Preprint from Proceedings of ITG/GI/GMM-Workshop "Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen", Meissen, Germany, February 2001, pp. 31-43

State Traversal guided by Hamming Distance Profiles *

Andreas Hett Christoph Scholl Bernd Becker

Institute of Computer Science
Albert-Ludwigs-University
79110 Freiburg im Breisgau, Germany
email: <name>@informatik.uni-freiburg.de

Abstract

In the last years symbolic techniques have revolutionized reachability analysis. Handling large, industrial designs is a key issue, involving the need to focus on memory consumption for BDD representation as well as time consumption to perform symbolic traversals of finite state machines. In this paper we address the problem of reachability analysis for large finite state machines, introducing a novel technique that performs reachability analysis using a sequence of "Hamming Distance guided" partial traversals based on dynamically chosen prunings of the transition relation. The efficiency and stability of our approach is demonstrated by experimental results: We succeed in completing reachability problems with significantly improved time performance and smaller memory requirements.

1 Introduction

One of the major problems in functional design verification is to decide whether a set of target states of a given *Finite State Machine* (FSM) can be reached from a set of initial states. Forward state space traversal techniques solve this problem by an iterative fixed point computation of all reachable states starting from the initial states. A significant number of techniques and refinements have been developed to make *Reachability Analysis* applicable for large designs.

Especially symbolic techniques which avoid an explicit representation of the set of reachable states and of the FSM transition relation by using BDD representations increased the problem sizes which could be solved by FSM traversal [8, 11, 13, 3].

In order to reduce time and memory consumption for circuits with realistic sizes, several improvements of the basic symbolic FSM traversal techniques have been proposed. To avoid huge BDD representations of monolithic transition relations for large FSMs, decomposition has been used: conjunctive partitioning for approximate FSM traversal (e.g. [7]) and disjunctive partitioning for exact FSM traversal (e.g. [4, 10]).

Other researchers replaced the pure breadth-first traversal of the original approach by a sequence of partial traversals [12, 5]. These methods take into account that traversals often produce the largest BDDs during intermediate steps. Therefore a sequence of simpler partial traversals is used to avoid large intermediate peak memory requirements.

*This work was supported in part by DFG grant Be 1176/8-3

In [12] single symbolic traversal steps are not initiated from the whole set of the newly reached states, but from subsets of it. The subsets are chosen in a way that their BDD representation has a "high density", i.e. many states are represented by a compact BDD.

In [5] a partial traversal is done based on a pruned transition relation. Information for pruning the transition relation is collected during a learning phase which determines "activity profiles" of the BDD nodes representing the transition relation. This is done by means of a limited number of FSM traversals with additional node activity analysis. Then, the transition relation is pruned, replacing "high cost" nodes by terminal zero, thus enabling a partial traversal method as an underapproximation of the reachable states. At the end, the partial traversal needs to be completed by using the original transition relation, accumulating all formerly left-out reachable states.

In this paper we introduce a novel technique for symbolic FSM traversal using sequences of partial traversals to avoid large peak memory requirements. In contrast to [12] and [5] our method has the following properties:

1. The transition relation is pruned based on an analysis of the newly reached states BDD. Thereby, two concepts are combined: partial traversals based on pruned transition relations and partial traversals based on subsets of the newly reached states set.

2. At first, we only traverse "short edges" in the state transition diagram. In the successive phases of the algorithm "longer and longer" edges are used.

3. Pruning of the transition relation is done dynamically during the FSM traversal.

4. In spite of the dynamic application of pruning, efficiency of the Computed Table[1] is guaranteed. The importance of this property is proven by recent research (e.g. [15]) which has shown, that the efficiency of the Computed Table plays a much more vital part in sequential applications like FSM traversals than in combinational applications.

Our experiments underline the quality of the approach. We consider reachability analysis for FMCAD'98 model checking traces [15, 14] as well as for ISCAS'89 benchmarks. In both cases we succeed in computing the results with significantly less memory requirements and improved runtime behavior. This is demonstrated by a comparison with symbolic FSM traversals both for monolithic and partitioned representations of the transition relation. As an example, the FMCAD model checking traces show runtime improvements for all traces, up to a factor 17. Also the *Peak Size*, i.e. the maximal number of nodes needed during a run, is significantly reduced, on average by a factor of more than 2.5. Finally we demonstrate the stability of our method with respect to parameter changes by an additional series of experiments.

The paper is structured as follows: In Section 2 basic definitions are given which are important for the understanding of the paper. Section 3 presents our approach to reachability analysis using distance driven partial traversals. Experimental results are presented in Section 4. Finally the results are summarized in Section 5.

2 Preliminaries

In this section we briefly provide essential definitions of *Binary Decision Diagrams*, *Finite State Machines* and *Exact State Space Traversal*.

[1] Applications of BDDs use a so-called *Computed Table* to prevent that identical computations are performed more than once [1].

2.1 Binary Decision Diagrams

Binary Decision Diagrams (BDDs) are directed acyclic graphs representing Boolean functions. In the restricted form of ROBDDs they even provide canonical representations. As defined in [2], ROBDDs are *ordered*, i.e. on each path from their root to a terminal node each input variable occurs only once and on each path the input variables occur in the same order. ROBDDs are *reduced*, i.e. they do not contain vertices either with isomorphic sub-graphs or with both outgoing edges pointing to the same node. Since we work only with ROBDDs in the following we briefly call them BDDs.

BDDs have proven to be an efficient data structure and nowadays they are widely used in applications of VLSI CAD, including traversals of FSMs.

2.2 Finite State Machines, Image Computation

A *Finite State Machine* (FSM) is defined as a 6-tuple $(I, O, S, \delta, \lambda, s_0)$ where I is the input alphabet, O is the output alphabet, S is a finite and non-empty set of states, $\delta : S \times I \rightarrow S$ is the next state function, $\lambda : S \times I \rightarrow O$ is the output function, and $s_0 \in S$ is the initial state.

Since we only consider FSMs corresponding to sequential circuits, in the following $I = \{0,1\}^k$, $O = \{0,1\}^m$ and $S = \{0,1\}^n$ contain bit vectors of fixed length. Then, the *characteristic functions* χ_R of subsets $R \subseteq S$ are Boolean functions $\chi_R : \{0,1\}^n \rightarrow \{0,1\}$ with $\chi_R(x) = 1 \Longleftrightarrow x \in R$.

The transition function $\delta : \{0,1\}^n \times \{0,1\}^k \rightarrow \{0,1\}^n$ can also be represented by the characteristic function of its Boolean relation $TR : \{0,1\}^n \times \{0,1\}^k \times \{0,1\}^n \rightarrow \{0,1\}$ with $TR(x,i,x') = 1 \Longleftrightarrow \delta(x,i) = x'$. TR is a characteristic function describing all existing transitions between states of the given FSM. The variables $x_1, \ldots, x_n$ corresponding to the first n arguments of TR are called *current state variables*, the variables $i_1, \ldots, i_k$ corresponding to the next k arguments of TR are called *(primary) input variables* and the variables $x'_1, \ldots, x'_n$ corresponding to the last n arguments are called *next state variables*.

If $FROM$ is a set of states in S, the image of $FROM$ under δ is defined as follows:

$$\textbf{Image}(\delta, FROM) = \{x' \in S | \exists i \in I, x \in FROM \text{ with } \delta(x,i) = x'\}.$$

In essence, the **Image** is the set of states that can be reached from the set of states $FROM$ by means of a single time-step (transition).

Thus, if the set of states $FROM$ is given by its characteristic function $FROM(x)$ and the transition relation is given by its characteristic function $TR(x,i,x')$, the image computation to determine the characteristic function $REACHED(x')$ of all states that can be reached from the set of states $FROM$ by a single transition can be performed by the following Boolean operations:

$$\begin{aligned} REACHED(x') &:= \textbf{Image}(TR(x,i,x'), FROM(x)) \\ &:= \exists_{x,i}(TR(x,i,x') \cdot FROM(x)) \\ &:= \exists_x(\tilde{TR}(x,x') \cdot FROM(x)) \end{aligned} \qquad (1)$$

with $\tilde{TR}(x,x') = \exists_i TR(x,i,x')$. Since the existential quantification for the input variables i can be done *before* the image computation for $FROM$, we assume in the following, that this existential quantification was done at the beginning of the FSM traversal and for simplicity we use $TR(x,x')$ instead of $\tilde{TR}(x,x')$ for the transition relation of the FSM.

2.3 Exact Forward Traversal

Symbolic forward FSM traversals start with a state set $FROM$ containing only the initial state and apply a sequence of image computations in order to compute the set of reachable states. After each image computation step the set of new states resulting from this step is added to the total set of reachable states ($Total\_REACHED$ set). The algorithm terminates as soon as $Total\_REACHED$ reaches a fixed point.

3 FSM Traversal by a sequence of Hamming Distance guided partial traversals

3.1 Main Idea and Goals

This section describes our approach to perform FSM traversals by a sequence of *distance driven partial traversals.* The purpose of this approach is to prevent peak sizes in memory consumption, when the final reachable state set allows a compact BDD representation, but intermediate results of the straightforward BFS based FSM traversal cannot be represented by BDDs of reasonable size. We have the challenge to choose a suitable order of adding new reachable states to the set of already reached states such that the representation of the set of reached states is as compact as possible. More precisely, we pursue the following goals with our distance driven partial traversal strategy:

Goal 1: We try to use $FROM$ sets with compact BDD representations as starting points for image computations.

Goal 2: For each image computation step we use a subset of the transition relation. This subset should contain only transitions leading us to a set of new states, providing a compact BDD when added to the set of already accumulated reachable states ($Total\_REACHED$).

Applications of BDDs use a so-called *Computed Table* to prevent that identical computations are performed more than once [1]. Recent research (e.g. [15]) has shown, that the efficiency of a Computed Table plays a much more vital part in sequential applications like e.g. FSM traversals than in combinational applications. The importance of the Computed Table for sequential applications leads us to an important third goal:

Goal 3: The performance of the Computed Table should not be decreased by the subsetting of the transition relation.

The intuition behind our method to achieve Goal 1 is that states with similar Hamming weights (number of 1's in the bit vector) [9] are supposed to combine to a compact BDD representation. Therefore the $FROM$ set of image computation should contain states with similar Hamming weights. We assume that we start the FSM traversal from the initial state $(0 \dots 0)$ with Hamming weight 0. Then we continue with states having small Hamming weights and – step by step – we increase the Hamming weights of the states, which are starting points of one time step of reachability analysis.

The subsetting of the transition relation is done in a way that we reach only new states "with a small distance" to the states in the $FROM$ set, i.e. states whose Hamming weight does not differ very much from the Hamming weights of the states in the $FROM$ set. This fulfills (heuristically) our Goal 2.

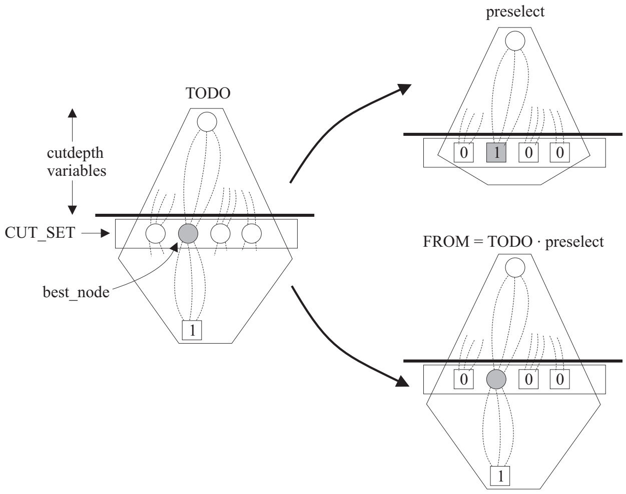

Figure 1: Determination of condition *preselect*.

If we change the pruning of the transition relation several times during the reachability analysis algorithm, we have to be careful how to prune the transition relation to achieve *Goal 3* all the same. In contrast to our approach, in [5], e.g., pruning of the transition relation is based on a replacement of nodes of the transition relation with “high cost” (determined in a learning phase based on an “activity profile”) by constant zero. However this replacement leads to the fact that the two transition relations, which result from different prunings, will typically have only a few cofactors in common. Thus it is not likely that the recursive procedure to compute the AND-EXIST operator of equation 1 will encounter common subproblems leading to Computed Table hits.

I.e. if we would apply a straightforward generalization of [5], namely a dynamic application of this kind of pruning for several times during the algorithm, efficiency of the Computed Table would decrease with high probability. (Note that this is no problem in [5], since the pruning is performed only once based on an initial learning phase.)

Since we want to adapt pruning dynamically during the traversal, we choose another pruning approach, which leads to subsets of the transisition relation having many cofactors in common with the original transition relation, such that *Goal 3* is fulfilled, too.

More details of the complete algorithm and the pruning method in particular are given in the next section.

3.2 Detailed Description of the Algorithm

In the first part of this section we describe, how the BDD $TODO$ representing the reached states, which where not yet used as starting points for image computations, is pruned before an image computation to achieve *Goal 1*. Afterwards we describe our dynamic pruning of the transition relation TR and finally show, how all parts work together leading to an algorithm, which performs a full FSM traversal using a sequence of partial distance driven FSM traversals.

Pruning of $TODO$

In contrast to the straightforward BFS traversal algorithm we do not start an image computation from the set of all newly reached states, but only from a subset of them to achieve *Goal 1*. To restrict the states we perform an AND operation between the representation of the states, which were not yet processed ($TODO$), and a characteristic function *preselect*. *preselect* is

procedure **iterate_until_converge** ($TODO$, $preselect$, $select$)
{
  $FROM := TODO \cdot preselect$
  $TODO := TODO \cdot \overline{preselect}$

  do
  {
    $TR' := TR \cdot preselect \cdot select$
    $New\_REACHED :=$ **Image**(TR', $FROM$)
    $Total\_REACHED := Total\_REACHED \cup New\_REACHED$
    $FROM := New\_REACHED \cdot preselect$
    $TODO := TODO \cup (New\_REACHED \cdot \overline{preselect})$
  }
  until (empty($FROM$))
}

Figure 2: Partial Traversal with respect to *preselect*, *select*

Reachability_Analysis
$\cdots$
$TODO := s_0$
$Hamming\_Weight := 1$

do
{
  do
  {
    ($preselect$, $select$) := **determine_selectors**($TODO$, $Hamming\_Weight$)
    iterate_until_converge($TODO$, $preselect$, $select$)
  }
  until (empty($TODO$))

  increase $Hamming\_Weight$
  $TODO := Total\_Reached$
}
until ($Hamming\_Weight$ = number_of_next_state_variables)

Figure 3: Reachability Analysis using Hamming Distance guided partial traversals

determined based on a Hamming weight metric by a procedure **determine_selectors**. The procedure analyzes the BDD $TODO$ representing the characteristic function of the states not yet processed. It considers a set CUT_SET of nodes of BDD $TODO$ immediately below a cut line after the first *cutdepth* variables (see also Figure 1). In a first step for each node v_i in CUT_SET we consider all assignments to current state variables, which define a path passing through v_i and leading to terminal one (these assignments represent certain states of $TODO$) and for each node v_i we compute the sum of the Hamming weights of these assignments. (Note that this computation can be done in time linear to the number of nodes of $TODO$.) Since we want to start with states having low Hamming weights we choose the node $best\_node \in CUT\_SET$ as the one with the smallest sum. Now *preselect* is the characteristic function of all assignments to the first *cutdepth* variables which lead to node *best_node*. The image computation is then started with $preselect \cdot TODO$ instead of $TODO$.

Pruning of *TR*

To achieve our *Goal 2*, we prune the transition relation TR to collect only states with similar Hamming weights. The pruning can be viewed as a selection of edges in the state transition diagram of the FSM. It is done by a conjunction $TR' := TR \cdot preselect \cdot select$ of TR with the characteristic function *preselect* and a new characteristic function *select*. First, the characteristic function *preselect* selects only edges, which start from states fulfilling condition *preselect*. However, not all such edges are considered, but only "short edges". Here "short edges" denote edges connecting states with similar Hamming weights. We select only edges between states whose Hamming distance is less or equal to a constant $Hamming\_Weight$ [2]. The selection of short edges is done by a conjunction with the characteristic function *select* depending on next state variables where

$$\begin{aligned} ON(select) &= \{y \mid \exists y' \in ON(preselect) \text{ with} \\ &\quad Hamming\_distance(y, y') \leq Hamming\_Weight\}. \end{aligned}$$

Partial Traversal in Phases

Using our pruning methods for the BDD $TODO$ and for the transition relation TR we obtain an algorithm for FSM traversal which proceeds in rounds and phases. The algorithm is illustrated in Figures 2 and 3.

In summary, the complete algorithm proceeds in $\lceil \log(cut\_depth) \rceil + 1$ phases. In each phase we work with a constant Hamming weight to restrict the "length of edges" in the transition relation. Each phase is divided into rounds. In each round, depending on the choice of the condition *preselect*, we process a different subspace of the total state space until no new states can be reached in this subspace. In each round a pruned transistion relation TR' is chosen dynamically.

Procedure **iterate_until_converge** (see Figure 2) performs a single round of the algorithm. It performs a fixed point iteration starting from a set $TODO$ of states using the pruned transition relation $TR' = TR \cdot preselect \cdot select$. All reached states are collected in set $Total\_REACHED$. Since *iterate_until_converge* starts image computations only from states fulfilling condition *preselect*, we have to collect states which are reached, but not yet processed by image computations, in a new set $TODO$ (lines 4, 12).

Figure 3 gives an overview of the whole FSM traversal algorithm: We start the first phase with Hamming weight 1 to compute the selectors *select* and *preselect* (lines 19, 25). Now we iterate in procedure **iterate_until_converge** the image computation until no new states are reached assuming selectors *select* and *preselect* (line 26). This process is repeated until the set $TODO$ provided by **iterate_until_converge** will become empty (loop of lines 23–28).

When $TODO$ is empty, we are not finished however, since we used a pruned transition relation with only "short edges". Now we have to enter a new phase: We increase $Hamming\_Weight$ (line 30), which restricts the selection of edges to be included in the pruned transition relation, now allowing also longer edges. For each phase we double the constant $Hamming\_Weight$ and we repeat the process until $Hamming\_Weight$ is maximal, i.e. until it equals the number *cut_depth* of state variables (loop of lines 21–33). Finally we have accumulated all reachable states in $Total\_REACHED$.

Experimental results in Section 4 prove that the order in which we visit new reached states in our distance driven traversal is really efficient to reduce peak sizes in memory consumption, which occur for the straightforward BFS based traversal.

Furthermore, also the runtime behaviour is improved. Using our special method to prune the transition relation TR we also succeed in achieving Goal 3: If we can assume that corresponding

[2]For reasons of efficiency the Hamming weight of the states is only considered for the first *cut_depth* state variables here.

Circuit	|TR|	Depth	|Reached|	#Reached	Original Method		Distance Guided	
					Peak Size	Runtime	Peak Size	Runtime
furnace17	7,264	174	845	8.9 x 10^{19}	4.0 M	139	**0.2 M**	**8**
key10	9,426	151	17,179	1.1 x 10^{12}	3.8 M	165	2.1 M	**67**
over12	6,782	90	3,671	5.9 x 10^{16}	9.2 M	507	**1.2 M**	**32**
mmgt20	6,167	144	9,756	8.1 x 10^{31}	4.2 M	248	**0.7 M**	**30**
dme2-16	141,840	433	8,353	1.4 x 10^{18}	5.7 M	500	3.8 M	269
dpd75	7,409	371	4,396	4.1 x 10^{60}	5.2 M	766	**2.4 M**	**238**
ftp3	6,399	58	55,937	5.9 x 10^{8}	3.6 M	339	2.5 M	283

Table 1: FMCAD'98 benchmarks - monolithic transition relations

Circuit	Traversal Depth	#Reachable States
s1269	9	1.1 x 10^{9}
s3271	16	1.3 x 10^{31}
s3330	7	7.3 x 10^{17}
s4863	4	2.2 x 10^{19}

Table 2: Characteristics of ISCAS'89 benchmarks

current and next state variables are neighboured in the BDD variable order (which is usually true in FSM traversal applications), *preselect* and *select* depend only on the first $2 \cdot cutdepth$ variables in the variable order, such that cofactors of $TR' := TR \cdot preselect \cdot select$ with respect to $2 \cdot cutdepth$ variables (or more variables) will also occur as cofactors of TR. Since the recursive BDD synthesis procedures are always working with a same set of cofactors of TR, we achieve an efficient Computed Table usage leading also to small runtimes (see Section 4).

4 Experimental Results

In this section, experimental results on the traversal techniques introduced in this paper are presented and compared with standard traversal, partitioned traversals and partitioned traversals combined with activity profiling [5]. The executive machine for all measurements was an Ultra-II model 2170 workstation with 1 GByte main memory. For all presented measurements the memory limit was given by 800 MByte and 5,000 seconds runtime. In all tables improvements of more than 100 % are presented in bold face, all runtimes are given in seconds, peak sizes represent numbers of BDD nodes.

Table 1 contains runtimes and memory performance results for model checking traces first introduced in [15] for use as a comparison basis of different BDD packages. For these traces the relevant FSM information for performing reachability analysis has been extracted without any modifications of the synthesis process originally given. $|TR|$ denotes the number of BDDs nodes of the transition relation. *Depth* is the traversal depth of the FSM. The columns $|Reached|$ and *#Reached* denote the number of BDDs nodes for the reachable states set and the number of reachable states, respectively. The column *Original Method* denotes our competitor, a standard FSM traversal process provided by the CUDD package [6] fully exploiting the rich set of newly added features for version 2.3.0 (e.g. the death-row for delayed freeage of BDDs improv-

Circuit	PT	#Cluster	|TR|	Original Method		Activity Profiling		Distance Guided	
				Peak Size	Runtime	Peak Size	Runtime	Peak Size	Runtime
s1269	5,000	6	12,122	10.7 M	4,596	0.4 M	18	0.8 M	52
s3271	500	17	6,158	1.9 M	4,191	1.3 M	664	3.5 M	**329**
s3330	500	17	7,891	timeout	timeout	1.4 M	358	1.9 M	320
s4863	5,000	39	85,384	0.4 M	53	0.2 M	76	0.9 M	49
s1269	500	12	5,946	10.2 M	4,577	11.6 M	2,411	**1.3 M**	**109**
s3271	5,000	7	21,403	timeout	timeout	1.5 M	1,761	4.8 M	**545**
s3330	5,000	6	20,950	timeout	timeout	0.6 M	2,610	2.2 M	**798**
s4863	500	50	61,447	timeout	timeout	timeout	timeout	**5.6 M**	**350**

Table 3: ISCAS'89 benchmarks – partitioned transition relations

ing Computed Table efficiency). For our method (denoted by *Distance Guided*) we applied a "cutdepth" value of 8 variables.

When comparing the values presented in the Table 1, an average performance improvement of a factor of about 2.9 for the time performance can be noticed. For some traces, the runtimes even yield an improvement factor of upto 17 (*furnace17*). Large peak sizes can be avoided by our traversal thanks to the focus on compact state sets representation, yielding an average improvement factor of almost 3 concerning peak sizes. Again some of the benchmarks yield results outstandingly better than the average value, e.g. *over12* and *mmgt20* with improvement factors of about 7.

A major problem when performing reachability analysis relies on the fact that in many cases it is not feasible to even construct the initial problem, i.e. the transition relation monolithically. Therefore the transition relation needs to be build using a conjunctive or disjunctive partitioning. In the following we will underline the fact, that our approach yields adequate results for non-monolithic transition relations too.

As underlying software platform for the following series of experiments the traversal tool *PdTrav 1.2* provided at [6] was used. Table 2 gives the values for the traversal depth as well as the number of reachable states for the ISCAS'89 benchmarks used for our measurements. It needs to be mentioned that the benchmarks *s1512*, *s3384* and *s5378* were excluded from the tables since, independent of the approach considered here, they did not finish calculations either due to given memory or runtime limit when using a fixed variable ordering. All initial variable orderings used were provided by [6].

In Table 3 we present memory and runtime comparisons of three traversal methods all implemented in the *PdTrav 1.2* traversal tool. *Original Method* denotes a straighforward BFS based traversal, *Activity Profiling* shows the results for the approach presented in [5]. The usage of this method demands the setting of several parameters (e.g. pruning threshold and heuristics, number of iterations for the learning phase, choice of image computation during learning phase and next FROM set selection). For the sake of simplicity we used the parameter settings separately provided for each benchmark [6] and applied the available scripts. The scripts also provide a suggested clustering for the transisition relation. The partitioning threshold (*PT*) is given in column 2 of the table. The corresponding number of clusters (*#Cluster*) and the size of the shared BDDs representing the transition relation ($|TR|$) are presented in columns 3 and 4. To underline the quality of our results, i.e. giving an impression of the stability of our method, we additionally present measurements for a second partitioning threshold (PT) (taken from the set $\{500, 5.000\}$), giving a different starting point for the same circuit. The results for the second set of PTs are shown in the lower half of Table 3. For our method (denoted by *Distance Guided*) we applied a "cutdepth" value of 14 variables. It should be mentioned that this is the only parameter that has to be set for distance driven FSM traversal. Moreover, we made the experience that the heuristics are robust against small changes in the "cutdepth" value.

Obviously, the choice of clustering, i.e. the representation of the initial problem has large impact on the overall complexity of the synthesis process. Benchmark *s4863* is a good example, being handled in less than one minute when having a "good" clustering, but on the other hand is not solvable for the *Original Method* and *Activity Profiling* within the given limits if not. As shown, among the approaches considered here, only the method *Distance Guided* is capable to partly overcome the handicap of a "bad" clustering and offers a reasonably robust behaviour. Overall, our approach outperforms a straightforward BFS based traversal both in BDD node peak sizes and runtimes for non-monolithic transition relations. For some of the peak sizes our results are slightly worse than the *Activity Profiling* results. Concerning runtimes, up to one exception, our runtimes are the best of all presented competitors.

5 Conclusions

We have presented a novel technique for symbolic FSM traversal which is based on a sequence of Hamming Distance guided partial traversals using a dynamic pruning of the transition relation and the state sets as well.

Our experimental results underline the quality of the approach, showing that Hamming Distance guided FSM traversal has much smaller memory requirements than straightforward BFS based traversal and significantly improved time performance. Furthermore, it also compares favorably to more sophisticated methods, like partitioned traversal combined with activity profiling.

As part of ongoing work, we are currently investigating the chances and influences of an automatical adjustment and variation of the "best" cut depth during the reachability process. Since for non-monolithic transition relations the variable support for all partitions can vary a lot (e.g. dependent on the methods for clustering), another point of great interest is the heuristical choice of the "cut depth" for each transition relation partition independently.

References

[1] K.S. Brace, R.L. Rudell, and R.E. Bryant. Efficient implementation of a BDD package. In *Design Automation Conf.*, pages 40–45, 1990.

[2] R.E. Bryant. Graph - based algorithms for Boolean function manipulation. *IEEE Trans. on Comp.*, 35(8):677–691, 1986.

[3] J.R. Burch, E.M. Clark, K.L. McMillan, and D.L. Dill. Sequential circuit verification using symbolic model checking. In *Design Automation Conf.*, pages 46–51, 1990.

[4] G. Cabodi, P. Camurati, L. Lavagno, and S. Quer. Disjunctive partitioning and partial iterative squaring: An effective approach for symbolic traversal of large circuits. *Design Automation Conf.*, 34:728–733, 1997.

[5] G. Cabodi, P. Camurati, and S. Quer. Improving symbolic traversals by means of activity profile. *Design Automation Conf.*, 36:306–311, 1999.

[6] G. Cabodi and S. Quer. http://www.polito.it/~quer/software.htm.

[7] H. Cho, G.D. Hachtel, E. Macii, B. Plessier, and F. Somenzi. Algorithms for approximate fsm traversal. In *Design Automation Conf.*, pages 25–30, 1993.

[8] O. Coudert, C. Berthet, and J.C. Madre. Verification of sequential machines based on symbolic execution. In *Automatic Verification Methods for Finite State Systems, LNCS 407*, pages 365–373, 1989.

[9] R. W. Hamming. Error detecting and error correcting codes. *Bell System Technical Jour.*, 9:147–160, April 1950.

[10] A. Narayan, A. Isles, J. Jain, R.K. Brayton, and A.L. Sangiovanni-Vincentelli. Reachability analysis using partitioned-robdds. In *Int'l Conf. on CAD*, pages 388–393, 1997.

[11] O.Coudert and J.C. Madre. A unified framework for the formal verification of sequential circuits. In *Int'l Conf. on CAD*, pages 126–129, 1990.

[12] K. Ravi and F. Somenzi. High-density reachability analysis. In *Int'l Conf. on CAD*, pages 154–158, 1995.

[13] H. Touati, H. Savoj, B. Lin, R.K. Brayton, and A.L. Sangiovanni-Vincentelli. Implicit enumeration of finite state machines using BDDs. In *Int'l Conf. on CAD*, pages 130–133, 1990.

[14] B. Yang. http://www-cgi.cs.cmu.edu/afs/user/bwolen/Web/software/.

[15] B. Yang, R.E. Bryant, D.R. O'Hallaron, A. Biere, O. Coudert, G. Janssen, R.K. Ranjan, and F. Somenzi. A performance study of BDD-based model checking. In *Proceedings of Formal Methods in Computer-Aided Design, LNCS 1522*, pages 255–289, 1998.