

Computing Cost-Optimal Definitely Discriminating Tests

Anika Schumann

Cork Constraint Computation Centre
University College Cork
Cork, Ireland
a.schumann@4c.ucc.ie

Jinbo Huang

NICTA and
Australian National University
Canberra, ACT 0200 Australia
jinbo.huang@nicta.com.au

Martin Sachenbacher

Institut für Informatik
Technische Universität München
85748 Garching, Germany
sachenba@in.tum.de

Abstract

The goal of testing is to discriminate between multiple hypotheses about a system—for example, different fault diagnoses—by applying input patterns and verifying or falsifying the hypotheses from the observed outputs. Definitely discriminating tests (DDTs) are those input patterns that are guaranteed to discriminate between different hypotheses of non-deterministic systems. Finding DDTs is important in practice, but can be very expensive (\sum_2^p -complete). Even more challenging is the problem of finding a DDT that minimizes the cost of the testing process, i.e., an input pattern that can be most cheaply enforced and that is a DDT. This paper addresses both problems. We show how we can transform a given problem into a Boolean structure in decomposable negation normal form (DNNF), and extract from it a Boolean formula whose models correspond to DDTs. This allows us to harness recent advances in both knowledge compilation and satisfiability for efficient and scalable DDT computation in practice. Furthermore, we show how we can generate a DNNF structure compactly encoding all DDTs of the problem and use it to obtain a cost-optimal DDT in time linear in the size of the structure. Experimental results from a real-world application show that our method can compute DDTs in less than 1 second for instances that were previously intractable, and cost-optimal DDTs in less than 20 seconds where previous approaches could not even compute an arbitrary DDT.

Introduction

Testing involves applying input patterns to a system such that different hypotheses about the system can be verified or falsified from the observed outputs. 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, which means that applying an input can lead to several possible outputs. One major source of non-determinism is model abstraction that aggregates the

domains of (continuous) system variables into sets of values such as “low,” “med,” and “high.” In such cases the resulting models can no longer be assumed to be deterministic functions, even if the underlying system behavior is deterministic. Another source of non-determinism is the testing 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 the device is being tested.

In the area of diagnosis, Struss (1994) introduced *definitely discriminating tests* (DDTs) for non-deterministic systems modeled as constraints. For a DDT, the sets of possible outputs are disjoint and thus it will necessarily distinguish among hypotheses. Finding DDTs is important in practice because it helps to reduce the number of input patterns that need to be applied, but it is in general very expensive (\sum_2^p -complete). In automata theory, Alur, Courcoubetis, and Yannakakis (1995) studied the analogous problem of generating *strong distinguishing sequences* for non-deterministic finite state machines, which for sequences of bounded length can be reduced to that of finding DDTs (Esser and Struss 2007).

Heinz and Sachenbacher (2009) introduced *optimal distinguishing tests* (ODTs), which generalize DDTs by maximizing the ratio of distinguishing over non-distinguishing outcomes. An efficient method for computing ODTs was presented in (Schumann, Sachenbacher, and Huang 2009), based on encoding the ODT problem into a Boolean formula, and compiling the formula into a graph in *decomposable negation normal form* (DNNF) (Darwiche 2001; Darwiche and Marquis 2002). The DNNF allows efficient derivation of good upper bounds on ratios of model counts, which are then used to prune a systematic search for ODTs.

This method is able to compute DDTs as a special case, and in that role exhibits much better performance than an earlier algorithm by Sachenbacher and Schwoon (2008) specialized for DDTs, as shown in (Schumann, Sachenbacher, and Huang 2009). Hence it represents the state of the art in DDT computation. As also shown in (Schumann, Sachenbacher, and Huang 2009), however, it can still run out of memory on instances of moderate size, which can be explained by the fact that the final DNNF graph is computed based on a number of auxiliary DNNF graphs, and the compilation can require a multitude of auxiliary variables to be introduced, which leads to slower computation and higher

memory requirements.

In this paper, we present a DNNF-based method that is adapted specifically to the problem of computing DDTs, and requires computation of just a single DNNF graph defined over the system variables. This representation is used to transform a quantified Boolean formula defining the DDT problem into a SAT problem that can be solved using an existing SAT solver. This leads to a significantly more efficient method for computing DDTs compared to that of (Schumann, Sachenbacher, and Huang 2009).

Furthermore, we present a generalization of DNNF-based test generation that takes into account the cost of enforcing a given input pattern. More specifically, we show how we can compute the values of input variables that can be most cheaply enforced and that constitute a DDT. This is achieved by constructing a DNNF graph, in two steps, that represents all DDTs and allows for the retrieval of a cost-optimal DDT in time linear in the size of the graph.

Experimental results from a real-world application show that our method can compute DDTs for instances that were previously intractable, and cost-optimal DDTs where previous approaches could not even compute an arbitrary DDT.

The remainder of the paper is organized as follows. We start with an example of the testing problem, before formally defining DDTs. We then review the notion of DNNF and a generic procedure for cost minimization with DNNF, in the context of our testing problem. The remaining sections present our new methods for computing both DDTs and cost-optimal DDTs, and an experimental evaluation of them, followed by the conclusion of the paper with a few words on future work.

Example

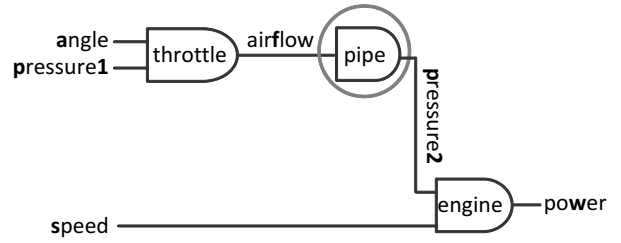
We start with a small example that motivates and illustrates the problems we are tackling and to which we will refer throughout the paper.

Consider the simplified model of an automotive engine’s air intake as shown in Figure 1. It consists of three components: engine, pipe, and throttle. There are three controllable input variables (angle a , pressure p_1 , speed s), two internal variables (airflow f , pressure p_2), and one observable output variable (power w). As a simplification, we abstract from the actual values for these variables and distinguish only the values high (H) and low (L) for each variable. The models of the three components are given at the bottom of Figure 1:

- the higher the angle and pressure, the more airflow will be generated by the **throttle**;
- the higher the airflow, the more pressure is built up in the **pipe**;
- the higher this pressure and the speed the more power is generated by the **engine**.

However, in case of a **broken-pipe**, no pressure is built up and no air-fuel-mixture will reach the engine.

The cost-optimal testing problem for this example is to find a test that can be most cheaply enforced and that will determine with certainty whether the pipe is broken, i.e., which of the following two hypotheses holds: $M_1 = \{\text{throttle,}$



throttle			pipe		engine			broken-pipe	
a	p_1	f	f	p_2	p_2	s	w	f	p_2
L	L	L	L	L	L	L	L	L	L
L	H	H	H	H	L	H	L	H	L
H	L	L			H	L	L		
H	L	H			H	L	H		
H	H	H			H	H	H		

Figure 1: Model of a gasoline engine’s air intake system with a possibly faulty pipe component.

pipe, engine}, $M_2 = \{\text{throttle, broken-pipe, engine}\}$. For this example, we assume that enforcing a value L is always cheaper than enforcing a value H. Hence the cheapest possible test is $a = L$, $p_1 = L$, and $s = L$. However, this is not a DDT because when stimulating the system with these input values we will certainly observe low power for both of the hypotheses and thus will not be able to distinguish them. Indeed, there are only two DDTs in this case: (L,H,H) and (H,H,H) (where we will observe low power only if the pipe is broken). Hence, (L,H,H) is the cost-optimal DDT for this example.

Definitely Discriminating Tests (DDTs)

Following the framework in (Heinz and Sachenbacher 2009; Schumann, Sachenbacher, and Huang 2009), 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 $\mathcal{D} = D(v_1) \times \dots \times D(v_n)$ are the finite domains of finitely many variables $v_j \in \mathcal{V}$, $j = 1, \dots, n$, and $\mathcal{C} = \{C_1, \dots, C_m\}$ is a finite set of constraints with $C_i \subseteq \mathcal{D}$, $i = 1, \dots, m$. We denote by X the set of all solutions, that is, assignments $\vec{x} \in \mathcal{D}$ to the variables such that all constraints are satisfied. More formally, $X = \{\vec{x} \mid \vec{x} \in \mathcal{D}, \mathcal{C}(\vec{x})\}$, where $\mathcal{C}(\vec{x})$ denotes $\forall i \vec{x} \in C_i$.

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

A DDT is an assignment to the input variables such that the sets of observable possible outputs for the different hypotheses are disjoint:

Definition 1 (Definitely Discriminating Test) Consider $k \in \mathbb{N}$ hypotheses M_1, \dots, M_k with input variables \mathcal{I} and output variables \mathcal{O} . Let \mathcal{X}_i be the output function of hypothesis M_i . An assignment $\vec{t} \in \mathcal{D}(\mathcal{I})$ is a *definitely discriminating test (DDT)* if $\forall i \mathcal{X}_i(\vec{t}) \setminus \bigcup_{j \neq i} \mathcal{X}_j(\vec{t}) = \mathcal{X}_i(\vec{t})$.

For testing with non-deterministic automaton models instead of logical theories or CSPs, there exists the analogous notion of *strong distinguishing sequences* (Alur, Courcoubetis, and Yannakakis 1995; Boroday, Petrenko, and Groz 2007). These are input sequences for a non-deterministic finite state machine, such that based on the generated outputs, one can determine the internal state either for all feasible runs of the machine. Finding such sequences with a length bounded by some $k \in \mathbb{N}$ can be reduced to the problem of finding DDTs, by unrolling the automaton into a constraint network using k copies of the automaton’s transition and observation relation (Esser and Struss 2007). Hence in this paper we restrict ourselves to models given as (static) networks of finite-domain constraints.

Decomposable Negation Normal Form (DNNF)

We briefly review graph-based representations of propositional theories (Darwiche and Marquis 2002). A propositional theory is in negation normal form (NNF) 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* if the conjuncts of every AND-node share no variables. A DNNF (decomposable NNF) is *smooth* if the disjuncts of every OR-node mention the same set of variables. Such a graph G represents 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 the term *subgraph* always denotes a graph satisfying all of the above properties.

DNNF graphs can be generated for propositional theories in conjunctive normal form (CNF) using the publicly available C2D compiler (Darwiche 2005), and smoothness can

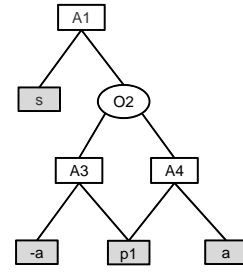


Figure 2: DNNF representing set of DDTs for the example in Figure 1. “A” and “O” indicate AND- and OR-node, respectively. Numbers in non-leaf nodes are their identifiers.

be ensured with negligible overhead. The complexity of this compilation is polynomial in the number of variables and exponential only in the treewidth of the system in the worst case. Figure 2 illustrates a smooth DNNF graph representing the set of DDTs for the example shown in Figure 1.¹

Cost Minimization with DNNF

Given a smooth DNNF representation of a propositional theory it is possible to obtain a cost-optimal model in time linear in the size of the DNNF as shown in Algorithm 1.² It takes as inputs a smooth DNNF graph and the cost values $c(l)$ for each of its literals. The cost-optimal model, i.e., the subgraph for which $\sum_i c(l_i)$ is the lowest, is obtained by a bottom-up propagation of cost values where each OR-node is labeled by the lowest cost value of its children and each AND-node is labeled by the sum of the cost values of its children.

For example, let us assume the following costs for enforcing values of the input variables in the example of Figure 1:

$$\begin{array}{lll} c(-a) = 5 & c(-p_1) = 10 & c(-s) = 7 \\ c(a) = 10 & c(p_1) = 20 & c(s) = 15 \end{array}$$

where $c(-x)$ (resp. $c(x)$) denotes the cost for enforcing the value L (resp. H) on variable x . A cost-optimal model for this example can then be found by propagating these cost values using Algorithm 1 as shown in Figure 3 (left). This model is given by the leaves of a *cost-optimal subgraph*. The latter graph is retrieved by traversing the DNNF top-down such that for each OR-node a child with the lowest cost value is kept and for each AND-node all children are kept. Figure 3 (right) illustrates the cost-optimal subgraph. Hence a cost-optimal model for this example is $(-a, p_1, s)$. Its cost is retrieved from the cost label of the root, 40 in this case.

Note that a cost-optimal subgraph may only give a partial

instantiation of the inputs, for example: $^{15} s \quad ^{20} p_1$ In

¹Note that this graph is also *deterministic*, i.e., the disjuncts of every OR-node are pairwise logically inconsistent. Although not required in this work, this property is always enforced by C2D.

²This is a variation of the minimization procedure described in (Darwiche 2001), which labels leaves with 0 and 1 only.

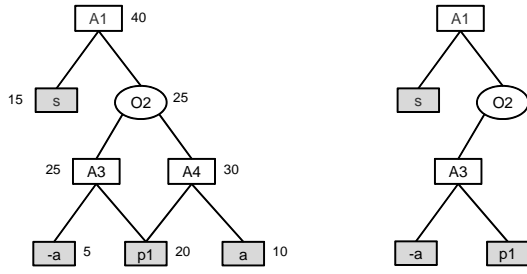


Figure 3: Propagation of cost values for the DNNF graph shown in Figure 2 on the left and a cost-optimal subgraph on the right.

that case a cost-optimal DDT is obtained by simply enforcing the cheapest value on each remaining variable, and its cost is obtained by adding the costs of these enforced values to the cost label of the root of the subgraph.

Algorithm 1 Cost Minimization with Smooth DNNF

$$c(N) = \begin{cases} c(l) & \text{if } N \text{ is a leaf node representing} \\ & \text{literal } l \\ \min_i c(N_i) & \text{if } N = \bigvee_i N_i \\ \sum_i c(N_i) & \text{if } N = \bigwedge_i N_i \end{cases}$$

Boolean Formulation of DDTs

In order to exploit the above efficient minimization procedure for retrieving a cost-optimal DDT, we aim to compile the DDT problem into smooth DNNF. Intuitively, this requires a propositional theory that encodes input vectors such that it is not possible for the corresponding outputs based on the different hypotheses to overlap. Such a theory in DNNF can be obtained in a number of steps, described next.

Since each hypothesis M_i is a CSP, we may assume that each M_i is given as a logical theory in conjunctive normal form (CNF) via any known translation of CSP to SAT (Walsh 2000). Let M_1 and M_2 be the two hypotheses we wish to distinguish about a system with $(\mathcal{I}, \mathcal{O}, \mathcal{L})$ as input, output, and internal variables. As pointed out in (Sachembacher and Schwoon 2008), the existence of a DDT is characterized by the following quantified Boolean formula:

$$\exists \mathcal{I} \forall \mathcal{O} \forall \mathcal{L} \neg(M_1 \wedge M_2),$$

or equivalently,

$$\exists \mathcal{I} \neg(\exists \mathcal{O} \exists \mathcal{L} (M_1 \wedge M_2)),$$

which states that there is an input vector such that the system cannot behave in a way consistent with both hypotheses.

Removing the quantifiers and enlisting the projection operation, we obtain the following propositional theory (denoted F) over the input variables \mathcal{I} only:

$$F = \neg(\Pi_{\mathcal{I}}(M_1 \wedge M_2)),$$

where $\Pi_{\mathcal{I}}(X)$ is the projection of theory X on variables \mathcal{I} . The models of theory F are precisely the set of all DDTs.

Computing DDTs via SAT

We are now ready to tackle the two tasks described earlier: computing DDTs, and computing cost-optimal DDTs. The first requires one call to a DNNF compiler followed by one call to a SAT solver.

Since projection can be performed in linear time on DNNF (Darwiche 2001), a single call to a DNNF compiler suffices to put $\Pi_{\mathcal{I}}(M_1 \wedge M_2)$ (i.e., $\neg F$) in DNNF. This leverages the power of DNNF compilation to obtain a compact representation of a theory over the input variables \mathcal{I} only.

Figure 4 (left) depicts a result of this compilation, a DNNF graph $\mathcal{G}_{\bar{F}}$,³ for the example in Figure 1. Note that this graph does not need to be smooth, while the one described in the next section does.

What we actually need to pass on to a SAT solver is the negation of this graph, in CNF. This can be obtained using the well-known Tseitin translation (Tseitin 1968) coupled with DeMorgan’s laws, where the resulting CNF will contain auxiliary variables but have a size polynomial in the DNNF size. Tseitin’s translation consists of three steps:

- introduction of one auxiliary variable for each non-leaf node of the DNNF,
- generation of clauses associated with each new variable,
- collection of all clauses into a single CNF with an additional constraint that forces the root to be true.

With our example graph $\mathcal{G}_{\bar{F}}$, we will introduce three new variables: $O1, A2, O3$. Now if we impose negation at the root, apply DeMorgan’s laws to push negations down to leaves, and finally apply Tseitin’s translation, we obtain:

$$\begin{aligned} O3 &\leftrightarrow (a \wedge \neg a) \\ A2 &\leftrightarrow (p1 \vee \neg O3) \\ O1 &\leftrightarrow (s \wedge \neg A2) \\ O1 &\leftrightarrow \text{true} \end{aligned}$$

The corresponding clauses are shown in Figure 4 (right) and serve as a CNF encoding of the DDT problem, ready to be passed to a SAT solver.

³We have kept the trivial OR-node $(\neg a \vee a)$ in the graph for illustration purposes.

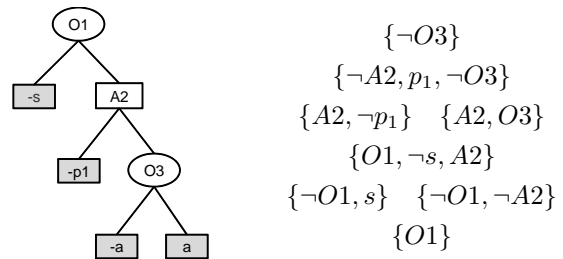


Figure 4: DNNF representing the non-DDTs from which the clauses on the right are derived, which encode the DDTs.

Computing Cost-Optimal DDTs

Models of the CNF encoding described in the previous section correspond to the set of DDTs, and given sufficient computational resources a SAT solver will be able to find one if it exists.

To enable efficient cost minimization, we now invoke once more the DNNF compiler to turn this CNF into smooth DNNF, to which Algorithm 1 can then be applied to efficiently obtain a cost-optimal DDT (or the set of all cost-optimal DDTs if desired). See Figure 3 for an example run of this algorithm.

In a nutshell, we are able to solve the cost-optimal DDT problem with two DNNF compilations and two linear-time procedures (one to generate a CNF for the negation of $\mathcal{G}_{\bar{F}}$ and one to retrieve a cost-optimal DDT).

To conclude our description of the two new DDT computation methods, Figure 5 summarizes and illustrates the overall high-level procedures for both.

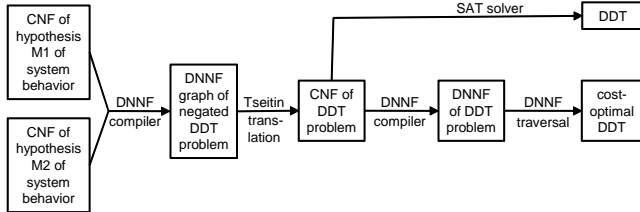


Figure 5: Overview of methods for DDTs and cost-optimal DDTs.

Experimental Evaluation

We evaluated our new testing methods on a model of an automotive engine test-bed (Luo et al. 2007), which consists of a throttle, a pipe, and an engine component. Compared to the simplified example in Figure 1, this model is more detailed as it contains additional variables and equations to also take into account the engine’s valve control.

The model was originally given in the form of a mixed discrete-continuous model. It has been turned into a set of 40 discrete problem instances of varying sizes by applying domain abstractions of different granularity to the continuous variables in the model, using the method described in (Lunze and Nixdorf 2001), and a direct encoding from CSP to SAT (Walsh 2000).

The goal is then, for each of the instances, to find leaks in the pipe component by assigning four controllable variables, and observing three measurable variables.

The experiments were performed on a Linux Dual-Core PC with 1GB of RAM using the DNNF compiler C2D (Darwiche 2005) and the SAT solver Tinisat (Huang 2007). Table 1 shows the computation times for finding a DDT using the previous ODT method (Schumann, Sachenbacher, and Huang 2009), a cost-optimal DDT using our new method (two compilations), and a DDT using our new method (compilation+SAT). Blank entries signify failure due to memory exhaustion.

inst.	ODT	coDDT	DDT	inst.	DDT
1	0.06	0.03	0.01	21	84.5
2	0.07	0.03	0.01	22	110
3	0.10	0.04	0.02	23	146
4	17.7	0.42	0.05	24	167
5	396	1.7	0.17	25	191
6	1566	7.4	0.44	26	246
7	-	19.8	0.82	27	286
8	-	35.1	1.31	28	345
9	-	154	2.42	29	432
10	-	213	3.78	30	518
11	-	250	5.94	31	615
12	-	502	8.18	32	670
13	-	880	11.7	33	792
14	-	1292	16.8	34	914
15	-	1969	24.6	35	1056
16	-	-	35.4	36	1139
17	-	-	33.0	37	1329
18	-	-	42.1	38	1492
19	-	-	58.0	39	1724
20	-	-	65.8	40	1894

Table 1: Computation times in seconds for previous method (ODT) and our new methods (cost-optimal DDT and DDT).

The results indicate improvements of several orders of magnitude over the previous state of the art. For example, the largest instance solved by ODT (# 6) took it 1566 seconds, but took our new DDT method only 0.44 seconds, and we were able to find a cost-optimal DDT in 7.4 seconds. The improvement allowed much larger instances to be solved with both our new methods, and particularly with the compilation+SAT method for DDTs.

As we briefly discussed in the introduction, the poorer performance of the ODT method appears to be partly due to the more complex DNNF compilations required. Figure 6 illustrates this issue in more concrete terms, where the number of nodes in the largest DNNF graph generated is shown for each instance, for the ODT method and our new DDT method.

For example, for instance 6 the DNNF has already over 100,000 nodes for the ODT method, but less than 3,000 nodes for the new method. Figure 6 does not show DNNF sizes for the second compilation required by our method for cost-optimal DDTs, but they are always smaller than in the first compilation except in the case involving instance 1.

Finally, that our method for DDTs scales much better than for cost-optimal DDTs is consistent with the difference in complexity between SAT and compilation. For an illustration of this difference, consider the fact that compilations produced by the C2D compiler are known to permit model counting in linear time (Darwiche and Marquis 2002)—model counting is known to be #P-complete while SAT is NP-complete.

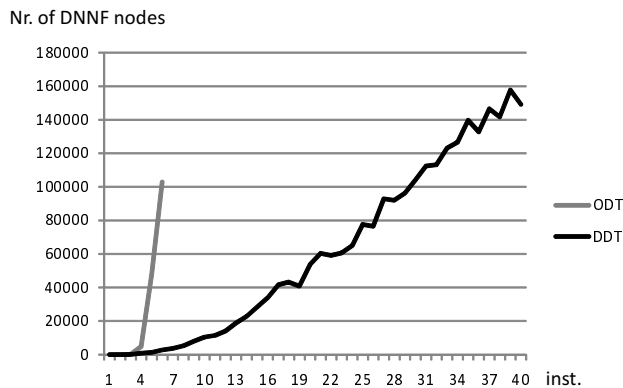


Figure 6: Number of nodes of the largest DNNF generated for each of the instances in Table 1.

Conclusion and Future Work

We have presented a new algorithm for computing DDTs that exhibits much better efficiency and scalability than previous approaches on a realistic benchmark. This was achieved by constructing a Boolean formula that encodes the complex DDT problem (Σ_2^P -complete) and that can be transformed into a SAT problem via DNNF compilation, which is often very efficient for real-world problems that have structure.

In addition we have studied the problem of computing DDTs that can be most cheaply enforced, a problem which is very relevant in practice but has not been considered before. We proposed an algorithm for this problem that involves one additional step of DNNF compilation such that cost-optimal DDTs can be retrieved from the final DNNF in time linear in the size of the DNNF. With this algorithm we were able to compute cost-optimal DDTs for instances where previous methods could not even find an arbitrary DDT.

The success of our new methods can be partly ascribed to the fact that they are better able to exploit structural properties of the system (compared with Sachenbacher and Schwoon 2008), or that they require less complex DNNF compilations (compared with Schumann, Sachenbacher, and Huang 2009).

Topics for future work include extending our approach to systems that admit possibly discriminating tests but not DDTs, and to systems whose non-determinism is quantified by a probabilistic model.

Acknowledgments

This work was supported by the Science Foundation Ireland under the ITOBO Grant No. 05/IN/I886, the Deutsche Forschungsgemeinschaft under grant SA 1000/2-1, 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.

References

- Alur, R.; Courcoubetis, C.; and Yannakakis, M. 1995. Distinguishing tests for nondeterministic and probabilistic machines. In *Proceedings of the ACM Symposium on Theory of Computing*, 363–372.
- Boroday, S.; Petrenko, A.; and Groz, R. 2007. Can a model checker generate tests for non-deterministic systems? *Electronic Notes in Theoretical Computer Science* 190:3–19.
- Darwiche, A., and Marquis, P. 2002. A knowledge compilation map. *Journal of Artificial Intelligence Research* 17:229–264.
- Darwiche, A. 2001. Decomposable negation normal form. *Journal of the ACM* 48(4):608–647.
- Darwiche, A. 2005. The C2D compiler user manual. Technical report, Comp. Sci. UCLA.
- Esser, M., and Struss, P. 2007. Fault-model-based test generation for embedded software. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI)*, 342–347.
- Heinz, S., and Sachenbacher, M. 2009. Using model counting to find optimal distinguishing tests. In *Proceedings of the Sixth International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR)*, 117–131.
- Huang, J. 2007. A case for simple SAT solvers. In *Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming (CP)*, 839–846.
- Lunze, J., and Nixdorf, B. 2001. Representation of hybrid systems by means of stochastic automata. *Mathematical and Computer Modeling of Dynamical Systems* 4:383–422.
- Luo, J.; Pattipati, K.; Qiao, L.; and Chigusa, S. 2007. An integrated diagnostic development process for automotive engine control systems. *IEEE Trans. on Systems, Man, and Cybernetics* 37(6):1163–1173.
- Sachenbacher, M., and Schwoon, S. 2008. Model-based testing using quantified CSPs. In *ECAI-08 Workshop on Model-based Systems*.
- Schumann, A.; Sachenbacher, M.; and Huang, J. 2009. Constraint-based optimal testing using DNNF graphs. In *Proceedings of the 15th International Conference on Principles and Practice of Constraint Programming (CP)*, 731–745.
- Struss, P. 1994. Testing physical systems. In *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI)*, 251–256.
- Tseitin, G. 1968. On the complexity of derivation in propositional calculus. *Studies in Constructive Mathematics and Mathematical Logic* 115–125.
- Walsh, T. 2000. SAT vs. CSP. In *Proceedings of the Sixth International Conference on Principles and Practice of Constraint Programming (CP)*, 441–456.