Constraint-Based Optimal Testing Using DNNF Graphs*

Anika Schumann[1], Martin Sachenbacher[2], and Jinbo Huang[3]

[1] Cork Constraint Computation Centre, University College Cork, Ireland
[2] Institut für Informatik, Technische Universität München, Germany
[3] NICTA and Australian National University, Australia

Abstract. The goal of testing is to distinguish between a number of hypotheses about a system—for example, different diagnoses of faults—by applying input patterns and verifying or falsifying the hypotheses from the observed outputs. Optimal distinguishing tests (ODTs) are those input patterns that are most likely to distinguish between hypotheses about non-deterministic systems. Finding ODTs is practically important, but it amounts in general to determining a ratio of model counts and is therefore computationally very expensive.

In this paper, we present a novel approach to constraint-based ODT generation, which uses structural properties of the system to limit the complexity of computation. We first construct a compact graphical representation of the testing problem via compilation into *decomposable negation normal form*. Based on this compiled representation, we show how one can evaluate distinguishing tests in linear time, which allows us to efficiently determine an ODT. Experimental results from a real-world application show that our method can compute ODTs for instances that were intractable for previous approaches.

Keywords: Algorithms, applications, testing, DNNF graphs.

1 Introduction

Testing asks whether a system can be stimulated with input patterns, such that different hypotheses about the system can be verified or falsified from the observed output patterns. Applications include model-based fault analysis of technical systems, autonomous control (acquiring sensor inputs to discriminate among competing state estimates), and bioinformatics (designing experiments that help distinguish between different possible explanations of biological phenomena).

In many real-world applications of testing, the underlying models are non-deterministic; applying an input can lead to several possible outputs. Different

* This work was supported by Deutsche Forschungsgemeinschaft under grant SA 1000/2-1, the Science Foundation Ireland under the ITOBO Grant No. 05/IN/I886, and NICTA. The latter is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

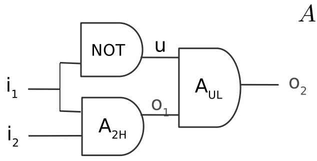
I.P. Gent (Ed.): CP 2009, LNCS 5732, pp. 731–745, 2009.

notions of testing with such non-deterministic models have been introduced. In the area of diagnosis, [13] introduced *definitely* and *possibly discriminating tests* (DDTs and PDTs) for systems modeled as constraints. For a DDT, the sets of possible outputs are disjoint and thus it will necessarily distinguish among hypotheses, whereas for a PDT, the sets partially overlap and thus it may or may not distinguish among hypotheses. In automata theory, [1] studied the analogous problem of generating *strong* and *weak distinguishing sequences* for non-deterministic finite state machines; for sequences of length at most $k \in \mathbb{N}$, this can be reduced to the problem of finding DDTs and PDTs [7].

For example, consider the network shown in Figure 1, which consists of one NOT component and two adders A_{2H} and A_{UL}. The former is high dominant upon receiving input $i_2 = H$ and the latter is low dominant upon receiving input $u = L$. Here we consider the two hypotheses M_1, M_2 that either both adders function normally, i.e. $M_1 = \{\text{NOT}, A_{2H}, A_{UL}\}$ or that both adders are faulty, i.e. $M_2 = \{\text{NOT}, A'_{2H}, A'_{UL}\}$. Then there exists no DDT, but two PDTs: $[-i_1, i_2]$ and $[-i_1, -i_2]$.

[8] introduced *optimal distinguishing tests* (ODTs), which generalize DDTs and PDTs by maximizing the ratio of distinguishing over non-distinguishing possible outcomes. In the example from Figure 1, the PDT $[-i_1, i_2]$ has a better distinguishing ratio than the PDT $[-i_1, -i_2]$, and is therefore an ODT for this example. Finding ODTs is important as it reduces the number of tests to be executed and the overall costs of the testing process. [8] proposed and analyzed a simple greedy-type algorithm to approximate ODTs, which in some real-world applications produces test inputs whose distinguishing ratios are close to those of ODTs.

In this paper, we present a novel search algorithm to compute ODTs (and thus DDTs and PDTs), which exploits structural properties of the model to limit the complexity of optimal test generation. Its main feature is a carefully constructed graph—through manipulation of logical theories and compilation into *decomposable negation normal form* (DNNF) [6]—that allows efficient computation of good upper bounds on ratios of model counts. These upper bounds are used



A_{2H}:

i_1	i_2	o_1
L	L	L
L	H	L
L	H	H
H	L	H
H	H	H

A'_{2H}:

i_1	i_2	o_1
L	L	L
L	H	L
H	L	H
H	H	H

NOT:

i_1	u
L	H
H	L

A_{UL}:

u	o_1	o_2
L	L	L
L	H	L
H	L	L
H	L	H
H	H	H

A'_{UL}:

u	o_1	o_2
L	L	L
L	H	L
H	L	L
H	H	L

Fig. 1. Circuit with two possibly faulty adders: A'_{2H} and A'_{UL}

to prune the search in a way motivated by a recent planning algorithm [9]. We show that our method can compute ODTs for instances that were intractable for previous approaches.

2 Background

Following the framework in [8,12,13], we assume that the system can be modeled as a *constraint satisfaction problem* (CSP), which is a triple $M = (\mathcal{V}, \mathcal{D}, \mathcal{C})$, where $\mathcal{D} = D(v_1) \times \ldots \times D(v_n)$ are the finite domains of finitely many variables $v_j \in \mathcal{V}$, $j = 1, \ldots, n$, and $\mathcal{C} = \{C_1, \ldots, C_m\}$ is a finite set of constraints with $C_i \subseteq \mathcal{D}$, $i = 1, \ldots, m$. We denote by X the set of all solutions, that is, assignments $\boldsymbol{x} \in \mathcal{D}$ to the variables such that all constraints are satisfied. That is, $X = \{\boldsymbol{x} \mid \boldsymbol{x} \in \mathcal{D},\ \mathcal{C}(\boldsymbol{x})\}$, where $\mathcal{C}(\boldsymbol{x})$ denotes $\boldsymbol{x} \in C_i$ for all $i = 1, \ldots, m$.

In addition, the system under investigation defines a set of *controllable* (input) variables $\mathcal{I}$ and a set of *observable* (output) variables $\mathcal{O}$. Formally, a hypothesis M for a system is then a CSP whose variables are partitioned into $\mathcal{V} = \mathcal{I} \cup \mathcal{O} \cup \mathcal{L}$, such that $\mathcal{I}$ and $\mathcal{O}$ are the input and output variables of the system, and for all assignments to $\mathcal{I}$, the CSP is satisfiable. The remaining variables $\mathcal{L} = \mathcal{V} \setminus (\mathcal{I} \cup \mathcal{O})$ are called internal state variables. We denote by $\mathcal{D}(\mathcal{I})$ and $\mathcal{D}(\mathcal{O})$ the cross product of the domains of the input and output variables, respectively: $\mathcal{D}(\mathcal{I}) = \times_{v \in \mathcal{I}} D(v)$ and $\mathcal{D}(\mathcal{O}) = \times_{v \in \mathcal{O}} D(v)$.

The goal of testing is then to find assignments of the input variables $\mathcal{I}$ that will cause different assignments of output variables $\mathcal{O}$ for different hypotheses. In the general case of non-deterministic systems, an input assignment can yield more than one possible output assignments. This case is frequent in practice; one reason is that in order to reduce the size of a model, it is common to aggregate the domains of system variables into small sets of values such as 'low', 'med', and 'high'; a side-effect of this abstraction is that the resulting models can no longer be assumed to be deterministic functions, even if the underlying system behavior was deterministic. Another reason is the test situation itself: even in a rigid environment such as an automotive test-bed, there are inevitably variables or parameters that cannot be completely controlled while testing the device.

In order to capture sets of outputs, for a given hypothesis M and an assignment $\boldsymbol{t} \in \mathcal{D}(\mathcal{I})$ to the input variables, we define the *output function* $\mathcal{X} : \mathcal{D}(\mathcal{I}) \to 2^{\mathcal{D}(\mathcal{O})}$ with $\boldsymbol{t} \mapsto \{\boldsymbol{y} \mid \boldsymbol{y} \in \mathcal{D}(\mathcal{O}),\ \exists\, \boldsymbol{x} \in X : \boldsymbol{x}[\mathcal{I}] = \boldsymbol{t} \wedge \boldsymbol{x}[\mathcal{O}] = \boldsymbol{y}\}$, where $2^{\mathcal{D}(\mathcal{O})}$ denotes the power set of $\mathcal{D}(\mathcal{O})$, and $\boldsymbol{x}[\mathcal{I}]$, $\boldsymbol{x}[\mathcal{O}]$ denote the restriction of the vector $\boldsymbol{x}$ to the input variables $\mathcal{I}$ and the output variables $\mathcal{O}$, respectively. Note that since M will always yield an output, $\mathcal{X}(\boldsymbol{t})$ is non-empty.

2.1 Distinguishing Tests

Non-deterministic models have given rise to the introduction of so-called *possibly* and *definitely distinguishing tests*, for short PDT and DDT, respectively [13]. The first type of test (PDT) might distinguish between hypotheses, as the sets of possible outputs partially overlap, whereas the second type (DDT) will necessarily do so, as the sets of possible outputs are disjunct:

Definition 1 (Distinguishing Tests). *Consider $k \in \mathbb{N}$ hypotheses $M_1, \ldots, M_k$ with input variables $\mathcal{I}$ and output variables $\mathcal{O}$. Let $\mathcal{X}_i$ be the output function of hypothesis M_i with $i \in \{1, \ldots, k\}$. An assignment $\boldsymbol{t} \in \mathcal{D}(\mathcal{I})$ to $\mathcal{I}$ is a* possibly distinguishing test *(PDT), if there exists an $i \in \{1, \ldots, k\}$ such that $\mathcal{X}_i(\boldsymbol{t}) \setminus \bigcup_{j \neq i} \mathcal{X}_j(\boldsymbol{t}) \neq \varnothing$. An assignment $\boldsymbol{t} \in \mathcal{D}(\mathcal{I})$ is a* definitely distinguishing test *(DDT), if for all $i \in \{1, \ldots, k\}$ it holds that $\mathcal{X}_i(\boldsymbol{t}) \setminus \bigcup_{j \neq i} \mathcal{X}_j(\boldsymbol{t}) = \mathcal{X}_i(\boldsymbol{t})$.*

For testing with non-deterministic automata models instead of logical theories or CSPs, there exists the analogous notion of so-called *weak* and *strong distinguishing sequences* [1,3]. Finding such sequences with a length bounded by some $k \in \mathbb{N}$ can be reduced to the problem of finding PDTs and DDTs, by unrolling automata into a constraint network using k copies of the automata's transition and observation relation [7]. Therefore, in this paper we restrict ourselves to models given as networks of finite-domain constraints.

2.2 Optimal Distinguishing Tests

Due to limited observability or a high degree of non-determinism, it is not uncommon that a DDT for the hypotheses does not exist, and one can instead only find PDTs. This motivates a *quantitative measure* for tests that refines and generalizes the previous notions of PDTs and DDTs. The intuition is that if we assume the possible outcomes (feasible assignments to the output variables) to be (roughly) equally likely, a PDT will be more likely to distinguish among two given hypotheses compared to another PDT, if the ratio of possible outcomes that are unique to a hypothesis versus all possible outcomes is higher. The notion of optimal distinguishing tests introduced in [8] formalizes this goal of finding tests that discriminate among two hypotheses as good as possible:

Definition 2 (Distinguishing Ratio). *Given a test input $\boldsymbol{t} \in \mathcal{D}(\mathcal{I})$ for two hypotheses M_1, M_2 with input variables $\mathcal{I}$ and output variables $\mathcal{O}$, we define $\Gamma(\boldsymbol{t})$ to be the ratio of feasible outputs that distinguish among the hypotheses versus all feasible outputs:*

$$\Gamma(\boldsymbol{t}) := \frac{|\mathcal{X}_1(\boldsymbol{t}) \cup \mathcal{X}_2(\boldsymbol{t})| - |\mathcal{X}_1(\boldsymbol{t}) \cap \mathcal{X}_2(\boldsymbol{t})|}{|\mathcal{X}_1(\boldsymbol{t}) \cup \mathcal{X}_2(\boldsymbol{t})|}$$

Γ is a measure for test quality that refines the notion of PDTs and DDTs: if Γ is 0, then the test does not distinguish at all, as both hypotheses lead to the same observations (output patterns). If the value is 1, then the test is a DDT, since the hypotheses always lead to different observations. If the value is between 0 and 1, then the test is a PDT (there is some non-overlap in the possible observations). Note that Γ is well-defined since for any chosen $\boldsymbol{t} \in \mathcal{D}(\mathcal{I})$, the sets $\mathcal{X}_1(\boldsymbol{t})$ and $\mathcal{X}_2(\boldsymbol{t})$ are non-empty.

An *optimal distinguishing test* (ODT) is one that has the maximal distinguishing ratio. For the example in Figure 1, the two PDTs $[-i_1, i_2]$ and $[-i_1, -i_2]$ for the hypotheses $M_1 = \{\text{NOT}, A_{2H}, A_{UL}\}$ and $M_2 = \{\text{NOT}, A'_{2H}, A'_{UL}\}$ have the following distinguishing ratios:

$\Gamma([-i_1, i_2]) = \frac{|\{o_1,o_2\},\{-o_1,o_2\}|}{|\{o_1,o_2\},\{-o_1,o_2\},\{-o_1,-o_2\}|} = \frac{2}{3}$ and
$\Gamma([-i_1, -i_2]) = \frac{|\{-o_1,o_2\}|}{|\{-o_1,o_2\},\{-o_1,-o_2\}|} = \frac{1}{2}$.
Therefore, the input $[-i_1, i_2]$ is an ODT for this example.

2.3 Deterministic DNNF

We briefly review graph-based representations of propositional theories. A propositional theory f is in negation normal form (NNF) [6] if it only uses conjunction (and, $\wedge$), disjunction (or, $\vee$), and negation (not, $\neg$), and negation only appears next to variables. An NNF is *decomposable* (DNNF) if conjuncts of every conjunction share no variables. A DNNF is *deterministic* (d-DNNF) if disjuncts of every disjunction are pairwise logically inconsistent. A d-DNNF is *smooth* if disjuncts of every OR node mention the same set of variables. In the rest of the paper we also assume that every variable of the logical theory appears in a smooth d-DNNF graph (this can always be ensured in polynomial time [6]). Such graphs represent each of its models by a subgraph G_s that satisfies the properties:

- every OR node in G_s has exactly one child,
- every AND node in G_s has the same children as it has in G, and
- G_s has the same root as G.

Henceforth we will denote subgraphs with these properties as *m-subgraphs.* Those that satisfy only the first two properties we will denote as *s-subgraphs.* Further we say that a subgraph is *labeled* by a literal l if it has a leaf node l, and that it is *consistent* with a partial assignment X to the d-DNNF variables if its labels are consistent with X.

d-DNNF graphs can be generated for propositional theories in conjunctive normal form (CNF) using the publicly available C2D compiler [5]. The complexity of this operation is polynomial in the number of variables and exponential only in the treewidth of the system in the worst case. Furthermore, given a DNNF graph G one can compute its projection $\Pi_\Sigma(G)$ on variable set Σ in linear time. Without impact on the computation time we therefore assume that M_1 and M_2 are defined over input and output variables only.

The left graph of Figure 2 illustrates a smooth d-DNNF graph G_N representing the models for the numerator of Γ for all test vectors for the example illustrated in Figure 1, i.e. $G_N = (M_1 \vee M_2) \wedge \neg(M_1 \wedge M_2)$. Thus it consists of the three models below that are each represented by a m-subgraph:

Model	Nodes of corresponding m-subgraph
$\{-i_1, i_2, o_1, o_2\}$	A1, O2, A3, O5, A6, $-i_1$, i_2, o_1, o_2
$\{-i_1, i_2, -o_1, o_2\}$	A1, O2, A3, O5, A7, $-i_1$, i_2, $-o_1$, o_2
$\{-i_1, -i_2, -o_1, o_2\}$	A1, O2, A4, A7, $-i_1$, $-i_2$, $-o_1$, o_2

Based on a smooth d-DNNF graph G the number of models $|G(X)|$ consistent with a partial assignment X to the d-DNNF variables can be determined by

Algorithm 1. Model counting with respect to instantiation X

$$\Lambda(N) = \begin{cases} 1 & \text{if } N \text{ is a leaf node consistent with X} \\ 0 & \text{if } N \text{ is a leaf node inconsistent with X} \\ \sum_i \Lambda(N_i) & \text{if } N = \bigvee_i N_i \\ \prod_i \Lambda(N_i) & \text{if } N = \bigwedge_i N_i \end{cases}$$

counting the number of consistent m-subgraphs in G. This is done by a bottom-up traversal of the graph that computes for each node N the number of consistent s-subgraphs $\Lambda(N)$ rooted in N. Hence, the $\Lambda(N)$ value of the root of the graph denotes the total number of consistent models represented by G. Algorithm 1 describes this linear time procedure [4]. An example of it is shown on the right of Figure 2. The numbers next to the nodes of that graph denote the $\Lambda(N)$ values computed by Algorithm 1 when running it with $X = [-i_1, i_2]$. Hence for the numerator of $\Gamma([-i_1, i_2])$ we get $|\mathcal{X}_1([-i_1, i_2]) \cup \mathcal{X}_2([-i_1, i_2])| - |\mathcal{X}_1([-i_1, i_2]) \cap \mathcal{X}_2([-i_1, i_2])| = 2$.

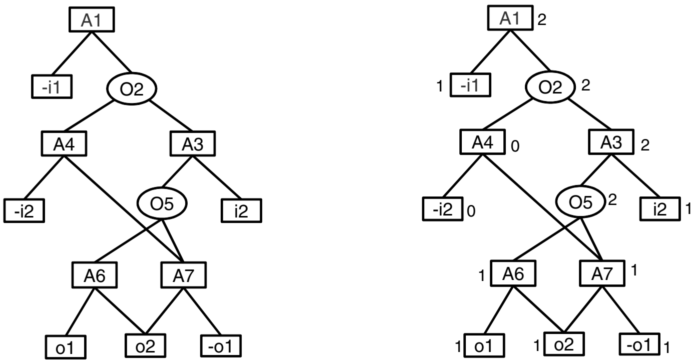


Fig. 2. Smooth d-DNNF graph G_N representing $(M_1 \vee M_2) \wedge \neg(M_1 \cap M_2)$ for the example shown in Figure 1. "A" and "O" indicate an And and an Or node, respectively. Numbers in non-leaf nodes are their identifiers. On the right, the numbers next to the nodes denote the $\Lambda(N)$ values computed by Algorithm 1 when running it with $X = [-i_1, i_2]$.

3 ODT Computation Using a d-DNNF Graph

The last section suggests that we can straightforwardly exploit the linear time d-DNNF based model counting algorithm for our ODT search. Similarly to the generation of a graph G_N representing the models of the numerator of the distinguishing ratio Γ we could also generate a graph representing the models of the denominator of Γ. Based on these two graphs we could then determine the Γ value for any test vector in linear time. However, in order to obtain the test vector with maximal distinguishing ratio, i.e. the ODT, such an approach requires the computation of the Γ values for every complete instantiation of a test vector (CITV). Since the number of test cases is exponential in the number of input variables this procedure would be infeasible for large applications.

This section presents a d-DNNF based branch-and-bound approach that does not require the Γ computation for every test vector. Its main component is a linear time algorithm that computes the upper Γ bound for any partial instantiation of a test vector (PITV). Such a procedure requires the simultaneous count of the models for the numerator and denominator of Γ based on d-DNNF graphs with identical structure. Since it is computationally very expensive to ensure that two independently generated d-DNNF graphs have the same structure we represent the whole ODT problem by a single d-DNNF graph that allows both: the computation of the numerator and that of the denominator of Γ for every test vector. The developed branch-and-bound approach then consists of the following building blocks that are each detailed in the following subsections:

- representation of the ODT problem as single d-DNNF graph $\mathcal{G}$,
- linear time algorithm that computes the Γ value for any CITV based on $\mathcal{G}$,
- linear time algorithm that computes an upper bound for the Γ value for any PITV based on $\mathcal{G}$, and
- an exhaustive search algorithm that iteratively sets input variables until either all variables are set and the Γ value for that CITV is computed or until the upper Γ bound for the PITV is not higher than the Γ value for a previously computed CITV.

3.1 Encoding the ODT Problem as Single d-DNNF Graph

We now describe how we can represent the ODT problem as a single d-DNNF graph $\mathcal{G}$ that allows the distinction of its models into those that belong to the numerator of Γ and those that do not. In order to achieve such a partitioning of nodes we introduce an auxiliary variable d and label every m-subgraph representing a model consistent with the numerator with the literal $-d$ and add the literal d to the remaining m-subgraphs. The latter comprises of the models $G_{\bar{N}} = M_1 \wedge M_2$ as stated in Definition 2. Thus, the propositional formula represented by $\mathcal{G}$ is defined as follows:

$$\begin{aligned} \mathcal{G} &= ((M_1 \vee M_2) \wedge \neg(M_1 \wedge M_2) \wedge \neg d) \vee (M_1 \wedge M_2 \wedge d) \\ &= (G_N \wedge \neg d) \vee (G_{\bar{N}} \wedge d) \end{aligned}$$

For our example we generate a graph $\mathcal{G}$ that represents the following models:

$$\begin{array}{ll} \{-d, -i_1, i_2, o_1, o_2\}, & \{d, -i_1, i_2, -o_1, -o_2\}, \text{ and} \\ \{-d, -i_1, i_2, -o_1, o_2\}, & \{d, -i_1, -i_2, -o_1, -o_2\}. \\ \{-d, -i_1, -i_2, -o_1, o_2\}, & \end{array}$$

The models on the left correspond to the numerator models of Γ and the ones on the right to the denominator models of Γ. The graph is illustrated in Figure 3. Its definition ensures that the distinguishing ratio can be obtained as follows:

$$\begin{aligned}
\varGamma(\boldsymbol{t}) &= \frac{|\mathcal{X}_1(\boldsymbol{t}) \cup \mathcal{X}_2(\boldsymbol{t})| - |\mathcal{X}_1(\boldsymbol{t}) \cap \mathcal{X}_2(\boldsymbol{t})|}{|\mathcal{X}_1(\boldsymbol{t}) \cup \mathcal{X}_2(\boldsymbol{t})|} \\
&= \frac{|\mathcal{X}_1(\boldsymbol{t}) \cup \mathcal{X}_2(\boldsymbol{t})| - |\mathcal{X}_1(\boldsymbol{t}) \cap \mathcal{X}_2(\boldsymbol{t})|}{(|\mathcal{X}_1(\boldsymbol{t}) \cup \mathcal{X}_2(\boldsymbol{t})| - |\mathcal{X}_1(\boldsymbol{t}) \cap \mathcal{X}_2(\boldsymbol{t})|) + |\mathcal{X}_1(\boldsymbol{t}) \cap \mathcal{X}_2(\boldsymbol{t})|} \\
&= \frac{|G_N(\boldsymbol{t})|}{|G_N(\boldsymbol{t})| + |G_{\bar{N}}(\boldsymbol{t})|} && \text{see definition of } G_N \text{ and } G_{\bar{N}} \\
&= \frac{|\mathcal{G}(\boldsymbol{t} \wedge \neg d)|}{|\mathcal{G}(\boldsymbol{t} \wedge \neg d)| + |\mathcal{G}(\boldsymbol{t} \wedge d)|} && \text{see definition of } G \\
&= \frac{|\mathcal{G}(\boldsymbol{t} \wedge \neg d)|}{|\mathcal{G}(\boldsymbol{t})|} && \text{since } d \vee \neg d \text{ evaluates to } \mathit{true}
\end{aligned}$$

The computation of the distinguishing ratio based on the later fraction is the subject of the next subsection.

3.2 Computation of $\varGamma(\boldsymbol{t})$ Based on DNNF Graph $\mathcal{G}$

We now show that we can compute the distinguishing ratio by resorting to the single graph $\mathcal{G} = (G_N \wedge \neg d) \vee (G_{\bar{N}} \wedge d)$. Although this algorithm has the same complexity than the one based on graphs G_N and $G_{\bar{N}}$, which we described earlier, we chose to present it since it will be the basis for the upper bound algorithm detailed in the next subsection. The latter requires the simultaneous computation of numerator $\varLambda_\alpha$ and denominator $\varLambda_\beta$ values of $\varGamma$. This can be done by a single bottom-up traversal of the graph as described in Algorithm 2. The model counting procedure itself is almost identical to the one shown in Algorithm 1. The only difference is that we set the numerator value $\varLambda_\alpha$ for the leaf node labeled d to 0. This results from the fact that Algorithm 2 is executed with respect to the instantiation $\boldsymbol{t}$ only, but that the numerator of $\varGamma$, i.e. $|\mathcal{G}(\boldsymbol{t} \wedge \neg d)|$, is defined with respect to instantiation $\boldsymbol{t} \wedge \neg d$. Hence we have to explicitly add the constraint $\neg d$, i.e. set $\varLambda_\alpha(d)$ to 0.

The bottom node label of graph $\mathcal{G}$ shown on the left of Figure 3 denotes the $\varLambda_\alpha$ and $\varLambda_\beta$ values for $\boldsymbol{t} = [-i_1, -i_2]$. From its root label the distinguishing ratio can be retrieved as formally stated in Theorem 1. Note that the distinguishing ratio of a node is only guaranteed to be correct if it is obtained from the $\varLambda_\alpha$ and $\varLambda_\beta$ values of its children. Resorting to their $\varGamma$ values would not be sufficient, since the numerator and denominator values for OR nodes need to be computed separately (see also Algorithm 1). For instance, consider node $O5$ shown on the left of Figure 3. Its $\varGamma$ value of $\frac{2}{3}$ cannot be obtained from the $\varGamma$ values of its children which are 0 (for node $A9$) and 1 (for nodes $A7$ and $A8$).

Theorem 1 (Test Assessment). *Let G be the root node of a smooth d-DNNF graph $\mathcal{G}$ representing the propositional formula $((M_1 \vee M_2) \wedge \neg(M_1 \wedge M_2) \wedge \neg d) \vee (M_1 \wedge M_2 \wedge d)$. Then $\varGamma(\boldsymbol{t}) = \varGamma(G, \boldsymbol{t})$ for any complete instantiation $\boldsymbol{t}$ of input variables.*

The correctness of this Theorem follows from the fact that $\mathit{\Gamma}(\boldsymbol{t}) = \frac{|\mathcal{G}(\boldsymbol{t} \wedge \neg d)|}{|\mathcal{G}(\boldsymbol{t})|}$ as derived in the previous subsection and from the basic d-DNNF model counting procedure shown in Algorithm 1.

Algorithm 2. Test assessment with respect to instantiation $\boldsymbol{t}$

For a leaf N we set:

$$\mathit{\Lambda}_\alpha(N) = \begin{cases} 0 & \text{if } N = d \text{ or} \\ & N \text{ is inconsistent with } \boldsymbol{t} \\ 1 & \text{otherwise} \end{cases} \qquad \mathit{\Lambda}_\beta(N) = \begin{cases} 1 & \text{if } N = d \\ \mathit{\Lambda}_\alpha(N) & \text{otherwise} \end{cases}$$

For remaining nodes we compute:

$$(\mathit{\Lambda}_\alpha(N), \mathit{\Lambda}_\beta(N)) = \begin{cases} (\sum_i \mathit{\Lambda}_\alpha(N_i), \\ \quad \sum_i \mathit{\Lambda}_\beta(N_i)) \text{ if } N = \bigvee_i N_i \\ (\prod_i \mathit{\Lambda}_\alpha(N_i), \\ \quad \prod_i \mathit{\Lambda}_\beta(N_i)) \text{ if } N = \bigwedge_i N_i \end{cases}$$

Then we compute the distinguishing ratio for each node:

$$\mathit{\Gamma}(N, \boldsymbol{t}) = \begin{cases} 0 & \text{if } \mathit{\Lambda}_\beta(N) = 0 \\ \frac{\mathit{\Lambda}_\alpha(N)}{\mathit{\Lambda}_\beta(N)} & \text{otherwise} \end{cases}$$

3.3 Upper Bounds for Partial Test Vectors

While the computation of the distinguishing ratio could have also been done based on two separate d-DNNF graphs, we now show how this single graph is essential to our new method for computing upper bounds on the distinguishing ratio based on a PITV $\boldsymbol{t}_p$. These bounds $\mathit{\Gamma}'(N, \boldsymbol{t}_p)$ are also obtained for each node N by a bottom-up traversal of graph $\mathcal{G}$. We will show that for any CITV $\boldsymbol{t}$ of $\boldsymbol{t}_p$ we have $\mathit{\Gamma}(N, \boldsymbol{t}) \leq \mathit{\Gamma}'(N, \boldsymbol{t}_p)$ and that we can therefore retrieve an upper bound of the distinguishing ratio from the $\mathit{\Gamma}'$ value of the root of $\mathcal{G}$.

Naturally, the search for an ODT will be the more efficient the tighter the upper bounds. The tightest possible bound for a node N is $\mathit{\Gamma}'(N, \boldsymbol{t}_p) = \mathit{\Gamma}(N, \boldsymbol{t})$ and indeed there is a well defined set of nodes for which we can obtain precisely that bound. This node set $\mathcal{N}_s$ is comprised of all those nodes whose $\mathit{\Gamma}$ value does not depend on a free input variable, i.e. on a variable in $\mathcal{I}_f$ which is not set by $\boldsymbol{t}_p$. This results from the fact that the $\mathit{\Gamma}$ value for a node is obtained from a bottom-up traversal of the graph. Hence it depends only on the truth value of its descendant variables, but not on the remaining variables of the graph. Let $\mathcal{N}_f$ denote the set of the remaining nodes, i.e. that are an ancestor of a free input variable.[1] Then resulting from $\mathcal{G}$ being smooth all children $\mathcal{N}_{childN}$ of an OR node N belong to the same node set, i.e. either $\mathcal{N}_{childN} \subseteq \mathcal{N}_s$ or $\mathcal{N}_{childN} \subseteq \mathcal{N}_f$. For instance, let us consider the computation of the $\mathit{\Gamma}(O2, [-i_2])$ value for our

[1] Formally, $N \in \mathcal{N}_f$, iff N is a leaf node of a variable $v \in \mathcal{I}_f$, or N is an AND or OR node that has at least one child $N_i \in \mathcal{N}_f$.

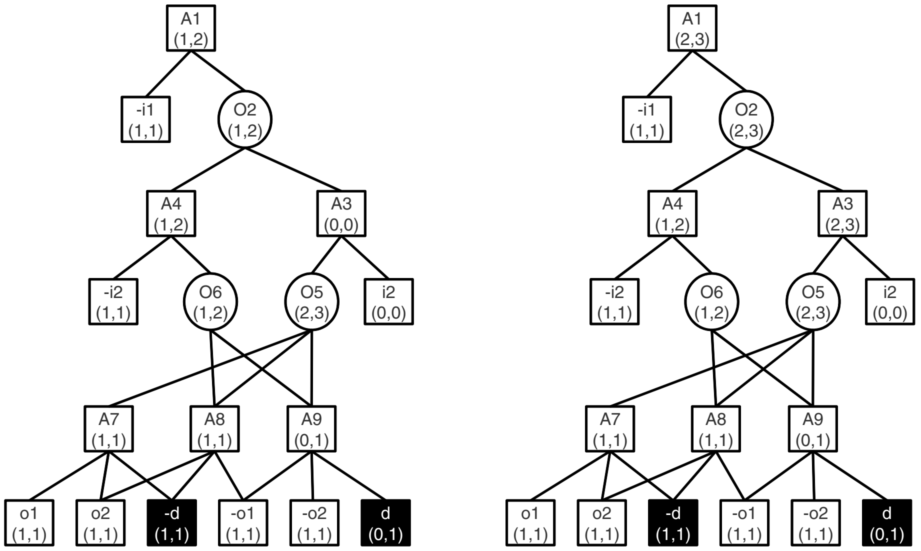


Fig. 3. Graph $\mathcal{G}$ for the example shown in Figure 1. On the left, the bottom labels $(\Lambda_\alpha(N), \Lambda_\beta(N))$ of the nodes refer to the test assessment values computed by Algorithm 2 when running it with $\boldsymbol{t} = [-i_1, -i_2]$ and on the right they to the upper bound values computed by Algorithm 3 when running it with $\boldsymbol{t}_p = [-i_1]$.

example (see left graph of Figure 3). Here we have $\mathcal{I}_f = \{i_1\}$ and thus neither one of the children $A3$ or $A4$ nor $O2$ itself is labeled by a free input variable. This means that $\Gamma(O2, [-i_2])$ is necessarily 1/2 regardless of how $\boldsymbol{t}_p = [-i_2]$ is completed, i.e. regardless of whether the CITV is $[-i_1, -i_2]$ or $[i_1, -i_2]$.

Hence we can compute the $\Gamma'(N, \boldsymbol{t}_p)$ value for any node $N \in \mathcal{N}_s$ using Algorithm 2. In addition, that algorithm can also be used to obtain the $\Gamma'(N, \boldsymbol{t}_p)$ value for any leaf node $N \in \mathcal{N}_f$. This results from the fact that the free variables are not inconsistent with $\boldsymbol{t}_p$ which implies that $\Lambda_\alpha(N)$ and $\Lambda_\beta(N)$ and therefore the $\Gamma(N, \boldsymbol{t}_p)$ value are set to the maximal value 1.

Moreover we show (see proof below) that also the Γ' value of an AND node $N \in \mathcal{N}_f$ can be obtained in analogy to Algorithm 2. Only for OR nodes with more than one child we need to apply a different procedure if the denominator value is larger than 0. This results from the fact that the Γ value of an OR node is retrieved from the separate summation of its children's numerator and denominator values, namely:

$$\Gamma(N, \boldsymbol{t}) = \frac{\Lambda_\alpha(N_1) + \Lambda_\alpha(N_2) \cdots + \Lambda_\alpha(N_j)}{\Lambda_\beta(N_1) + \Lambda_\beta(N_2) \cdots + \Lambda_\beta(N_j)}.$$

This means that the way in which the $\Lambda_\alpha(N_i)$ and $\Lambda_\beta(N_i)$ values of a free child N_i influence the $\Gamma(N, \boldsymbol{t})$ value depends not only on whether a CITV $\boldsymbol{t}$ will turn node N_i into a root of a consistent s-subgraph but also on the specific values of $\Lambda_\alpha(N_i)$ and $\Lambda_\beta(N_i)$. For instance, suppose N has two children N_1 and N_2 with

$\Lambda_\alpha(N_1) = 1$ and $\Lambda_\beta(N_1) = 2$. Depending on the values of N_2 the $\Gamma(N, \boldsymbol{t})$ value could be lower, equal, or higher to the one of $\Gamma(N_1, \boldsymbol{t})$=1/2:

$$\begin{aligned}
&\Gamma(N_1, \boldsymbol{t}) > \Gamma(N, \boldsymbol{t}) = \tfrac{1+1}{2+4} = \tfrac{1}{3} \text{ if } \Lambda_\alpha(N_2) = 1 \text{ and } \Lambda_\beta(N_2) = 4\\
&\Gamma(N_1, \boldsymbol{t}) = \Gamma(N, \boldsymbol{t}) = \tfrac{1+1}{2+2} = \tfrac{1}{2} \text{ if } \Lambda_\alpha(N_2) = 1 \text{ and } \Lambda_\beta(N_2) = 2 \text{ and}\\
&\Gamma(N_1, \boldsymbol{t}) < \Gamma(N, \boldsymbol{t}) = \tfrac{1+3}{2+4} = \tfrac{2}{3} \text{ if } \Lambda_\alpha(N_2) = 3 \text{ and } \Lambda_\beta(N_2) = 4.
\end{aligned}$$

Thus it is not possible to determine whether the $\Lambda_\alpha(N_i)$ and $\Lambda_\beta(N_i)$ values of a particular child should be considered for the upper bound computation of $\Gamma' = (N, \boldsymbol{t}_p)$ without looking at the specific values of its other children. Only the values of the child with maximal distinguishing ratio can be safely taken into account for the upper bound computation (see proof below). The procedure is described in Algorithm 3.

Algorithm 3. Upper bound with respect to instantiation $\boldsymbol{t}_p$

For a leaf N we have:

$$\Lambda'_\alpha(N) = \Lambda_\alpha(N) \qquad \text{and} \qquad \Lambda'_\beta(N) = \Lambda_\beta(N).$$

For remaining nodes we compute:

$$(\Lambda'_\alpha(N), \Lambda'_\beta(N)) = \begin{cases} (\Lambda_\alpha(N), \Lambda_\beta(N)) & \text{if } N = \bigvee_i N_i \text{ and } N \in \mathcal{N}_s \\ (\Lambda'_\alpha(N_m), & \text{if } N = \bigvee_i N_i \text{ and } N \in \mathcal{N}_f \text{ and} \\ \quad \Lambda'_\beta(N_m)) & \quad \Gamma(N_m, \boldsymbol{t}_p)\text{=} \max_i \Gamma(N_i, \boldsymbol{t}_p) \\ (\prod_i \Lambda'_\alpha(N_i), & \\ \quad \prod_i \Lambda'_\beta(N_i)) & \text{if } N = \bigwedge_i N_i \end{cases}$$

Then we compute the distinguishing ratio for each node:

$$\Gamma'(N, \boldsymbol{t}) = \begin{cases} 0 & \text{if } \Lambda'_\beta(N) = 0 \\ \frac{\Lambda'_\alpha(N)}{\Lambda'_\beta(N)} & \text{otherwise} \end{cases}$$

Note that Algorithm 2 can be regarded as a special case of Algorithm 3 where $\boldsymbol{t}$ is a complete instantiation of a test vector. It is precisely in this case that the computed value is guaranteed to be exact. Otherwise we certainly obtain an upper bound as formally stated in the following theorem:

Theorem 2 (Upper Bound). *Let $\boldsymbol{t}_p$ be a PITV and G the root node of the smooth d-DNNF graph $\mathcal{G}$ representing the propositional formula $((M_1 \vee M_2) \wedge \neg(M_1 \wedge M_2) \wedge \neg d) \vee (M_1 \wedge M_2 \wedge d)$. Then $\Gamma(\boldsymbol{t}) \leq \Gamma(G, \boldsymbol{t}_p)$ for any complete instantiation $\boldsymbol{t}$ of the free variables in $\boldsymbol{t}_p$.*

Proof
We prove this Theorem by showing that $\Gamma(N, \boldsymbol{t}) \leq \Gamma'(N, \boldsymbol{t}_p)$ for every CITV $\boldsymbol{t}$ of $\boldsymbol{t}_p$ and every node N. This is done by induction on the depth of graph $\mathcal{G}$. The

base case is straightforward. The distinguishing ratio $\Gamma(N, \boldsymbol{t}_p)$ of a leaf node N can either be 0 or 1. In order to ensure that $\Gamma'(N, \boldsymbol{t}_p)$ is an upper bound for $\Gamma(N, \boldsymbol{t}_p)$ it is therefore sufficient to set the $\Gamma'(N, \boldsymbol{t}_p)$ value for all leaves labeled by a free input variable to 1. This is precisely what Algorithm 3 does by setting the Λ_α and Λ_β values for these nodes to 1. Thus given a node N with children $N_1, \ldots, N_j$ we have the following induction hypothesis for the Λ_α and Λ_β values with respect to any CITV $\boldsymbol{t}$ of $\boldsymbol{t}_p$:

$$\frac{\Lambda_\alpha(N_1)}{\Lambda_\beta(N_1)} \leq \frac{\Lambda'_\alpha(N_1)}{\Lambda'_\beta(N_1)} \qquad \frac{\Lambda_\alpha(N_2)}{\Lambda_\beta(N_2)} \leq \frac{\Lambda'_\alpha(N_2)}{\Lambda'_\beta(N_2)} \qquad \cdots \qquad \frac{\Lambda_\alpha(N_j)}{\Lambda_\beta(N_j)} \leq \frac{\Lambda'_\alpha(N_j)}{\Lambda'_\beta(N_j)} \tag{1}$$

In the induction step we show that for any node whose children satisfy above hypothesis we also have $\Gamma(N, \boldsymbol{t}) \leq \Gamma'(N, \boldsymbol{t}_p)$. Here we distinguish two cases: (i) N is an AND node and (ii) N is an OR node.

(i) For an AND node N Algorithm 2 gives us:[2]

$$\begin{aligned}
&\Gamma(N, \boldsymbol{t}) = \frac{\Lambda_\alpha(N_1) \cdot \Lambda_\alpha(N_2) \cdots \ \cdot \Lambda_\alpha(N_j)}{\Lambda_\beta(N_1) \cdot \Lambda_\beta(N_2) \cdots \ \cdot \Lambda_\beta(N_j)} \\
\Rightarrow\ &\Gamma(N, \boldsymbol{t}) = \frac{\Lambda_\alpha(N_1)}{\Lambda_\beta(N_1)} \cdot \frac{\Lambda_\alpha(N_2)}{\Lambda_\beta(N_2)} \cdots \ \cdot \frac{\Lambda_\alpha(N_j)}{\Lambda_\beta(N_j)} \\
\Rightarrow\ &\Gamma(N, \boldsymbol{t}) \leq \frac{\Lambda'_\alpha(N_1)}{\Lambda'_\beta(N_1)} \cdot \frac{\Lambda'_\alpha(N_2)}{\Lambda'_\beta(N_2)} \cdots \ \cdot \frac{\Lambda'_\alpha(N_j)}{\Lambda'_\beta(N_j)} && \text{see induction hypothesis} \\
\Rightarrow\ &\Gamma(N, \boldsymbol{t}) \leq \Gamma'(N, \boldsymbol{t}_p) && \text{see Algorithm 3}
\end{aligned}$$

(ii) In case an OR node N is not labeled by a free variable we have $\Gamma(N, \boldsymbol{t}) = \Gamma'(N, \boldsymbol{t}_p)$ and therefore $\Gamma(N, \boldsymbol{t}) \leq \Gamma'(N, \boldsymbol{t}_p)$. To prove the other case we denote with $\Lambda'_{\alpha Max} = \Lambda_\alpha(N_{max})$ and $\Lambda'_{\beta Max} = \Lambda_\beta(N_{max})$ the corresponding values for the node with the maximal distinguishing ratio, i.e. $\Gamma'(N_{max}, \boldsymbol{t}_p) = \frac{\Lambda'_{\alpha Max}}{\Lambda'_{\beta Max}} = \max_i \frac{\Lambda'_\alpha(N_i)}{\Lambda'_\beta(N_i)}$. This gives us:

$$\frac{\Lambda_\alpha(N_1)}{\Lambda_\beta(N_1)} \leq \frac{\Lambda'_{\alpha Max}}{\Lambda'_{\beta Max}} \qquad \frac{\Lambda_\alpha(N_2)}{\Lambda_\beta(N_2)} \leq \frac{\Lambda'_{\alpha Max}}{\Lambda'_{\beta Max}} \qquad \cdots \qquad \frac{\Lambda_\alpha(N_j)}{\Lambda_\beta(N_j)} \leq \frac{\Lambda'_{\alpha Max}}{\Lambda'_{\beta Max}}$$

see induction hypothesis

$$\begin{aligned}
\Rightarrow \qquad & \Lambda_\alpha(N_1) \cdot \Lambda'_{\beta Max} \leq \Lambda_\beta(N_1) \cdot \Lambda'_{\alpha Max} \\
& \cdots \\
& \Lambda_\alpha(N_j) \cdot \Lambda'_{\beta Max} \leq \Lambda_\beta(N_j) \cdot \Lambda'_{\alpha Max}
\end{aligned}$$

$$\Rightarrow\ \Lambda_\alpha(N_1) \cdot \Lambda'_{\beta Max} \cdots + \Lambda_\alpha(N_j) \cdot \Lambda'_{\beta Max} \leq \Lambda_\beta(N_1) \cdot \Lambda'_{\alpha Max} \cdots + \Lambda_\beta(N_j) \cdot \Lambda'_{\alpha Max}$$

$$\Rightarrow \qquad (\Lambda_\alpha(N_1) \cdots + \Lambda_\alpha(N_j)) \cdot \Lambda'_{\beta Max} \leq \Lambda'_{\alpha Max} \cdot (\Lambda_\beta(N_1) \cdots + \Lambda_\beta(N_j))$$

[2] In case the Λ_β value of a child is 0 the $\Gamma(N, \boldsymbol{t})$ value is 0 and hence necessarily less or equal to $\Gamma'(N, \boldsymbol{t}_p)$.

$$\Rightarrow \qquad \frac{\Lambda_\alpha(N_1)\cdots+\Lambda_\alpha(N_j)}{\Lambda_\beta(N_1)\cdots+\Lambda_\beta(N_j)} \leq \frac{\Lambda'_{\alpha Max}}{\Lambda'_{\beta Max}}$$

$$\Rightarrow \qquad \frac{\Lambda_\alpha(N_1)\cdots+\Lambda_\alpha(N_j)}{\Lambda_\beta(N_1)\cdots+\Lambda_\beta(N_j)} \leq \Gamma'(N, \boldsymbol{t}_p)$$

$$\Rightarrow \qquad \Gamma(N, \boldsymbol{t}) \leq \Gamma'(N, \boldsymbol{t}_p) \qquad \text{see Algorithm 2} \ \blacksquare$$

3.4 ODT Computation

With the d-DNNF graph $\mathcal{G}$ and the linear time algorithms to compute the precise distinguishing ratio for a CITV and an upper bound for a PITV we have obtained the basis for our ODT search method. This consists of a branch-and-bound search over the input variables. Iteratively we set the input variables until either all variables are set and the precise Γ value is obtained or until the upper bound of the PITV is lower than the Γ value of a previously computed CITV.

Interestingly, if $\mathcal{G}$ has a certain structure (see below) we can obtain an ODT without a search by making use of the facts (i) that we can compute an upper bound Γ_{UB} for the distinguishing ratio of the ODT by running Algorithm 3 with respect to instantiation *true*, and (ii) that there is exactly one test vector $\boldsymbol{t}$ that is consistent with the resulting subgraph $\mathcal{G}_T$. The latter consists of all nodes from whose Λ'_α and Λ'_β values the upper bound Γ_{UB} was derived.[3]

Note, since $\mathcal{G}_T$ was obtained from running Algorithm 3 with respect to instantiation *true* it means that all input variables belong to the set of free ones, i.e. $\mathcal{I}_F = \mathcal{I}$. Hence every *OR* node $N \in \mathcal{N}_T$ with an input variable as descendant belongs to the set $\mathcal{N}_f$ and has therefore only one child in $\mathcal{G}_T$. Therefore, the labels of the input variables of $\mathcal{G}_T$ form a unique CITV $\boldsymbol{t}$ which is exploited in the following Theorem:

Theorem 3. *Let G be the root node of a smooth d-DNNF graph $\mathcal{G}$ representing the propositional theory $((M_1 \vee M_2) \wedge \neg(M_1 \wedge M_2) \wedge \neg d) \vee (M_1 \wedge M_2 \wedge d)$ and satisfying the following condition: No two children of an OR node are labeled by the same value of an input variable. Then the test vector $\boldsymbol{t}$ consistent with the subgraph $\mathcal{G}_T$ that is obtained from computing $\Gamma'(G, true)$ is an ODT.*

Proof
We prove this Theorem by contradiction. Suppose $\Gamma(G, \boldsymbol{t}) \neq \Gamma'(G, true)$. Since Algorithms 2 and 3 differ only in the Γ computation for an *OR* node in $\mathcal{N}_f$ which has more than one child the assumption implies that there is an *OR* node $N \in \mathcal{N}_f$ in graph $\mathcal{G}$ with at least two children that are both consistent with $\boldsymbol{t}$. This implies that N has at least two children labeled by the same value of an input variable which contradicts the condition of the theorem. $\blacksquare$

[3] Formally, a node $N \in \mathcal{N}_T$ is in $\mathcal{G}_T$, iff one of the following conditions is satisfied: (i) N is the root, (ii) N is a child of an *AND* node in $\mathcal{N}_T$, (iii) $N \in \mathcal{N}_s$ is a child of an *OR* node in $\mathcal{N}_T$, or (iv) $N \in \mathcal{N}_f$ is the child with the maximal Γ' value among the children of an *OR* node in $\mathcal{N}_T$.

For our example the condition of above theorem holds. The right graph of Figure 3 also shows the upper bound values computed by Algorithm 3 when running it with $\boldsymbol{t}_p = true$. Thus we obtained the ODT $\boldsymbol{t} = [-i_1, i_2]$ for our example in linear time. Note that the condition is certainly satisfied if $\mathcal{G}$ is a constrained d-DNNF graph [11]. Unfortunately, the compilation into constrained d-DNNF graphs is more complex and will therefore only be possible for small graphs.

4 Experimental Evaluation

We evaluated our DNNF-based testing method on a model of an automotive engine test-bed [10], which consists of three major components: engine, pipe, and throttle. The goal is to find leaks in the pipe by assigning three to four controllable variables, and observing three to four measurable variables. The problem has been turned into sets of discrete instances of varying sizes by applying different abstractions to the original mixed discrete-continuous model, and using a direct encoding from CSP to SAT [14].

Table 1. Results of ODT (left) and DDT (right) computation

inst.	#nodes	time
1a	58	0.04
2a	103	0.06
3a	161	0.09
4a	205	0.10
5a	329	0.20
6a	245	0.21
7a	362	0.40
8a	4766	5.41
9a	2654	36.6
10a	65063	414

inst.	#nodes	time
1b	66	0.06
2b	124	0.07
3b	191	0.10
4b	4865	14.7
5b	48238	396
6b	102817	1566
7b	–	–
8b	–	–
9b	–	–
10b	–	–

Table 1 shows the experimental results for computing ODTs and DDTs. For each instance, we report the size (number of nodes) of the DNNF graph computed, and the computation time in seconds on a Linux Dual-Core PC with 1GB of RAM. For instances that have DDTs, we compared our method with the most recent specialized DDT approach [12] based on quantified Boolean formulas and the QBF solver sKizzo [2]. That approach was able to solve instances 1b–3b and ran out of memory for the rest. In contrast, our ODT approach could also solve instances 4b–6b.

Our method is the first that computes exact ODTs; hence we cannot compare it directly with previous approaches. However, we used the greedy algorithm of [8] to compute approximate solutions for the same problem instances. This algorithm was only able to solve instances 1a–7a; that is, for instances 8a–10a we were able to compute an optimal solution where the previous approach could not even compute a suboptimal one.

5 Conclusion and Future Work

Optimal distinguishing tests generalize and refine the notion of possibly and definitely distinguishing and strong and weak tests for non-deterministic systems. Since computing ODTs can be computationally very expensive, previous work has focused on approximate solutions. We presented a new method to compute exact ODTs based on innovative ways of compiling the ODT problem into DNNF and computing upper bounds to prune a systematic search. Experimental results show that the method is able to compute both ODTs and DDTs for instances that were too large for previous methods. Thus our approach provides a significant improvement for many applications where output patterns can be assumed to be equally likely or where probabilities are not given. Where probabilistic models are available, our technique provides a baseline from which new techniques can be developed. Note that both model counting and weighted model counting can be done with the same linear complexity on d-DNNF. In fact, this is the basis for the recent compilation based approach to probabilistic reasoning, and will provide the basis also for extending our work to the probabilistic case.

References

1. Alur, R., Courcoubetis, C., Yannakakis, M.: Distinguishing tests for nondeterministic and probabilistic machines. In: Proc. ACM Symposium on Theory of Computing, pp. 363–372 (1995)
2. Benedetti, M.: skizzo: A suite to evaluate and certify QBFs. In: Proc. CADE 2005 (2005)
3. Boroday, S., Petrenko, A., Groz, R.: Can a model checker generate tests for non-deterministic systems? Elec. Notes Theor. Comp. Sci. 190, 3–19 (2007)
4. Darwiche, A.: On the tractable counting of theory models and its application to belief revision and truth maintenance. In: CoRR (2000)
5. Darwiche, A.: The c2d compiler user manual. Technical report, UCLA (2005)
6. Darwiche, A., Marquis, P.: A knowledge compilation map. Journal of Artificial Intelligence Research 17, 229–264 (2002)
7. Esser, M., Struss, P.: Fault-model-based test generation for embedded software. In: Proc. IJCAI 2007, pp. 342–347 (2007)
8. Heinz, S., Sachenbacher, M.: Using model counting to find optimal distinguishing tests. In: Proc. of CPAIOR (2009)
9. Huang, J.: Combining knowledge compilation and search for conformant probabilistic planning. In: Proc. ICAPS 2006, pp. 253–262 (2006)
10. Luo, J., Pattipati, K., Qiao, L., Chigusa, S.: An integrated diagnostic development process for automotive engine control systems. IEEE Trans. on Systems, Man, and Cybernetics 37(6), 1163–1173 (2007)
11. Pipatsrisawat, K., Darwiche, A.: A new d-dnnf-based bound computation algorithm for functional EMAJSAT. In: Proc. of IJCAI 2009 (2009)
12. Sachenbacher, M., Schwoon, S.: Model-based testing using quantified CSPs. In: ECAI 2008 Workshop on Model-based Systems (2008)
13. Struss, P.: Testing physical systems. In: Proc. AAAI 1994, pp. 251–256 (1994)
14. Walsh, T.: SAT vs. CSP. In: Proc. of CP 2000, pp. 441–456 (2000)