Speeding up the Symbolic Multilevel Algorithm

Johann Schuster
University of the Federal Armed Forces Munich
johann.schuster@unibw.de

Markus Siegle
University of the Federal Armed Forces Munich
markus.siegle@unibw.de

***Abstract*—This paper describes two recent improvements to the symbolic multilevel algorithm. Firstly, it is shown how to use precalculations for the aggregation process of the matrix. Secondly, faster vector aggregation/disaggregation routines using precalculated indices are presented. Experiments show the practical use of these changes.**

I. Introduction

This paper is about a fast approach to the steady-state solution of Continuous Time Markov Chains (CTMCs), called multilevel algorithm, as originally presented in [1]. The idea is to perform iterative aggregation/disaggregation steps combined with additional smoothing steps in order to achieve a faster convergence than with standard numerical methods. There are several approaches to store large transition rate matrices of Markov Chains. For two of them, namely the Kronecker and MTBDD (Multi-Terminal Binary Decision Diagram) data structures, variants of the multilevel algorithm are known [2], [3].

The Kronecker-based approach aggregates submatrices of the Kronecker representation supporting arbitrary (and non-fixed) aggregation orderings and several smoothing algorithms, while the symbolic approach supports arbitrary aggregations of a power of 2 potential states but is so far limited to a fixed aggregation ordering. In this paper, we present a faster variant of the symbolic multilevel algorithm using precalculations. Most notable, the speedup is achieved by only a slight increase of memory. The algorithms presented are prototypically implemented in the PRISM tool [4]. As the PRISM numerical solvers are also used for the CASPA tool [5], the multilevel algorithm can easily be used for this tool as well. The paper is organised as follows: Sec. II gives a short introduction to the multilevel algorithm. In Sec. III, the use of symbolic data structures is sketched. The use of precalculations for symbolic matrix aggregations is shown in Sec. IV. Faster vector aggregation/disaggregation routines are presented in Sec. V. A comparison of the new algorithm variant and the older version from [6] is given in Sec. VI. Finally, Sec. VII concludes the paper.

II. Multilevel Algorithm

The multilevel algorithm for Markov Chains has been presented in [1]. Its basic idea is to aggregate the given Markov Chain M to a smaller one M' which can be solved more easily. The solution of M' is then disaggregated to a solution of M. It is well-known that a finite CTMC with n states can be represented by a $n \times n$ matrix, namely its infinitesimal generator matrix Q. For the following explanations we only look at the transition matrix *trans*, that is Q with all diagonal elements set to 0. We sketch the basic multilevel idea by means of a small two-level example: Suppose we are given a transition matrix $trans$ and an initial solution vector π as seen in the upper left part of Fig. 1. Two neighbouring states

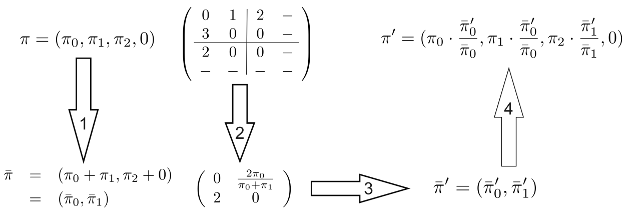

$\pi = (\pi_0, \pi_1, \pi_2, 0)$
$\left(\begin{array}{cc|cc} 0 & 1 & 2 & - \\ 3 & 0 & 0 & - \\ \hline 2 & 0 & 0 & - \\ - & - & - & - \end{array}\right)$
$\pi' = (\pi_0 \cdot \frac{\bar{\pi}'_0}{\bar{\pi}_0}, \pi_1 \cdot \frac{\bar{\pi}'_0}{\bar{\pi}_0}, \pi_2 \cdot \frac{\bar{\pi}'_1}{\bar{\pi}_1}, 0)$
1
2
4
$\bar{\pi} = (\pi_0 + \pi_1, \pi_2 + 0) = (\bar{\pi}_0, \bar{\pi}_1)$
$\begin{pmatrix} 0 & \frac{2\pi_0}{\pi_0+\pi_1} \\ 2 & 0 \end{pmatrix}$
3
$\bar{\pi}' = (\bar{\pi}'_0, \bar{\pi}'_1)$

Fig. 1. Basic aggregation/disaggregation scheme

should be aggregated as indicated by the division of the matrix in 2×2 blocks. A basic aggregation/disaggregation procedure can be divided into the following steps (as indicated by the arrows in Fig. 1):

1) Aggregation of π: The state probabilities of all states belonging to an aggregated are summed up
2) Aggregation of $trans$: aggregated rates for non-diagonal blocks are calculated
3) Calculation of an approximate solution
4) Disaggregation of the solution to an approximate solution of the original matrix

In order to improve the solution additional smoothing steps have to be performed (i.e. iteration steps with conventional solvers). The basic multilevel procedure has to be repeated until a certain accuracy of the result is achieved. Our sketch only shows a so-called *two-level* algorithm. Recursive solution of aggregated systems up to a certain recursion depth leads to the multilevel v-cycle.

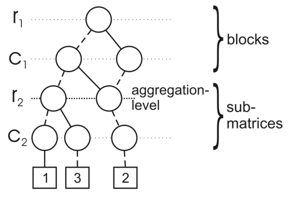
III. Adaptation to MTBDD data structure

A. Data structure preliminaries

The PRISM tool uses MTBDD data structures to store the $n \times n$ transition matrix. Compositional modelling leads to unreachable states (either by synchronisation constraints or by the fact that MTBDDs always encode a power of 2 states). Therefore unreachable states have to be taken into account for the multilevel algorithm. In the example shown in Fig. 2a we assume that the states corresponding to the first three rows/columns are reachable, while the state corresponding to the last row/column is unreachable. In the following, a

$$\left(\begin{array}{cc|cc} 0 & 1 & 2 & - \\ 3 & 0 & 0 & - \\ \hline 2 & 0 & 0 & - \\ - & - & - & - \end{array}\right)$$

(a) Matrix



(b) MTBDD

Fig. 2. Example Matrix and corresponding MTBDD

row/column index always starts with 0. The symbolic encoding works as follows:

- A matrix is interpreted as a function $\mathbb{N}\times\mathbb{N} \rightarrow \mathbb{R}$, mapping row and column index to the real value of the matrix entry.
- Rows and columns are stored as dual numbers with $k := \lceil \log n \rceil$ Bits (in our case 3 states $\rightarrow$ 2 Bits)
- Row- (r_i) and column variables (c_i) are mixed to bit strings $(r_1, c_1, r_2, c_2, \ldots, r_k, c_k)$
- The bit strings are encoded by paths in a binary decision diagram ($0 \simeq \texttt{false}$, $1 \simeq \texttt{true}$)
- The real values of the matrix entries are represented by the leaves corresponding to such paths
- Reduction rules avoid the storage of redundant paths
- Zero-leaves (i.e. zero-values of the matrix) are omitted

With this transformation in mind, entry $(0, 1) \mapsto 1$ in the example matrix is coded as bit string $(0, 0, 0, 1)$ and is represented by the leftmost path in Fig. 2b. For a more verbose explanation we refer to [6].

B. Aggregation in the context of MTBDDs

An aggregation of two neighbouring states would correspond to an aggregation of the 2×2 blocks as indicated in Fig. 2a. The same aggregation in MTBDD terms is given in Fig. 2b. It can be defined considering level r_2 as the aggregation level of the MTBDD. In the aggregated MTBDD, only the MTBDD variables labelled as blocks remain (they determine the 2×2 structure of the aggregated matrix), while the nodes labelled by submatrices are aggregated. In the aggregated matrix, the nodes in the aggregation level now play the role of the terminal nodes and have to be filled with the aggregated values by the aggregation routine. In this way the symbolic multilevel algorithm can always aggregate a power of 2 potential states to one aggregated state.

IV. Node Characterisation

In this section it is shown how redundant calculations can be avoided using precalculations. Aggregated values for nodes that can be precalculated have to be stored only once, therefore saving memory.

A. Characterisation of Nodes

Looking again at Fig. 2b, for the characterisation we are only interested in the *aggregation nodes*, i.e. nodes that belong to the aggregation level (for real calculations there can be usually more than one aggregation level). Firstly, looking at the corresponding matrix in Fig. 2a we can distinguish between nodes belonging to diagonal blocks in the aggregated matrix and those that do not. Diagonal elements of the aggregated matrices are calculated on the fly as the negative sums of the non-diagonal elements and they are stored in a separate vector. Therefore in the sequel we only regard nodes that belong to non-diagonal elements, i.e. nodes, where aggregations have to be performed. Next, we introduce the notation of *reducible* and *non-reducible* nodes by means of the simple four-state-model given in Fig. 3a. It can be seen that the aggregated rate $\bar{\lambda}$ depends on the cumulative sums of the rates to be aggregated (λ_0 and λ_1, respectively) and the current state probabilities (π_0 and π_1, respectively) of the states from which they emanate. Reducible nodes are those where the aggregation equation cancels out and the result can be precalculated, otherwise we call them non-reducible nodes. There are three different cases:

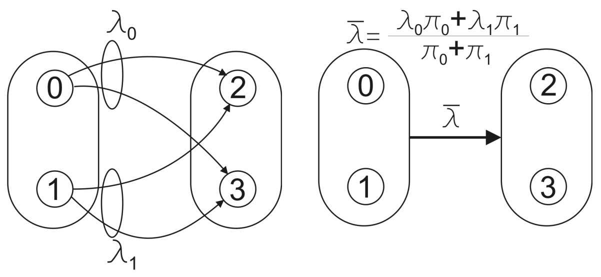
- $\lambda_0 = \lambda_1$: fraction cancels out $\Rightarrow$ *reducible*
- $\lambda_0 = 0$, $\lambda_1 \neq 0$ (without loss of generality):
  - cancels out for $\pi_0 = 0 \Rightarrow$ *reducible*
  - does not cancel out for $\pi_0 \neq 0 \Rightarrow$ *non-reducible*
- $\lambda_0 \neq \lambda_1$, both non-zero: no cancellation $\Rightarrow$ *non-reducible*

Note that this characterisation can easily be generalised to a larger number of states to be aggregated. Further we would like to stress that it is possible to do all these calculations by purely symbolic operations.

B. Aggregation clash

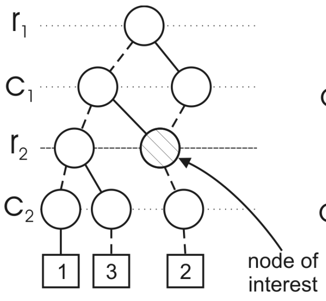
Similar to an offset clash introduced in [7], in the multilevel context, the notion of an aggregation clash is important. Consider once again our running example given by the matrix in Fig. 2a.

According to Sec. III-A, the corresponding MTBDD representation is given in Fig. 4a. Suppose again that r_2 is the aggregation level, i.e. consider aggregations of 2×2 submatrices. The problem is that there is one node (marked by *node of interest*) representing both the top right $\begin{pmatrix} 2 & - \\ 0 & - \end{pmatrix}$ (non-reducible) and bottom left $\begin{pmatrix} 2 & 0 \\ - & - \end{pmatrix}$ (reducible) submatrix. In this case we speak of an an *aggregation clash*. This clash has to be resolved by splitting this node into two separate nodes, as seen in Fig. 4b. In Sec. VI we will see that aggregation clashes are much rarer than offset clashes.

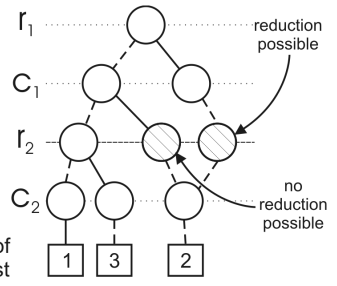

λ_0
0
2
1
3
λ_1
$\bar{\lambda} = \frac{\lambda_0\pi_0 + \lambda_1\pi_1}{\pi_0 + \pi_1}$
0
1
$\bar{\lambda}$
2
3

(a) Rates to be aggregated (b) Aggregated rate

Fig. 3. Aggregation of matrices and MTBDDs

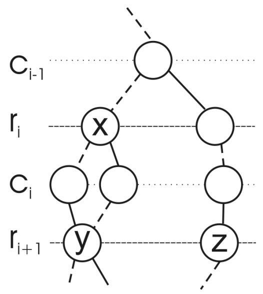


(a) MTBDD trans

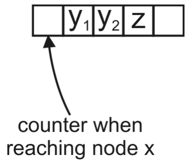


(b) Clash resolved

Fig. 4. Example: Aggregation clash



(a) Higher aggregations



(b) Critical values

Fig. 5. Skipping reducible nodes

C. Aggregation Offsets

Without exploiting the reducibility of nodes, the matrix aggregation amounts to a (in our case depth-first-) traversal of the MTBDD *trans* up to the variable level of the system to be aggregated. Once the nodes are distinguished, aggregations of reducible nodes do not have to be calculated in every iteration, but they can be precalculated once as shown above. When an aggregated value is known in advance, it is desirable to be able to skip the aggregation of such a node. A slight problem arises when successive aggregations have to be performed, as all aggregated values of one aggregation level are stored sequentially: In Fig. 5a a part of an MTBDD representing a transition matrix is given. Assume there are two aggregation levels r_i and r_{i+1} and assume further that x is a diagonal (or reducible) node for the aggregation level r_i, and therefore might be skipped (i.e. its sub-MTBDD is not processed). Let y be an non-reducible node and let us further assume that during the aggregation in r_{i+1} the aggregated matrix values were stored as indicated in Fig. 5b (two paths leading from x to y, so two values y_1 an y_2 have to be stored for the aggregated values of node y). Now the point is that if during the aggregation at level r_i node x is skipped (i.e. its sub-MTBDD is not processed), the counter indicating the position of the aggregated value has to be updated as well. Otherwise in the example node z would get the value y_1 as two incrementations of the counter were missed. In the spirit of the offset labelling concept presented in [7] we therefore introduce the concept of aggregation offsets. That means for every node x at an aggregation level r_i we precalculate the number of non-reducible nodes in aggregation level r_{i+1} that are reached during a traversal of the sub-MTBDD below x. This is the offset to be added to the array pointer for the aggregated values when such a node is skipped.

V. Vector Aggregation/Disaggregation

Profiling revealed that the vector aggregation/disaggregation used in [6] is too time-consuming compared to the other operations for the multilevel algorithm. For a faster approach we make use of the fact that the MTBDD offset labelling allows for a compact storage of the vector of reachable states (without gaps for unreachable states - this was the aim of the hybrid approach [7]). Therefore, vector aggregation/disaggregation operations can be based on indices of the borders between states belonging to different aggregates. It is sufficient to know how many states are grouped to a certain aggregate. Instead of traversing the entire BDD $reach$ every time an aggregation/disaggregation is performed, the aggregation borders are extracted within a single BDD traversal as a preprocessing step. Algorithm 1 shows the principle of aggregating a vector *vect* to an aggregate *agg_vect*, given that the borders (array *aggborder*) are available. First, the counter j for the vector elements of *vect* is reset. The outer FOR-loop runs through all elements of the aggregate *agg_vect*. The current element of the aggregate is then reset and the inner loop adds elements of *vect* until the aggregation-border of the current state i is reached. As a drawback, this approach has a higher memory consumption, as for every aggregation as many borders as there are aggregated values have to be stored.

Algorithm 1 Faster vector aggregation
```
$j = 0$
for $i = 0$ to AGGVECTORSIZE do
    $agg\_vect[i] = 0$
    while $j < aggborder[i]$ do
        $agg\_vect[i] = agg\_vect[i] + vect[j]$
        $j = j + 1$
    end while
end for
```

VI. Experimental Results

We present some experimental results for the Flexible Manufacturing System (FMS) [8] , and compare the new results to the results given in [6]. The numerical results (i.e. state probabilities, residual) remained the same, so they are omitted. We use the same number of smoothing steps as in [6], namely 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 and the same sparse matrix substitutions. The results are shown in Tab. I, which is organised as follows (Subscript old references to the algorithm presented in [6], subscript new references to the algorithm presented in this paper):

- scaling: the model scaling parameter, i.e. the initial number of raw parts for each machine of the FMS.

sca-ling	states	trans.	agg.-level	MTBDD minterms			MTBDD nodes			memory ratio	speedup
				total	red.	ratio	total	red.	ratio		
5	$1.5 \cdot 10^5$	$1.1 \cdot 10^6$	47	$4.0 \cdot 10^4$	$3.3 \cdot 10^4$	0.83	41	10	0.24	1.06	1.17
			34	$1.7 \cdot 10^4$	$1.3 \cdot 10^4$	0.73	136	55	0.40		
			15	$5.1 \cdot 10^2$	$1.8 \cdot 10^2$	0.34	91	22	0.24		
6	$5.4 \cdot 10^5$	$4.2 \cdot 10^6$	47	$1.2 \cdot 10^5$	$1.0 \cdot 10^5$	0.87	53	11	0.20	1.04	1.18
			34	$4.3 \cdot 10^4$	$3.2 \cdot 10^4$	0.74	193	70	0.36		
			15	$9.4 \cdot 10^2$	$3.2 \cdot 10^2$	0.34	122	27	0.22		
7	$1.6 \cdot 10^6$	$1.4 \cdot 10^7$	47	$2.9 \cdot 10^5$	$2.5 \cdot 10^5$	0.84	70	11	0.16	1.03	1.14
			34	$9.2 \cdot 10^4$	$6.6 \cdot 10^4$	0.71	268	82	0.31		
			15	$1.6 \cdot 10^3$	$5.5 \cdot 10^2$	0.34	157	32	0.20		
8	$4.5 \cdot 10^6$	$3.9 \cdot 10^7$	59	$6.7 \cdot 10^5$	$5.8 \cdot 10^5$	0.87	86	12	0.14	1.02	1.24
			43	$1.8 \cdot 10^5$	$1.3 \cdot 10^5$	0.72	354	98	0.28		
			19	$2.6 \cdot 10^3$	$8.7 \cdot 10^2$	0.34	196	37	0.19		
9	$1.1 \cdot 10^7$	$9.9 \cdot 10^7$	59	$1.4 \cdot 10^6$	$1.2 \cdot 10^6$	0.85	108	12	0.11	1.01	1.21
			43	$3.4 \cdot 10^5$	$2.4 \cdot 10^5$	0.69	465	110	0.24		
			19	$3.9 \cdot 10^3$	$1.3 \cdot 10^3$	0.34	239	42	0.18		

TABLE I
IRREDUCIBLE/REDUCIBLE, MEMORY AND SPEEDUP STATISTICS FOR THE FMS MODEL

- states: number of reachable states
- trans.: number of reachable transitions
- agg.-level.: aggregation levels used
- MTBDD minterms: total number of minterms, minterms belonging to reducible nodes, ratio between the two
- MTBDD nodes: total number of nodes, reducible nodes, ratio between the two
- memory: memory ratio $\frac{mem_{new}}{mem_{old}}$
- speedup: speedup ratio $\frac{t_{old}}{t_{new}}$

It is remarkable that for all the FMS parameters we measured, no aggregation clash occurred. Note that there might be a large number of reducible minterms in contrast to only a few reducible nodes on the same level. This indicates a good sharing of reducible nodes. One can see that the number of reducible nodes decreases for higher aggregations. Most interestingly, there is only a slight increase in memory usage in contrast to a relatively good speedup. This is the case as the memory saved by the node characterisation for the matrix aggregations is compensated by the memory needed for the border markings for the vector operations. The speedup achieved is mostly due to the faster vector routines, the time saved in the matrix traversal for skipping known aggregations is nearly neutralised by the additional distinction (reducible/non-reducible node).

VII. CONCLUSION

In this paper we have presented two modifications to the symbolic multilevel algorithm presented in [6]. While the concept of node characterisation saves memory due to a compact storage of the aggregated matrices, the modified vector routines demand more memory but allow for faster iterations. Using these two concepts we have shown that with only a negligible effort of memory there is a significant improvement in the runtime.

With the concepts introduced here we are currently working on a variant of the symbolic multilevel algorithm for multi-core processors. Queueing jobs for parallel matrix aggregations or parallel smoothing cycles is possible by using the aggregation offsets introduced in this paper.

By introducing an intermediate sparse matrix concept for the BDD *reach*, similar to the concept already used for the MTBDD *trans*, we expect also fast vector operations but with more moderate memory demands.

As the aggregation procedure now is significantly faster than before, it would be interesting to use other parameter sets (for example [2] often use only one pre- and post-smoothing step), which will probably lead to faster convergence.

ACKNOWLEDGMENT

The authors would like to thank Deutsche Forschungsgemeinschaft for supporting this work under grant SI 710/5-1. This work is partially supported by DFG/NWO bilateral research programme ROCKS.

REFERENCES

[1] G. Horton and S. Leutenegger, "A Multi-Level Solution Algorithm for Steady-State Markov Chains," *ACM Performance Evaluation Review*, vol. 22, no. 1, pp. 191–200, May 1994, proceedings of the ACM Sigmetrics and Performance 1994, International Conference on Measurement and Modeling of Computer Systems.

[2] P. Buchholz and T. Dayar, "Comparison of Multilevel Methods for Kronecker-based Markovian Representations," *Computing*, vol. 73, no. 4, pp. 349–371, 2004.

[3] J. Schuster and M. Siegle, "A Multilevel Algorithm based on Binary Decision Diagramms," in *14th Int. Conf. on Analytical and Stochastic Modelling Techniques and Applications (ASMTA'07)*, K. Al-Begain, A. Heindl, and M. Telek, Eds., June 2007, pp. 129–136.

[4] "PRISM website," http://www.prismmodelchecker.org, (last checked April 2010).

[5] J. Schuster and M. Siegle, "Dependability modelling with the stochastic process algebra tool CASPA," in *Procceedings of the DYADEM workshop*. ACM, 2010 (in press).

[6] ——, "A symbolic multilevel method with sparse submatrix representation for memory-speed tradeoff," in *14. GI/ITG Conf. Measurement, Modelling and Evaluation of Comp. and Communic. Systems (MMB08)*. VDE Verlag, 2008, pp. 191–205.

[7] D. Parker, "Implementation of symbolic model checking for probabilistic systems," Ph.D. dissertation, School of Computer Science, Faculty of Science, University of Birmingham, 2002.

[8] G. Ciardo and K. Trivedi, "A decomposition approach for stochastic reward net models," *Performance Evaluation*, vol. 18, no. 1, pp. 37–59, 1993.