used for the given, not aggregated system, while *coarse system* will mean an aggregated system. Furthermore, let the set of reachable states of the Markov chain be stored in a BDD *reach.* Usually in a BDD nodes with identical then- and else-successors are eliminated, but for compatibility with the given algorithms it is assumed that in the paths leading to the terminal 1-node of *reach* all don't care nodes are inserted explicitly. This is also done in [20] in order to store offset information in these nodes.

For an efficient implementation of the aggregation procedure in an MTBDD environment it is necessary to use macro states which combine 2^i neighbouring states of the potential state space. In terms of BDD *reach* this means that for one aggregation step i variables are aggregated, starting at the leaves of the graph.

To be consistent with the ordinary multilevel algorithm we use the following convention: With the notations of section II, fix a number of levels l to get a hierarchy of the fine system and l aggregated systems. Each level $i \in \{l-1, \ldots, 0\}$ in the multilevel algorithm is assigned to a variable level, the level l corresponds to the terminal nodes of BDD *reach.* In order to avoid confusion, in the following, aggregation levels referencing the algorithm in Fig. 1 will be called *aggregation level* or *agg_level*, while the term *aggregation variable level* or *agg_var_level* will be used for aggregation variable levels in BDD *reach.*

The example given in Fig. 5 (a) indicates that the mapping from aggregation levels to aggregation variable levels is well defined: Aggregation level 2 corresponds to the fine system, i.e. the constant level of BDD *reach.* The aggregation level 1 reduces the fine system by removing variable s_3, aggregation level 0 takes the result of aggregation 1 and removes variable s_2.

As the assignment above is in fact a bijection, both aggregation levels and aggregation variable levels could be used to describe the symbolic multilevel algorithm. For simplicity, the algorithms will always be described using the aggregation variable levels.

Aggregating a MTBDD according to its variables is not perfectly symmetric. It depends on how many reachable states are within one partition of the potential state space. Aggregating only unreachable states will lead to an unreachable state in the aggregate. Fig. 5 (b) sketches the aggregation idea using the BDD *reach* from Fig. 5 (a). The arrows indicate how the elements in the probability vectors are summed up in the two aggregation steps. Unreachable states are crossed

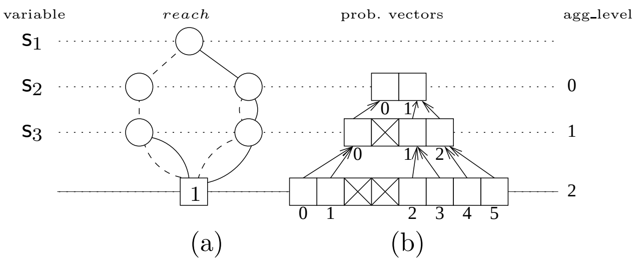

Fig. 5. Aggregation process

out. The numbers under the vectors are the indices of the elements in the vector of reachable states. The aggregation scheme is given by the pattern of reachable and unreachable states.

The following subsections provide the necessary extensions to the MTBDD structure and the corresponding algorithms for performing multilevel cycles.

A. Multi-offset labelling

For the aggregations of the solution vector and the iteration matrix it is necessary to provide BDD *reach* with offset information of the fine and all aggregated systems.

The offset labelling algorithm in Fig. 6 is basically the same as the one given in [20]. The constant ZERO denotes the terminal zero node, the constant FINAL is the variable level where the recursion shall bottom out. The algorithm is identical to the one in [20] when setting FINAL to the level for the constants in the MTBDD. Otherwise, when FINAL is set to an aggregation level, nontrivial subgraphs (i.e. subgraphs finally leading to the terminal 1 node) starting with nodes in the FINAL level play the role of the terminal 1 node.

Note that once the algorithm has finished for a certain FINAL level, for further calculations only the else-offsets and the total number of reachable states are significant. So after a reset the then-offsets may be reused for calculating the offsets of the next system. The offset nodes provide arrays for the else-offsets and one entry for a then-offset.

Depending on the number of systems one MTBDD variable level belongs to, different numbers of offsets have to be stored. For example, the root node always has $l+1$ offsets, while the nodes up to the variable level corresponding to the first aggregation level always carry the offsets of the fine system only.

For the example BDD *reach* in Fig. 5 (a) the multi-offset labelling is shown in Fig. 7. The nodes for s_1 carry from left to right offsets for the fine level, aggregation level 1 and 0, while nodes for s_2 only have offsets for the fine level and aggregation level 1, and so on.

B. Aggregating a vector

Once the BDD *reach* is multi-offset labelled, the aggregation of an iteration vector can be performed with the algorithm shown in Fig. 8. As the given algorithm only sums up the probability masses of single states belonging to the same aggregate, the coarse vector has to

```
(1) label(node, $\mathsf{s}_i$)
(2)   if (node=ZERO)
(3)     return 0
(4)   if ($\mathsf{s}_i$=FINAL)
(5)     return 1
(6)   if (node's offsets not calculated)
(7)     node.then_off := label(node.then, $\mathsf{s}_{i+1}$)
(8)     node.else_off[system] := label(node.else, $\mathsf{s}_{i+1}$)
(9)   return node.then_off+node.else_off[system]
```

Fig. 6. Offset labelling algorithm

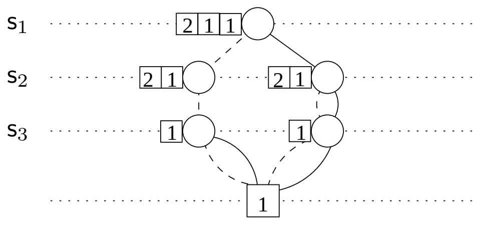

Fig. 7. Multi-offset labelling

be initialised with zeroes.

Let S be the variable level of the system to be aggregated and D the variable level of the aggregated system. Set i_S and i_D to the corresponding indices in the offset arrays. Note that $i_D = i_S + 1$ as only neighbouring levels can be aggregated.

The invocation of agg_vector(root(reach),0,0,s_1) performs the aggregation. The algorithm basically calculates the correct offsets for the summations. Until the D level is reached, in lines (7-10) both the fine and the coarse offsets are modified. Between D and S level in lines (11-13) only the fine offset is modified. When reaching the S level, lines (4-6) perform the summation according to the calculated offsets. In lines (2-3) unreachable subgraphs are skipped, as they either remain unreachable after aggregation or do not contribute to the aggregtion with a reachable state.

C. Aggregated iteration matrix

From the offset labelled BDD *reach* it is possible to derive the multi-offset labelled MTBDD of the iteration matrix from the ordinary MTBDD of the iteration matrix. The basic algorithm is the same as in [20]. The main difference is that in the generated MTBDD not the actual offset values but rather pointers to the multi-offset arrays are stored.

During the construction process for every node two pointers to BDD *reach* are stored: One for the s and one for the t variables. As these pointers represent the whole set of offsets corresponding to this node, the usual comparison may be used to detect offset clashes and will be sufficient: A node to be inserted in the offset labelled MTBDD for the iteration matrix is equal to another node on the same level if and only if the node and its offset pointer coincide. Equal nodes won't be re-inserted when constructing the multi-offset labelled MTBDD while unequal nodes have to be inserted.

```
(1)  agg_vector(node, offset, coarse_offset, s_i)
(2)    if (node = ZERO)
(3)      return
(4)    else if (level = $S$)
(5)      coarse_vector[coarse_offset]+=vector[offset]
(6)      return
(7)    else if (level < $D$)
(8)      agg_vector(node.else, offset,
                     coarse_offset, s_{i+1})
(9)      agg_vector(node.then, offset+node.else_off[$i_S$],
                     coarse_offset+node.else_off[$i_D$], s_{i+1})
(10)     return
(11)   else if ($D \leq$ level < $S$)
(12)     agg_vector(node.else, offset,
                     coarse_offset, s_{i+1})
(13)     agg_vector(node.then, offset+node.else_off[$i_S$],
                     coarse_offset, s_{i+1})
(13)     return
```

Fig. 8. Vector aggregation algorithm

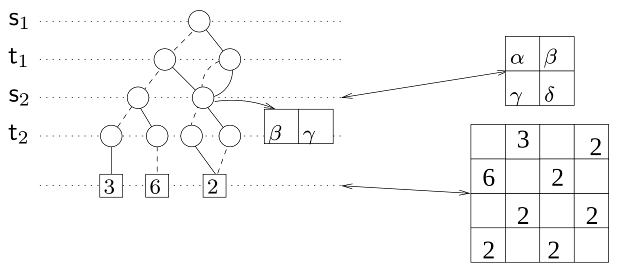

Fig. 9. Aggregation in a MTBDD

An example of the aggregation step is shown in Fig. 9. The multi-offset labelling is not shown in the figure. However, as in this example the reachable state space equals the potential state space, the offset arrays for all the nodes in the s_1 and t_1 variable levels are just [2,1] while the offsets for all nodes in the s_2 and t_2 variables are equal to [1]. The aggregation of the s_2-Variable causes a reduction of the matrix size to one quarter.

To represent the aggregated matrices it remains to store the matrix entries of the aggregated systems in the given MTBDD. It will turn out to be useful to store the diagonal entries of the aggregated systems in separate vectors, so in the MTBDD only non-diagonal elements will have to be stored. Therefore, at the MTBDD nodes in the aggregation variable levels arrays are added storing the aggregated values calculated with eq. (2). The size of such an array is calculated individually for each node by counting the number of paths from the root node leading to this node. In the summation, paths leading to a diagonal value of the aggregated system are skipped. This can be checked by evaluating the offsets corresponding to the aggregated system: The s- and t-offsets have to be different. As long as the BDD is always traversed in a depth first order, it suffices to store for every node in an aggregation variable level the current position of its pointer to the array of aggregated values. Below, this pointer will be called *aggregate_pointer*. Prior to reading or writing values of an aggregated system, the pointers in the corresponding aggregation level have to be reset. Every time a node is visited during BDD traversal its pointer position is incremented. So the pointer keeps the index of the current value in the array as indicated by the unidirectional arrow in Fig. 9.

For the aggregation step, let S be the variable level the aggregation starts from, D the variable level of the aggregated system and i_S and i_D the corresponding indices of the offset arrays. Before calculating the aggregated rates according to eq. (2), the values in the arrays of the D level and the diagonal array have to be reset to 0. The aggregation algorithm is given in Fig. 10. For simplicity, only the case where S is the constant level is shown. In the other case, where S is an aggregation variable level, reading *node.value* in line (5) would change to first incrementing this node's aggregate_pointer and then reading the value given by the current pointer position. To avoid another parameter in the agg_matrix algorithm, the global variable current_ptr is used.

The invocation of agg_matrix(root(MTBDD),0,0,s_1) performs the aggregation. Basically, the algorithm calculates the offsets of the source and destination system. Whenever the level D is reached, the pointer to the current coarse value is set to the next position in the array of aggregated values as seen in lines (12-13). Traversing the nodes from level D to level S, line (8) ensures that only the fine system's offsets are calculated in the next recursive call. At line (5) the S level is reached, so the current weighted fine vector's value is added to the current coarse value. After a node of level D is completely processed, the current coarse value is normalized in line (17) by the coarse vector entry corresponding to the coarse offset of this node, that is the probability mass of all the states in this aggregate, and finally in line (18) this new non-diagonal matrix entry is subtracted from the diagonal value. For the sake of simplicity, the recursion in line (15) on the s-variables, which is given by four different possibilities, is not shown in detail.

With the aggregated system smoothing steps can be performed in the usual way. Prior to a matrix vector multiplication of an aggregated system, the corresponding counters in each node of the aggregation level have to be reset.

D. Correction of the upper level solution

The correction of an upper level solution by an aggregated solution according to eq. (3) is done in a similar way as the aggregation of a vector by a single depth first traversal of the reachability BDD.

E. Sparse matrix approach

The sparse matrix approach, as proposed in [20] is adapted to the multilevel case in the following way: For speeding up the smoothing steps and aggregation method of the fine system, the lower levels of

```
(1)  agg_matrix(node, row_offset, coarse_row_offset, $\mathsf{s}_i$)
(2)    if (node = ZERO)
(3)      return
(4)    if (level = $S$)
(5)      increment value of current_ptr by
           vector[row_offset]*node.value
(6)      return
(7)    if (level $\geq D$)
(8)      coarse_offsets_change := false
(9)    else
(10)     coarse_offsets_change := true
(11)   if (level = $D$)
(12)     increment node.aggregate_pointer
(13)     current_ptr := node.aggregate_pointer
(14)   if (not diagonal element)
(15)     recurse on s-variables updating required offsets
(16)     if (level = $D$)
(17)       normalise value of current_ptr by
             coarse_vector[coarse_row_offset]
(18)       subtract value of current_ptr from
             coarse_diag[coarse_row_offset]
```

Fig. 10. Matrix aggregation algorithm

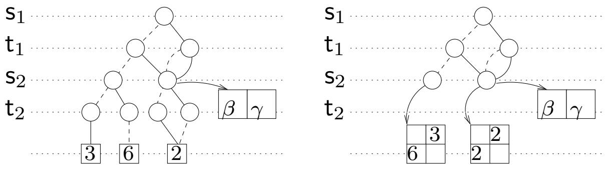

Fig. 11. Adding sparse matrices

the MTBDD are substituted by sparse matrices. In our current implementation of the algorithm, only the nodes up to the first aggregation variable level can be replaced by sparse matrices. C. f. the example in Fig. 11 where the first aggregation level is s_2, so the sparse matrix level cannot exceed this level.

It is important to note that, using our current implementation, the sparse matrix speedup is only achieved for the fine system. All the aggregated systems do not benefit from the sparse matrix speedup, as their levels still have to be processed recursively.

F. Memory considerations

In the current version of the symbolic multilevel algorithm, Jacobi iterations are used for the smoothing steps. So a lower bound for the memory consumption of the multilevel algorithm is the memory consumption of the Jacobi algorithm. The total memory consumption of the multilevel algorithm can be written as

$$matrix + diag + vect + max(vect, agg) + sm \qquad (4)$$

where *matrix* is the memory used for the transition matrix of the fine system - without diagonal elements - stored as multi-offset labelled MTBDD, *diag* is the memory for the vector of diagonal elements of the transition matrix of the fine system, *vect* is the memory for one iteration vector of the fine system, *agg* means the memory for additional aggregation information (i.e. $diag_i$ and two times $vect_i$ for each aggregated system i, one temporary iteration vector *temp* and the *rates* for the aggregated transition systems) and the optional sparse matrix memory *sm*. The vector *temp* ist used for the smoothing steps of the aggregated systems. It has the same size as the iteration vector of the finest aggregate. Depending on the number of aggregation levels and their location, the size of *agg* may vary considerably. Aggregation levels next to the root of the graph result in a low memory consumption, aggregation levels next to the terminal nodes result in high memory consumption.

In the current multilevel implementation, the nodes storing the matrix of the transition system are 4 Bytes bigger than the nodes used for Jacobi or pseudo Gauss Seidel iterations (24 Bytes).

IV. Experiments

In the following, experimental results of the MTBDD-based multilevel algorithm are given. The current implementation uses the probabilistic symbolic model checker PRISM as a framework. All measurements were performed on an Intel Xeon 3.0 GHz processor with 2 GByte of main memory.

The following numerical algorithms are compared:

- *JOR*, Jacobi OverRelaxation with relaxation parameter 0.9.
- *PGS* Pseudo Gauss-Seidel [20], a block oriented method which uses the Gauss-Seidel idea on the block level, inside the blocks JOR with overrelaxation parameter 0.9 is used.
- *ML1* Multilevel algorithm using 8 pre- and post-smoothing steps respectively on the fine system and 6 pre- and post-smoothing steps respectively for the aggregated systems.
- *ML2* Multilevel algorithm using 4 pre- and post-smoothing steps respectively on the fine system and 4 pre- and post-smoothing steps respectively for the aggregated systems.

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 ML experiment the aggregation variable levels for each aggregate are given in parenthesis, starting with the first and ending with the last aggregate. For the aggregation example given in Fig. 5 the algorithm would be called MLx(3,2), $x \in \{1,2\}$, i.e. the first aggregation reduces the fine system by eliminating s_3, the second aggregation reduces the first aggregate by eliminating s_2. In the MTBDD encoding the transition system, the corresponding t-variables are aggregated as well.

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

The tables are organised as follows: In the first three columns the model characteristics are shown: The *scaling* parameter of the model (e.g. number of tokens for the Kanban system), the number of reachable *states* and the number of *transitions* between the reachable states. Column *algorithm* specifies the numerical method used. In column *ML-cycles* the number of multilevel cycles until convergence is given. This only makes sense for the ML algorithm. Column *steps* gives the number of iteration steps until convergence for JOR and PGS. In the multilevel case the smoothing steps on the fine system are given. In *variable levels* the number of number of s-variables is shown. The next two columns give the number of s- (and t-) variables substituted by sparse matrices beginning from the bottom (*sparse levels*) or from the top (*block levels*) of the MTBDD (the latter is only applicable to the PGS scheme). In column *residual* the maximum norm of the vector $\vec{\pi}Q$ is given. The column *memory (kB)* shows the total memory consumption of the different algorithms in kilobyte, finally the column *time (s)* shows the consumed time until convergence was achieved.

The case studies given in the following subsections are chosen from the standard examples shipped with PRISM. The results are presented in Tables II-IV.

A. Tandem queueing network

This model was originally described in [11]. The remarkable part of the results given in Table II is that the multilevel algorithm converges faster although there are comparably few sparse levels used. Only for the smallest system considered (parameter 200) the standard methods are faster, which is due to the multilevel overhead.

B. Kanban manufacturing system

This model is described in [6]. For this model, initially the aggregation scheme reducing one submodel per aggregation, e.g. aggregation variable levels (37,25,13), was used. However, due to a low sparse level of 12 variables, this scheme did not lead to faster calculation time. So the aggregation levels for the multilevel algorithms were chosen in order to grant the same sparse level for the JOR and ML algorithms. The results are shown in Table III. It can be seen that in all cases the multilevel algorithm is between 1.7-1.9 times slower than the ordinary solvers used. But looking at the total number of iterations until convergence, the ML2 algorithm always takes fewer iterations on the fine system than the ordinary JOR solver. Thus speeding up the aggregation steps and iterations of the aggregated systems, which is part of our future work, the solution times are expected to improve.

C. Flexible manufacturing system

This model is described in [7]. The results are presented in Table IV. Aggregating the model according to the submodel structure with the aggregation variable levels (43,34,20), the number of total smoothing steps of the fine system was reduced for all cases except for parameter 4. However, because of the low sparse level this scheme did not lead to faster calculation time. In order to obtain more competitive results, the aggregation scheme was parametrised such that the same number of sparse levels as with the ordinary JOR algorithm was used. As shown in Table IV, except for the smallest system considered (parameter 4) the ML algorithm leads to fewer iterations of the fine system, but due to the multilevel overhead only for parameter 6 the ML1 algorithm slightly outperforms the standard solvers.

V. Conclusion

In this paper, a symbolic multilevel method has been presented. For that purpose, we introduced the concept of multi-offset labelled MTBDDs. The MTBDD data structure is supplied by additional arrays for storing the aggregated transition matrices in the same MTBDD as the fine system. Symbolic aggregation algorithms for vectors of reachable states and transition matrices are given. First empirical studies indicate that the symbolic multilevel algorithm could be a way to alleviate the state space explosion problem for the numerical solution of Markov chains. The experimental results also indicate that the benefits of the multilevel approach strongly depend on the model used.

References

[1] 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. *Form. Meth. in Sys. Design*, 10(2/3):171–206, 1997.

[2] P. Bazan and R. German. Approximate analysis of stochastic models by self-correcting aggregation. In *Proc. of QEST'05*, pages 134–144. IEEE Computer Society Press, 2005.

[3] P. Buchholz. Multilevel solutions for structured Markov chains. *SIAM J. Matrix Anal. Appl.*, 22(2):342–357, 2000.

[4] P. Buchholz and T. Dayar. Comparison of multilevel methods for kronecker-based markovian representations. *Computing*, 73(4):349–371, 2004.

[5] G. Ciardo, G. Luettgen, and R. Siminiceanu. Saturation: An efficient strategy for symbolic state-space generation. In *Proc. TACAS'01*, pages 328–342, Genova, Italy, 2001. Springer, LNCS 2031.

[6] G. Ciardo and M. Tilgner. On the use of Kronecker operators for the solution of generalized stochastic Petri nets. *ICASE Report*, 96(35), 1996.

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

[8] P.J. Courtois. *Decomposability, queueing and computer system applications*. ACM monograph series, 1977.

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

[10] H. Hermanns, M. Kwiatkowska, G. Norman, D. Parker, and M. Siegle. On the use of MTBDDs for performability analysis and verification of stochastic systems. *Journal of Logic and Algebraic Programming*, 56(1-2):23–67, 2003.

[11] 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.

[12] 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. Proceedings of the ACM Sigmetrics and Performance 1994, International Conference on Measurement and Modeling of Computer Systems.

[13] J.R. Koury, D.F. McAllister, and W.J. Stewart. Iterative Methods for Computing Stationary Distributions of Nearly

scaling	states	transistions	algorithm	ML-cycles	steps	variable levels	sparse levels	block levels	residual	memory (kB)	time (s)
200	80601	280599	ML1(16,12,8)	159	2544	17	2	-	$2.2204 \cdot 10^{-16}$	2423.0	92.35
			ML2(16,12,8)	240	1920	17	2	-	$1.1102 \cdot 10^{-16}$	2423.0	91.49
			JOR	-	3670	17	15	-	$2.2204 \cdot 10^{-16}$	2357.3	86.88
			PGS	-	2624	17	11	6	$2.2204 \cdot 10^{-16}$	868.4	75.19
400	321201	1121199	ML1(16,12,8)	341	5456	19	4	-	$2.2204 \cdot 10^{-16}$	5675.0	482.97
			ML2(16,12,8)	524	4192	19	4	-	$2.2204 \cdot 10^{-16}$	5675.0	503.47
			JOR	-	7389	19	14	-	$5.5511 \cdot 10^{-17}$	6101.9	860.11
			PGS	-	5274	19	12	7	$1.1102 \cdot 10^{-16}$	3290.7	682.29
800	1282401	4482399	ML1(16,12,8)	739	11824	21	6	-	$2.2204 \cdot 10^{-16}$	22574.7	3381.51
			ML2(16,12,8)	1133	9064	21	6	-	$2.2204 \cdot 10^{-16}$	22574.7	3659.10
			JOR	-	14880	21	15	-	$1.1102 \cdot 10^{-16}$	23443.7	7429.79
			PGS	-	10614	21	13	8	$2.2204 \cdot 10^{-16}$	22574.7	5686.90

TABLE II: Tandem model

scaling	states	transistions	algorithm	ML-cycles	steps	variable levels	sparse levels	block levels	residual	memory (kB)	time (s)
4	454475	3979850	ML1(13,8)	39	624	48	36	-	$5.4149 \cdot 10^{-10}$	8752.2	67.65
			ML2(13,8)	59	472	48	36	-	$4.8971 \cdot 10^{-10}$	8752.2	70.76
			JOR	-	803	48	36	-	$5.5631 \cdot 10^{-10}$	8721.0	37.28
			PGS	-	402	48	29	19	$5.5785 \cdot 10^{-10}$	5204.0	40.43
5	2546432	24460016	ML1(22,16,8)	54	864	48	27	-	$1.8432 \cdot 10^{-10}$	45701.1	574.47
			ML2(22,16,8)	79	632	48	27	-	$2.1860 \cdot 10^{-10}$	45701.1	582.64
			JOR	-	663	48	27	-	$2.2239 \cdot 10^{-10}$	45660.5	340.59
			PGS	-	573	48	27	19	$2.2321 \cdot 10^{-10}$	25842.1	341.11
6	11261376	115708992	ML1(24,17,9)	72	1152	48	25	-	$9.5764 \cdot 10^{-11}$	198947.8	3536.67
			ML2(24,17,9)	107	856	48	25	-	$9.9801 \cdot 10^{-11}$	198947.8	3692.36
			JOR	-	891	48	25	-	$9.6216 \cdot 10^{-11}$	198901.0	2083.72
			PGS	-	772	48	25	19	$9.4921 \cdot 10^{-11}$	111082.6	2142.28

TABLE III: Kanban model

scaling	states	transistions	algorithm	ML-cycles	steps	variable levels	sparse levels	block levels	residual	memory (kB)	time (s)
4	35910	237120	ML1(15,11,8)	61	976	55	41	-	$9.0486 \cdot 10^{-10}$	2376.9	7.50
			ML2(15,11,8)	90	720	55	41	-	$8.7807 \cdot 10^{-10}$	2376.9	8.17
			JOR	-	803	55	41	-	$8.7390 \cdot 10^{-10}$	2244.6	4.33
			PGS	-	749	55	33	22	$8.6704 \cdot 10^{-10}$	1447.6	5.54
5	152712	1111482	ML1(28,17,8)	56	896	55	28	-	$4.7222 \cdot 10^{-10}$	5184.0	39.31
			ML2(28,17,8)	83	664	55	28	-	$5.9121 \cdot 10^{-10}$	5184.0	41.12
			JOR	-	996	55	28	-	$5.5554 \cdot 10^{-10}$	4959.9	31.53
			PGS	-	936	55	28	22	$5.5231 \cdot 10^{-10}$	3798.6	36.37
6	537768	4205670	ML1(32,16,8)	50	800	55	24	-	$1.5117 \cdot 10^{-10}$	12641.0	132.99
			ML2(32,16,8)	81	648	55	24	-	$2.1612 \cdot 10^{-10}$	12641.0	152.10
			JOR	-	1189	55	24	-	$3.9511 \cdot 10^{-10}$	12304.8	139.23
			PGS	-	1124	55	24	22	$3.8807 \cdot 10^{-10}$	8171.7	152.97

TABLE IV: FMS model

Completely Decomposable Markov Chains. *SIAM Journal on Algebraic and Discrete Methods*, 5(2):164–186, June 1984.

[14] M. Kuntz, M. Siegle, and E. Werner. Symbolic Performance and Dependability Evaluation with the Tool CASPA. In *Europ. Perf. Engineering Workshop*, pages 293–307. LNCS 3236, 2004.

[15] M. Kwiatkowska, G. Norman, and D. Parker. Probabilistic model checking in practice: Case studies with PRISM. *ACM Performance Evaluation Review*, 32(4):16–21, 2005.

[16] K. Lampka and M. Siegle. Activity-local Symbolic State Graph Generation for High-Level Stochastic Models. In *Proc. of 13th GI/ITG Conference on Measuring, Modelling and Evaluation of Computer and Communication Systems (MMB)*, pages 245–263, Nürnberg, Germany, 2006.

[17] K. Lampka and M. Siegle. Analysis of Markov Reward Models unsing 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.

[18] R. Mehmood. *Disk-Based Techniques for Efficient Solution of Large Markov Chains.* PhD thesis, School of Computer Science, Faculty of Science, University of Birmingham, 2005.

[19] 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.

[20] D. Parker. *Implementation of symbolic model checking for probabilistic systems.* PhD thesis, School of Computer Science, Faculty of Science, University of Birmingham, 2002.

[21] PRISM website. `http://www.cs.bham.ac.uk/~dxp/prism/`.

[22] M. Siegle. *Behaviour analysis of communication systems: Compositional modelling, compact representation and analysis of performability properties.* Shaker Verlag, Aachen, 2002.

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

[24] Y. Takahashi. A Lumping Method for Numerical Calculation of Stationary Distributions of Markov Chains. Technical Report B-18, Tokio Institute of Technology, Dpt. of Information Sciences, June 1975.