An Efficient Heuristic for State Encoding Minimizing the BDD Representations of the Transition Relations of Finite State Machines*

Riccardo Forth

University Halle-Wittenberg
Institute for Computer Science
Halle (Saale), D-06099, Germany
e-mail: forth@informatik.uni-halle.de

Paul Molitor

University Halle-Wittenberg
Institute for Computer Science
Halle (Saale), D-06099, Germany
e-mail: molitor@informatik.uni-halle.de

Abstract— An efficient representation of the (smoothed) transition relation of a synchronous finite state machine (FSM) speeds up the traversal based symbolic verification and the functional simulation of the FSM. When using reduced ordered binary decision diagrams (BDDs), dynamic reordering algorithms are applied in order to keep the sizes of the BDDs tractable. However, when FSMs are represented by BDDs, the state encoding can be used as an additional optimization criteria. In this paper, we present a new algorithm for state encoding of FSMs that minimizes the BDD representations of the corresponding (smoothed) transition relations. Experimental results show the approach to be very efficient.

I. Introduction

Sequential circuit optimization and analysis have been subject of intensive investigations for several decades. Symbolic traversal based verification and functional simulation of sequential circuits, e. g., can be speeded up much by using reduced ordered binary decision diagrams (BDDs) [2] as representation of the behavior of the circuits [1, 3, 4, 10, 12]. Using this representation, the automatic verification of some industrial sequential circuits with up to 10^{100} states could be handled [6].

Applying BDD based methods, the crucial point is to keep the BDDs tractable. Heuristics and dynamic reordering algorithms [9, 11] are applied in order to compute efficient variable orders. However, when finite state machines (FSM) are represented by BDDs, the state encoding can be used as an additional optimization criteria to minimize the BDDs [7, 8].

In this paper, we present some ideas for state encoding that minimizes the BDD representation of the (smoothed) transition relation of an FSM, which plays a dominant role in traversal based symbolic verification and functional simulation. To the best of our knowledge, this is the first approach for state encoding of FSMs targeting the BDD minimization of the corresponding transition relations published in literature, till now.

The paper is structured as follows. Section II introduces basic notations and definitions that are needed for the understanding of the paper. We present the new approach for state encoding in Section III and IV. In Section V, the algorithm is discussed with respect to some new theoretical results proved by Meinel and Theobald [8]. This section also contains detailed statistical experiments. The paper closes with some conclusions (Section VI).

*This work has been supported in part by DFG grant Mo 645/4.

II. Preliminaries

We introduce basic notations and definitions that are needed for the understanding of the paper.

A. Binary Decision Diagrams

A *binary decision diagram* is a rooted directed acyclic graph $G = (V, E)$ with vertex set V containing two types of vertices, *non-terminal* and *terminal* vertices. A non-terminal vertex v has as label a variable $index(v) \in \{x_1, \ldots, x_n\}$ and two children $low(v)$, $high(v) \in V$. A terminal vertex v is labeled either by logical 0 or logical 1 and has no outpoing edge. A binary decision diagram is called *ordered* if each variable is encountered at most once on each path from the root to a terminal and the variables are encountered in the same order on all such paths. A binary decision diagram is called *reduced* if it does not contain vertices either with isomorphic sub-graphs or with both edges pointing to the same vertex. Since we work only with reduced ordered binary decision diagrams in the following, we briefly call them BDDs.

A BDD computes a Boolean function $f : \{0,1\}^n \to \{0,1\}$ in a natural manner: each assignment to the input variables $x_1, \ldots, x_n$ defines a unique path through the graph from the root to the terminals. The label of the terminal defines the value of the function on that input.

B. Representation of Boolean Relations

A Boolean relation $R \subseteq \{0,1\}^p \times \{0,1\}^q$ can be represented by its *characteristic Boolean function* $\chi_R : \{0,1\}^p \times \{0,1\}^q \to \{0,1\}$ which is defined by $\chi_R(\alpha, \beta) = 1$ if and only if $(\alpha, \beta) \in R$ holds. Obviously, this characteristic function χ_R is a representation of R and can be represented by a BDD.

C. BDD representation of finite state machines

Formal verification checks whether a design fulfills its golden specification. Thus, the encoding of the input and output alphabet is given. By this, a deterministic *finite state machine* (FSM) M can be defined by a 3-tuple (S,δ,λ) where S is the set of *states*, $\delta : S \times \{0,1\}^n \to S$ is the *next-state function*, which determines the next state in dependency of the present state and the assignment of the input variables, and $\lambda : S \times \{0,1\}^n \to \{0,1\}^m$ is the *output function* which determines the assignment of the output variables.

A FSM M can be represented by the BDD of the characteristic function $\chi_{R(M)}$ of the relation $R(M) \subseteq S \times \{0,1\}^n \times \{0,1\}^m \times S$ defined by $(x,i,o,y) \in R(M)$ if and only if $\delta(x,i) = y$ and $\lambda(x,i) = o$ hold. For this, the states of M have to be encoded by an injective function $c : S \to \{0,1\}^r$ with $r \geq \lceil \log |S| \rceil$. Thus, the BDD size of a FSM does not only depend on the variable ordering but also on the state encoding c of the FSM.

In this paper, we mainly consider the machine's *smoothed transition relation* $\Delta(M) \subseteq S \times S$ defined by $(x,y) \in \Delta(M)$ if and only if there exists an input vector $i \in \{0,1\}^n$ under which machine M will transition from state x to state y, i. e., if $\delta(x,i) = y$ holds. This relation plays a determinant role in symbolic verification by FSM traversal [5]. As just discussed, the BDD representation of the characteristic function of the smoothed transition relation $\Delta(M)$ does not only depend on the variable order used but also on the state encoding c.

To make the diction easier, we will call the BDD of $\chi_{\Delta(M)}$ with respect to c the *BDD of the smoothed transition relation* $\Delta(M)$, in the following.

D. The State Encoding Problem

The state encoding problem we investigate in this paper is the problem of finding a code c of the states of a given FSM M such that the BDD of the smoothed transition relation $\Delta(M)$ is minimal.

III. Theoretical foundation of the approach

The new heuristic which we present in the following is based on the theoretical foundation that we present in this section. For a better understanding, we first only consider variable orders in which the present state variables $x_1,\ldots,x_r$ precede the next state variables $y_1,\ldots,y_r$. Furthermore, we use the following notations.

A. Notations

The part of the BDD of the smoothed transition relation belonging to the present state variables and the next state variables is named *upper part* and *lower* part of the BDD, respectively. The set of edges that have their startpoints in the upper part and their endpoints in the lower part of the BDD is called *cut* of the BDD.

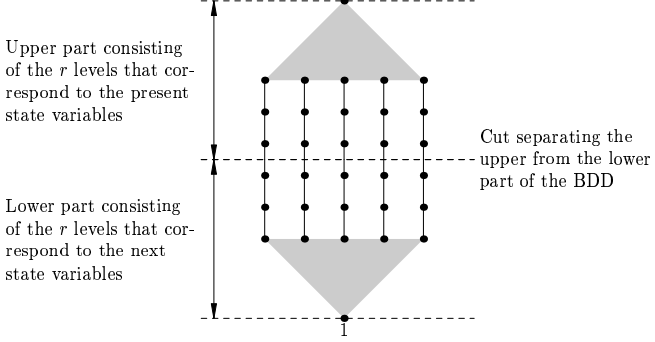

Fig. 1. Worst case BDD of the smoothed transition relation if STG is given by a path of length a. The BDD edges pointing to the 0-leaf have been omitted in the figure.

Furthermore, let $STG = (S,E)$ be the underlying directed graph of the state transition graph of the FSM M, i. e., in which the inscription concerning input and output assignments are deleted. Let us assume that STG does not contain any selfloops (otherwise, just delete them). A state s of STG that has at least one outgoing edge is called *from-state*, a node s that has at least one incoming edge is called *to-state*. Note that a node can be both.

B. Special cases of FSMs and correponding BDD representations

First, let us look at a very special case, namely, that the directed graph STG only consists of a path, say, of length a. Note that in this case the from-states are pairwise disjoint and the to-states, too. Because of this special structure, there are exactly a paths from the root to the 1-leaf of the BDD of $\Delta(M)$. Furthermore, the cut of the BDD has size a, i. e., there are exactly a BDD edges that cross the cut between the upper and lower part of the BDD. These observations serve as basis for proving an efficient upper bound on the size of the BDD of the smoothed transition relation of the FSM M under consideration. The worst case is that the BDD grows as fast as possible in the upper levels and maintains the width a as long as possible in the lower levels. For the size of the BDD of $\Delta(M)$ this results in the upper bound

$$2 \cdot a \cdot (r - \lceil \log a \rceil) + 2^{\lceil \log a \rceil + 1} + a + C$$

for some small constant C. Value r denotes the length of the state codes. The situation described is illustrated in Figure 1.

Now, let us consider the special case that STG is a tree. Let a_{from} be the number of from-states, i. e., the number of nodes of STG with at least one outgoing edge, and let a_{to} be the number of to-states, i. e., the number of nodes of STG with at least one incoming edge. The argumentation from above can be easily generalized and we obtain the upper bound

$$a_{from} \cdot (r - \lceil \log a_{from} \rceil) + 2^{\lceil \log a_{from} \rceil} + a_{to} \cdot (r - \lceil \log a_{to} \rceil) + 2^{\lceil \log a_{to} \rceil} + \min\{a_{from}, a_{to}\} + C.$$

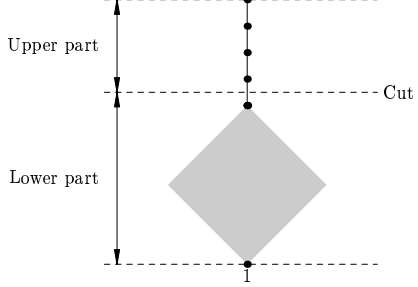

Fig. 2. BDD of the smoothed transition relation if $a_{from} = 1$ and $a_{to} = p-1$ hold. The BDD edges pointing to the 0-leaf have been omitted in the figure.

C. The approach

Now, we are ready for presenting our new state encoding algorithm. The central theme of our heuristic is to search for sub-trees of the state transition graph for which the upper bound given above is small.

The intuition behind the approach is that the size of the BDD can be influenced by the parameters a_{from} and a_{to}. Let us say that we have chosen a sub-tree consisting of p states. For this sub-tree, let us consider the attributes a_{from} and a_{to}. It is very easy to see that the inequations

$$p \leq a_{from} + a_{to} \leq 2 \cdot (p-1)$$

hold. We can show that for given p the upper bound from above is minimal, if $a_{from} + a_{to} = p$ and either $a_{from} = 1$ or $a_{to} = 1$ holds.

Thus, the algorithm tries to find sub-trees of the state transition graph with $a_{from} + a_{to} = p$ and $a_{from} = 1$ (or $a_{to} = 1$). A sketch of the BDD of a smoothed transition relation corresponding to such a sub-tree is shown in Figure 2, when using a variable order in which the present state variables $x_1, \ldots, x_r$ precede the next state variables $y_1, \ldots, y_r$. We assume that the variable order used is $x_1 < x_2 < \ldots < x_r < y_1 < y_2 < \ldots < y_r$. The BDD consists of $p-1$ paths from the root to the 1-leaf. At every level corresponding to the present state variables there is exactly one node in the BDD.

The states of such a sub-tree is encoded by our algorithm using a state encoding which locally leads to a mininal BDD. We distinguish three cases. If $a_{to} = 1$, things are simple, namely any encoding of the to-state is okay. Let us take value 0 as code of the from-state and value 1 as code of the to-state. For $a_{to} = 2$, the corresponding BDD is minimal if the codewords of the two to-states differ in only one bit. For $a_{to} \geq 3$ which is the general case, the BDD will become small if the codewords of the to-states have a long common prefix. A state encoding with this property is to encode the to-states by the usual binary representation of the numbers 0 to $a_{to} - 1$ which is called *standard state encoding* in literature.

Note that when proceeding to the interleaved variable order $x_1 < y_1 < x_2 < y_2 < \ldots < x_r < y_r$ usually used in literature, the state encoding for subtrees just described seems to be very efficient, too. If $p - 1 \leq 2^{r-j}$ holds, then shifting variable y_1 behind variable x_1, y_2 behind x_2, ..., and y_j behind x_j does not increase the BDD because the codewords of the to-states of the subtree have a common prefix of length at least j.

IV. The overall algorithm

The algorithm is a little bit more sophisticated than just described. Our approach first computes a set of sub-trees of the STG graph such that each node s of STG is root of exactly one such sub-tree. The nodes of these sub-trees are stored in $COV = \{T_1, \ldots, T_{|S|}\}$. The algorithm for this step is sketched in Figure 3.

```
1  Compute_the_subtrees (STG = (S,E))
2  begin
3     i := 1;
4     while S ≠ ∅
5           do Let s ∈ S be a node
6                 such that outdegree_S(s) is maximal;
7           T_i := {t ∈ S : (s,t) ∈ E} ∪ {s};
8           E := E \ {(s,t) : (s,t) ∈ E};
9           S := S \ {s};
10          i := i + 1;
11    od;
12 end
```

Fig. 3. Algorithm to compute a set of sub-trees with $a_{from} = 1$.

The encoding of these sub-trees is directed by the graph $G_{\subseteq}$, which remains fixed during the algorithm, and the graph $G_{\cap}$, which depends on the sub-tree T_i just under consideration.

- Graph $G_{\subseteq}$ is a directed graph $(\{T_0\} \cup COV, E_{\subseteq})$ whose nodes represent the sub-trees just computed and a node T_0 representing the complete set S of states of the FSM. There is an edge between node T_i and node T_j if and only if $T_j \subseteq T_i$ and there does not exist a node T_q with $T_j \subseteq T_q \subseteq T_i$.
- Graph $G_{\cap,i} = (V_{\cap,i}, E_{\cap,i})$ is an undirected graph. The set $V_{\cap,i}$ of nodes consists of the sub-trees T_j which are covered by T_i, i. e., $(T_i, T_j) \in E_{\subseteq}$. Set $E_{\cap,i}$ contains edge $\{T_{j_1}, T_{j_2}\}$ if and only if T_{j_1} and T_{j_2} are not disjoint.

The encoding process is mainly directed by the graph $G_{\subseteq}$. It encodes the sub-trees computed bottom up from the leaves to the root of $G_{\subseteq}$. We distinguish two cases. If the sub-tree T_i under consideration is a leaf of $G_{\subseteq}$, the states of the sub-tree are simply encoded as described above, i. e., the to-states are just numbered in the standard way. (The encoding can be changed by subsequent steps.) If the sub-tree T_i under consideration is not a leaf of $G_{\subseteq}$, we first compute the local graph $G_{\cap,i}$. Then, the sub-trees $T_j \in V_{\cap,i}$ which are already encoded because of the bottom-up approach are re-encoded. Here, the chronological order is determined by a procedure `Sort` that will be presented in detail in the next paragraph. The overall algorithm is summarized in Figure 4.

```
1  Encode (T_i)
2  begin
3    if T_i is a leaf of G_⊆ then return code(T_i); fi;
4    compute the local graph G_∩,i = (V_∩,i, E_∩,i);
5    while there exists T_j ∈ V_∩,i not already encoded
6          do Encode(T_j); od;
7
8    (T_j1, ..., T_jq) = Sort(G_∩,i);
9    code(T_i) := Re-encode(T_j1, ..., T_jq);
10   encode the states of T_i not yet encoded;
11
12   return code(T_i);
13 end
```

Fig. 4. Bottom-up algorithm for the encoding of the sub-trees.

The ordering $T_{j_1},\ldots,T_{j_q}$ of the nodes of the local graph $G_{\cap,i}$ is computed by procedure `Sort`. It looks for a longest path p_1 in $G_{\cap,i}$. The nodes of the path p_1 are numbered by 0 up to $|p_1|$ beginning at one endpoint of p_1. After deletion of the nodes of path p_1 from $G_{\cap,i}$, the next longest path p_2 is computed whose nodes are numbered by $|p_1|+1$ up to $|p_1|+|p_2|+1$. This process is continued until all the nodes of $G_{\cap,i}$ have obtained a number.

The `Re-encode` procedure re-encodes the states of the sub-trees $T_{j_1},\ldots,T_{j_q}$ covered by T_i beginning with the states of T_{j_1} up to the states of T_{j_q}. The re-encoding of the sub-tree T_{j_u} is done trying to maintain the order given by the encoding of T_{j_u} already computed by the bottom-up process. However, a state $s \in T_{j_u}$ is only re-encoded if it has not been re-encoded in a previous step, i. e., if $s \notin \cup_{t=1}^{u-1} T_{j_t}$. The procedure is sketched in Figure 5 and illustrated by means of an example in the next section.

```
1  Re-encode (T_j1, ...T_jq)
2  begin
3    i := 0;
4    for u = 1 to q do
5        forall state s ∈ T_ju according to code(T_ju) do
6          if s ∉ ∪_{t=1}^{u-1} T_jt then code(s) = i; fi;
7          i := i + 1;
8        od; od;
9    return ;
10 end
```

Fig. 5. Algorithm to re-encode the sub-trees $T_{j_1},\ldots,T_{j_q}$.

If T_{j_u} and $T_{j_{u+1}}$ contain common states, this approach raises hope that the encoding of these common states have long common prefixes with both the to-states of T_{j_u} and the to-states of $T_{j_{u+1}}$. Note that the algorithm guarantees minimal code length for the state encoding.

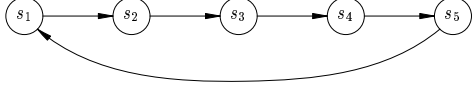

Fig. 6. State transition graph of a counter.

V. Discussion and Experimental Results

First, we discuss our algorithm when applied to counters. The problem of state encoding of counters has been investigated in depth by Meinel and Theobald [7, 8]. Then, we rate the quality of our approach by some statistical results obtained on the LGSynth91 benchmark set.

A. Application to counters

Before presenting detailed experimental results, we will draw our attention to some theoretical results proven by Theobald and Meinel [8]. For a class of counters, they derived exact BDD-sizes for important state encodings, presented sharp linear lower bounds and constructed worst-case encodings that lead to exponential-size BDDs. Specifically, they showed that the standard encoding leads to BDD representations of counters whose sizes are very close to the lower bounds proven.

It's easy to see that our algorithm does encode the states of these counters by standard encoding, too. This is guaranteed by the job done by the local graph. To demonstrate this statement (and to illustrate our algorithm by example), let us look at a counter consisting of 5 states. The state transition graph STG is shown in Figure 6.

Let us assume that procedure `Compute_the_subtrees` first generates sub-tree T_1 consisting of the states s_1 and s_2. Then, the global graph $G_\subseteq$ will consist of six nodes, namely the node T_0 representing the set $\{s_1,s_2,s_3,s_4,s_5\}$, the nodes $T_i = \{s_i, s_{i+1}\}$ for $i = 1,2,3,4$, and the node $T_5 = \{s_5\}$. The graph $G_\subseteq$ is shown in Figure 7. The encoding process first handles the sub-tree T_5 assigning code 0 to state s_5. Then the sub-trees T_1, T_2, T_3, and T_4 are encoded in a random order. As these four T_i only contains one next state, state s_i is locally encoded by 0 and state s_{i+1} is encoded by 1 (see Section IV). In the last iteration, set T_0 is encoded. Obviously, the local graph $G_{\cap,0}$ consists of the undirected path T_1,T_2,T_3,T_4. Thus, the re-encoding process first re-encode sub-tree T_1 trying to maintain the order s_1,s_2 already computed which results in code 0 for state s_1 and code 1 for state s_2. Then, the states of sub-tree T_2 are re-encoded. As state s_2 has already been re-encoded during the current call of procedure `Re-encode`, only state s_3 is handled. The algorithm assigns the next code, which is the value 2, to state s_3. This process is continued and results in the encoding that assigns value $i-1$ to state s_i. Thus, our algorithm computes the standard encoding for counters that has been proved to be very efficient in [8].

In an experiment, we have considered a counter with 2^9 states. We have randomly computed 2880 different state encodings for it which needs about one hour on our computer.

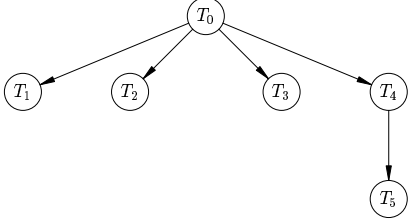

Fig. 7. Global graph $G_{\subseteq}$ of the counter.

The best and worst BDD with interleaved variable order $x_1 < y_1 < \ldots < x_9 < y_9$ generated during this run had size 1326 and 1444, respectively. Applying our heuristic using same variable order results in a BDD of size 107. This result experimentally gives evidence of the statement proved in [7] that nearly all state encodings of a counter with 2^n states, i. e., n state variables, lead to BDDs of size greater than $2^n/n$ with respect to the optimal variable orders. Our algorithm computes one of the few encodings which result in BDDs of size $O(n)$.

B. Statistical experimental results

As, to the best of our knowledge, no algorithm for state encoding targeting BDD minimization has been published in literature, till now, we present detailed statistical results.

To rate the quality of our heuristic, we have approximated the probability distribution of the BDD sizes of some LGSynth91 benchmark circuits with respect to the interleaved variable order by randomly computing state encodings for some hours. Figure 8 and 9 show the results obtained. In each of these diagrams the number of different state encodings (number of tests) which lead to the same BDD size (number of nodes) is pictured. The size of the BDD w.r.t. the interleaved variable order generated by our approach is marked by an arrow, respectively. In most cases, the algorithm presented in this paper computes a state encoding which seems to be nearly optimal with respect to the BDD size of the smoothed transition relation. Choosing state encodings randomly results in worse BDDs with high probability.

We have also applied the approach to general transition relations $R(M) \subseteq S \times \{0,1\}^n \times \{0,1\}^m \times S$ of finite state machines M. The statistical experimental results show that our approach applied to general transition relations is as efficient as applied to smoothed transition relations. Thus, the state encoding computed by our approach is suitable for functional simulation of FSMs, too.

VI. Conclusions

In this paper, we have presented a new algorithm for minimizing BDD representations of transition relations of finite state machines. In particular, in the case of counters, we have shown that the algorithm generates state encodings that have been proved to be very efficient by Meinel and Theobald [8]. Statistical results obtained on the LGSynth91 benchmark set prove the efficiency of the approach, in general.

References

[1] P. Ashar and S. Malik. Fast functional simulation using branching programs. In *IEEE Int'l Conf. on CAD*, pages 408–412, 1995.

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

[3] J. Burch, E. Clarke, D. Long, K. McMillan, and D. Dill. Symbolic model checking for sequential circuit verification. *IEEE Trans. on CAD*, 13(4):401–424, 1994.

[4] O. Coudert, C. Berthet, and J.C. Madre. Verification of sequential machines using Boolean functional vectors. In *Proceedings of IMEC-IFIP International Workshop on Applied Formal Methods for Correct VLSI Design*, pages 111–128, 1989.

[5] S.-Y. Huang and K.-T. Cheng. *Formal Equivalence Checking and Design Debugging*. Kluwer Academic Publishers, 1998.

[6] K. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.

[7] Chr. Meinel and T. Theobald. Local encoding transformations for optimizing OBDD-representations of finite state machines. Forschungsbericht 96-23, Fachbereich Mathematik und Informatik, Universitaet Trier, Germany, 1996.

[8] Chr. Meinel and T. Theobald. On the influence of the state encoding on BDD-representations of finite state machines. In *MFCS*, volume LNCS 1275 of *Lecture Notes on Computer Science*, pages 408–417. 1997.

[9] R. Rudell. Dynamic variable ordering for ordered binary decision diagrams. In *IEEE Int'l Conf. on CAD*, pages 42–47, November 1993.

[10] Chr. Scholl, R. Drechsler, and B. Becker. Functional simulation using binary decision diagrams. In *IEEE Int'l Conf. on CAD*, 1997.

[11] Chr. Scholl, D. Möller, P. Molitor, and R. Drechsler. BDD minimization using symmetries. *IEEE Trans. on CAD*, 18(2):81–100, February 1999.

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

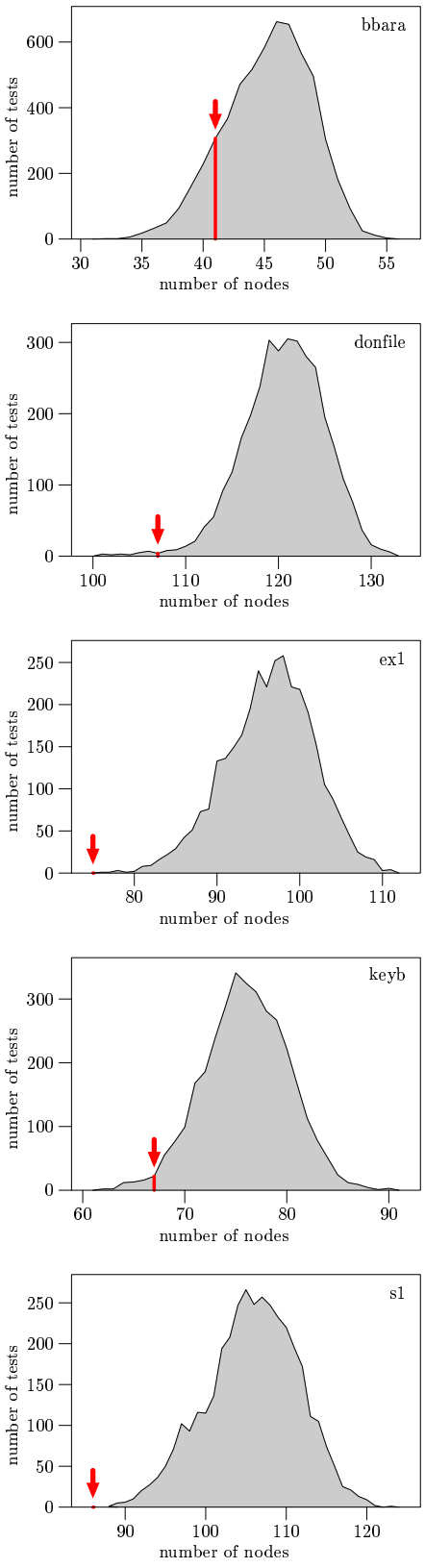

Fig. 8. Benchmark circuits ”bbara”,”donfile”,”ex1”,”keyb”, and ”s1”

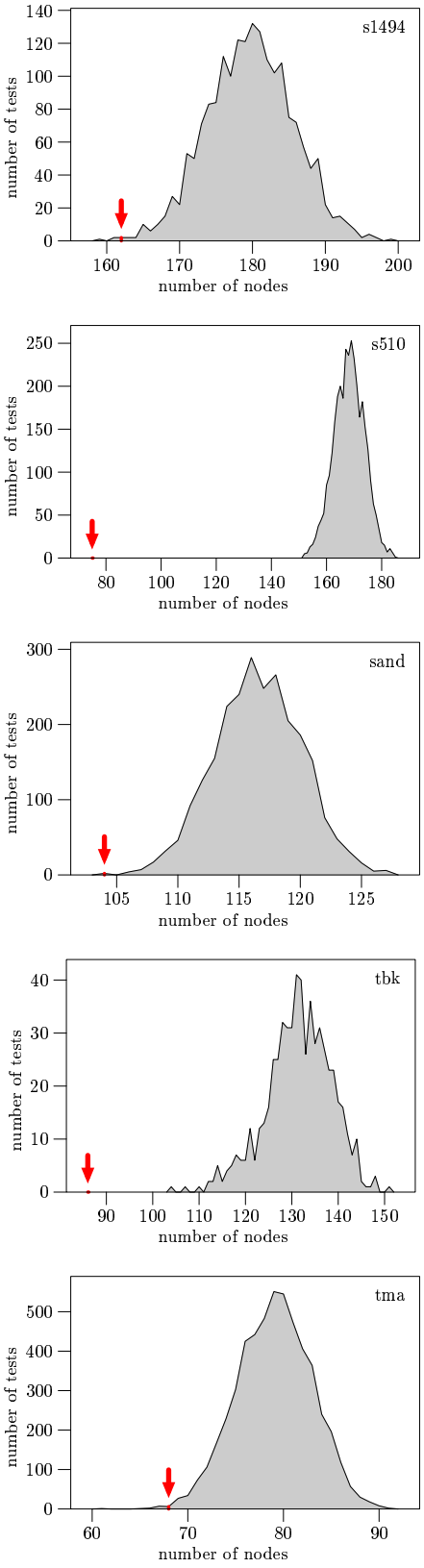

Fig. 9. Benchmark circuits ”s1494”,”s510”,”sand”,”tbk”, and ”tma”