On Compiling System Models for Faster and More Scalable Diagnosis

Jinbo Huang and Adnan Darwiche
Computer Science Department
University of California, Los Angeles
Los Angeles, CA 90095
{jinbo, darwiche}@cs.ucla.edu

Abstract

Knowledge compilation is one of the more traditional approaches to model-based diagnosis, where a compiled system model is obtained in an off-line phase, and then used to efficiently answer diagnostic queries on-line. The choice of a suitable representation for the compiled model is critical to the success of this approach, and two of the main proposals have been Decomposable Negation Normal Form (DNNF) and Ordered Binary Decision Diagram (OBDD). The contribution of this paper is twofold. First, we show that in the current state of the art, DNNF dominates OBDD in efficiency and scalability for some typical diagnostic tasks. This result is based on a step-by-step comparison of the complexities of diagnostic algorithms for DNNF and OBDD, together with a known succinctness relation between the two representations. Second, we present a tool for model-based diagnosis, which is based on a state-of-the-art DNNF compiler and our implementations of DNNF diagnostic algorithms. We demonstrate the efficiency of this tool against recent results reported on diagnosis using OBDD.

Introduction

One of the well established approaches to model-based diagnosis is based on *compiling* a system model into some tractable propositional language, and then applying polynomial-time operations on the compiled representation to efficiently answer diagnostic queries. For the success of this compilation-based approach, the selection of a suitable target compilation language is critical. An early example appeared in (de Kleer 1990), where propositional sentences that modeled a device were compiled into their prime implicants. It was later observed, however, that prime implicant representations tended to be impractically large for many real-world devices (Forbus & de Kleer 1993).

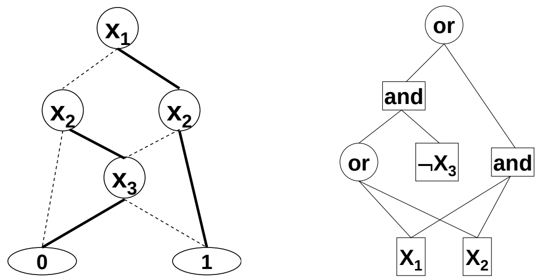
Ordered Binary Decision Diagrams (OBBDs) (Bryant 1986) are a tractable representation for propositional theories, and have been widely used in the field of formal verification. (Sztipanovits & Misra 1996) adopted OBDDs for the diagnosis of discrete-event systems, and there has since been an interest in using OBDDs also for model-based diagnosis of static systems (Torasso & Torta 2003; Torta & Torasso 2004). To harness the tractable operations offered by OBDDs, (Torasso & Torta 2003; Torta & Torasso 2004) proposed a novel algorithm that computes the minimum-cardinality diagnoses by an iterative procedure that involves a series of "cardinality filters" of tractable size. An effort has also been made to obtain better OBDD variable orderings by exploiting the causal structure of the system to be diagnosed.

In the present paper, we show that one can achieve better efficiency and scalability in model-based diagnosis by using a weaker representation known as Decomposable Negation Normal Form (DNNF) (Darwiche 2001). First of all, it is known that the DNNF language is a strict superset of, and strictly more succinct than, the OBDD language (Darwiche & Marquis 2002). This implies that (1) if a system model can be compiled successfully into OBDD, then it can also be compiled successfully into DNNF (since every OBDD is in DNNF), and (2) some system model may have a polynomial-size compilation in DNNF, yet not in OBDD.

This succinctness relation alone suggests that one can be better off with DNNF in terms of the time and space requirements for obtaining and storing the compiled system model. However, it does not say which representation will be more conducive to answering on-line diagnostic queries. In other words, it might be perfectly possible to have the system model in the less succinct OBDD, but then be able to perform on-line diagnosis more efficiently, because OBDD, being a stronger representation, supports more tractable queries than DNNF in general.

We show, via a step-by-step comparison, that this is not the case given the diagnostic algorithms currently available for both languages. Specifically, we show that the two typical diagnostic tasks—computing consistency-based diagnoses and those of minimum cardinality—have lower complexities for DNNF than for OBDD.

In light of these analytical results and recent practical advances in DNNF compilation, we present a software tool for diagnosis using DNNF. This tool is based on a state-of-the-art DNNF compiler (Darwiche 2004) and our implementations of DNNF diagnostic algorithms. We apply this tool to a diagnosis benchmark for which results have been recently reported on diagnosis using OBDD. The experiments indicate that the use of DNNF leads to faster compilation of the system model, a smaller representation for the system



(a) OBDD (b) DNNF

Figure 1: A propositional theory represented in OBDD and DNNF.

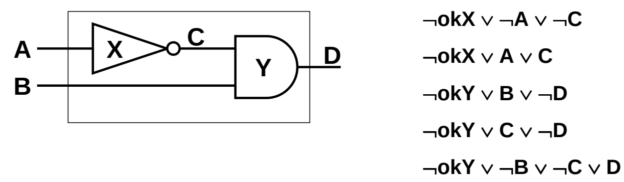


Figure 2: A circuit and its CNF encoding.

model, faster completion of individual diagnostic cases, and smaller representations for the sets of diagnoses computed.

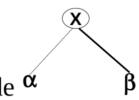
The remaining sections of the paper are organized as follows: We review the definitions and theoretical relations of the OBDD and DNNF languages; formulate the diagnosis problem and discuss the compilation of the system model into DNNF in the off-line phase; compare the complexities of on-line diagnostic algorithms for DNNF and OBDD to demonstrate the better efficiency and scalability of the DNNF-based method; describe our software tool for DNNF-based diagnosis; present results of an empirical study; and finally conclude with a short summary of contributions.

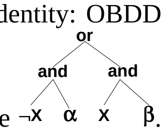
OBDD and DNNF

We briefly review in this section the definitions and theoretical relations of the OBDD and DNNF languages.

Introduced in (Bryant 1986), OBDDs are graph representations of propositional theories (or, equivalently, Boolean functions). An OBDD is a directed acyclic graph (DAG) with one root and at most two sinks. The sinks are labeled with 0 and 1, respectively; every internal node is labeled with a variable and has two children. In the example of Figure 1a, we have distinguished between the two children of a node, commonly referred to as *low* and *high*, by using a dotted line for the former and a solid line for the latter. It is required that variables appear in the same order on all root-to-sink paths. An OBDD represents a propositional theory by the following semantics: Given an instantiation I of the variables, one picks a path from the root to a sink while always choosing the low (high) child of a node if the variable labeling that node is set to 0 (1) by I; if the path ends with the 0-sink (1-sink), the theory evaluates to 0 (1) for this variable instantiation.

A propositional sentence in DNNF is also a rooted DAG, but with a different labeling scheme: Each leaf is labeled with a literal (i.e., variable or its negation) or a constant (i.e., 0 or 1), and each internal node is labeled with either "and" or "or." It is required that the DAG satisfy *decomposability*: No variable is shared between children of any and-node.[1] The semantics of DNNF is defined in the straightforward way: Each and-node (or-node) is the conjunction (disjunction) of its children. See Figure 1b for a DNNF example.

It has been observed in (Darwiche 2001) that any OBDD is also a DNNF sentence via the following identity: OBDD node 

is equivalent to DNNF node 

. This implies that any OBDD of tractable size immediately gives a DNNF sentence of tractable size. The reverse, however, is not true. There are known propositional theories that have polynomial-size representations in DNNF, but not in OBDD (Darwiche 2001).[2] As we mentioned earlier, this relative succinctness of DNNF, by itself, does not guarantee better performance in on-line diagnosis, but it will be the basis for our ability to achieve better efficiency and scalability in the off-line compilation phase, as we discuss next.

Compiling the System Model

In this section, we formulate the problem of consistency-based diagnosis, describe the encoding of system models and their compilation into DNNF using a state-of-the-art compiler (Darwiche 2004), and discuss the advantages of using DNNF over OBDD for compilation.

A *system description* is a triple $(\Delta, \mathbf{A}, \mathbf{O})$ where Δ, the system model, is a propositional sentence describing the behavior of the system to be diagnosed, and $\mathbf{A}$ and $\mathbf{O}$ are disjoint subsets of the variables of Δ, known respectively as *assumables* and *observables*. Intuitively, assumables represent the health of system components, and observables correspond to the appearance of the system that can be measured. Variables of Δ that are not in $\mathbf{A}$ or $\mathbf{O}$ will be referred to as *nonobservables*.

Given a system description $(\Delta, \mathbf{A}, \mathbf{O})$ and *observation* α, which is an instantiation of the observables $\mathbf{O}$, a *consistency-based diagnosis* is defined as an instantiation of the assumables $\mathbf{A}$ that is logically consistent with $\Delta \wedge \alpha$.

[1] When generalized to multi-valued variables, one can observe that DNNF is a strict superset of the AND/OR graphs recently studied in the context of belief and constraint networks (Dechter & Mateescu 2004b; 2004a). In fact, even *deterministic* DNNF (d-DNNF) (Darwiche & Marquis 2002), a strict subset of DNNF, is a strict superset of AND/OR graphs.

[2] For example, consider 32-bit integers X_i, $0 \le i < n$, each represented by 32 Boolean variables X_{ik}, $0 \le k < 32$. It is well-known that the OBDD encoding the proposition "not all integers X_i are distinct" has an exponential size for any variable ordering. There is, however, a quadratic-size DNNF encoding: $\bigvee_i \bigvee_{j>i} \bigwedge_k [(X_{ik} \wedge X_{jk}) \vee (\neg X_{ik} \wedge \neg X_{jk})]$. These are a class of propositional theories that are encountered in real-world applications where, for example, one wishes to verify that "no resource is allocated to two different users (McMillan 2002)."

Consider for example the small circuit shown in Figure 2. This circuit consists of an inverter X and an and-gate Y, and can be modeled by the following propositional sentence:

$$\Delta \equiv (okX \Rightarrow (A \Leftrightarrow \neg C)) \wedge (okY \Rightarrow (B \wedge C \Leftrightarrow D)). \quad (1)$$

Note that we have introduced okX and okY to represent the health of the respective components of the circuit—these two variables are therefore the assumables. The inputs A and B and output D of the circuit are the observables and the internal wire C is a nonobservable. An example observation may be $\neg A \wedge B \wedge \neg D$, for which $\neg okX \wedge okY$, $okX \wedge \neg okY$, and $\neg okX \wedge \neg okY$ are all the possible diagnoses.

With the compilation-based approach, the diagnostic task is divided into an off-line and an on-line phase. In this section we discuss the off-line phase, where one focuses on efficiently compiling the system model Δ into a compact representation in some target compilation language. In the next section we will discuss algorithms that answer on-line diagnostic queries using the compiled representation.

Compact Encoding of Multi-valued Variables

Since the behavior of a system can often be naturally described as a combination of the behaviors of its components, from here on we assume that the system model Δ is given in Conjunctive Normal Form (CNF). A CNF formula is defined as a conjunction of clauses, where each clause is a disjunction of literals; a literal is a variable or its negation. For example, the circuit in Figure 2, modeled by Equation 1, can also be specified by the CNF formula shown to its right.

In view of the fact that many systems of interest come with multi-valued variables, we have implemented an extension to the DNNF compiler of (Darwiche 2004), which allows a special type of clauses, called *e-clauses*, to be declared as part of the input CNF. Instead of regular disjunction, an e-clause $\oplus(l_1, \ldots, l_k)$ represents the exclusive-or of its literals. A variable x on a discrete multi-valued domain $\{v_1, \ldots, v_k\}$ then translates into k Boolean variables $x_1, \ldots, x_k$ constrained by a single e-clause $\oplus(x_1, \ldots, x_k)$: Each x_i encodes the proposition $x = v_i$ and the e-clause encodes that variable x must assume exactly one of its values.

We note that the use of e-clauses is in contrast to the commonly adopted method, used in (Darwiche 1998; Torta & Torasso 2004) for example, where a regular k-literal clause: $x_1 \vee \ldots \vee x_k$, plus $\frac{k(k-1)}{2}$ binary clauses: $\neg x_i \vee \neg x_{i+1}, \ldots, \neg x_i \vee \neg x_k$ for $i = 1$ to $k-1$, are required to specify the same constraint. In a search-based compiler such as that of (Darwiche 2004), allowing the compact e-clauses prevents the sheer volume of clauses from bogging down the compilation when the system comes with a large number of multi-valued variables. (For readers familiar with SAT algorithms, the compiler of (Darwiche 2004) runs a DPLL-style search on the CNF, and an e-clause is specially marked so that when any of its literals becomes 1 all others are immediately set to 0; not explicitly having the $\frac{k(k-1)}{2}$ binary clauses shortens the *watched* lists of all the literals involved.)

Compiling the System Model into DNNF

Now that the system model is available in CNF (extended to allow e-clauses), we will invoke the CNF-to-DNNF compiler of (Darwiche 2004) to obtain a compiled representation for the system model in DNNF.

Table 1: Compiling ISCAS89 circuits into OBDD & DNNF.

Circuit Name	CNF Vars	CNF Clauses	OBDD Size	OBDD Time	DNNF Size	DNNF Time
s820	312	1046	1372536	72.99	23347	0.07
s832	310	1056	1147272	76.55	21395	0.05
s838.1	512	1233	87552	0.24	12148	0.02
s953	440	1138	2629464	38.81	85218	0.26
s1196	561	1538	4330656	78.26	206830	0.44
s1238	540	1549	3181302	158.84	293457	0.94
s1488	667	2040	6188427	50.35	51883	0.19
s1494	661	2040	3974256	31.67	55655	0.18

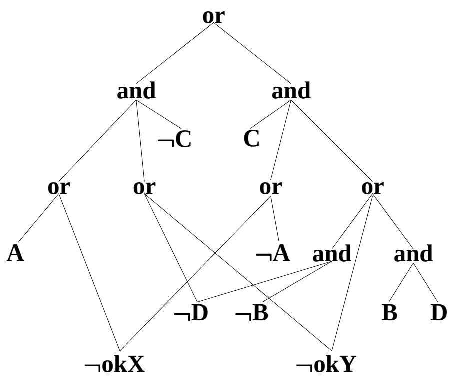
We would like to emphasize here that this compiler derives its efficiency and scalability, in large part, from exploiting the structure of the system in a fairly systematic way. Specifically, the compilation is driven by a *decomposition tree* (dtree) for the CNF formula, which is constructed in a preprocessing stage based on efficient heuristics that attempt to minimize its *width*. This results in generally efficient compilation for structured systems that admit low-width dtrees as the compilation is known to have a complexity that is polynomial in the number of variables and exponential only in the width of the dtree in the worst case.

This worst-case complexity of DNNF compilation was first shown in (Darwiche 1998). However, we note that the new DNNF compiler (Darwiche 2004) can achieve much better performance in the average case, thanks to several additional techniques that improve search efficiency, such as unit resolution, conflict-directed backtracking, clause learning, and more effective caching. In particular, systems with a high width, for which the earlier compiler (Darwiche 1998) would be hopeless as it used a strictly structure-based algorithm, are no longer necessarily difficult to compile.

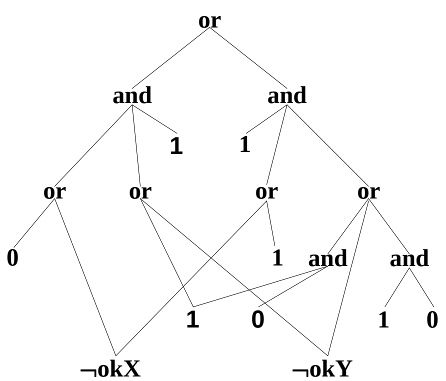
In the previous section we discussed the theoretical succinctness relation between the languages of DNNF and OBDD. On the empirical side, we would like to refer the reader to (Darwiche 2004) for results on a set of ISCAS85 and ISCAS89 circuits,[3] which were successfully compiled into DNNF in between a few seconds and a few minutes, yet some of which could not be compiled into OBDD using either the CUDD package (Somenzi Release 240) or the OBDD compilation algorithm of (Huang & Darwiche 2004).

To obtain a more concrete picture of this comparison, we experimented with a subset of these benchmark circuits, not covered in (Darwiche 2004), where compilation is possible for both DNNF and OBDD. For DNNF compilation we used the compiler of (Darwiche 2004); for OBDD compilation we used the compiler of (Huang & Darwiche 2004), with

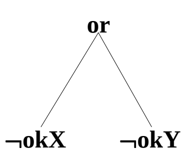
[3] These circuits are represented in CNF by introducing one variable for each internal wire and writing several clauses to model the behavior of each logic gate. The resulting CNF formula hence includes the input, output, as well as all internal wires of the circuit, but no assumables (i.e., variables modeling the health of the gates).



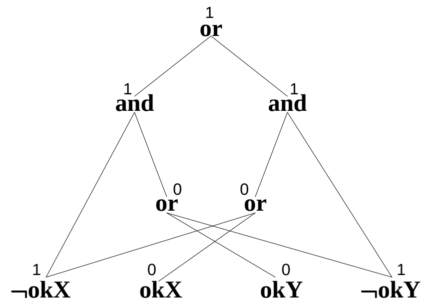
(a) Compiled system model in DNNF



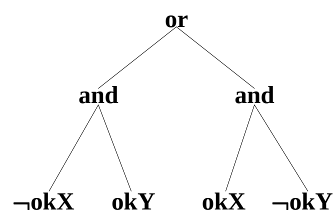
(b) Restricting the model to the observation ($\neg A \wedge B \wedge \neg D$) and existentially quantifying the nonobservable (C)



(c) Simplifying the DNNF to obtain a representation for all diagnoses



(d) Smoothing the DNNF and computing the minimum cardinality



(e) Pruning the DNNF to obtain a representation for minimum-cardinality diagnoses

Figure 3: Diagnosis using DNNF.

MINCE variable orders (Aloul, Markov, & Sakallah 2001), as it has been reported to outperform CUDD-based compilation on these circuits (Huang & Darwiche 2004). The results are shown in Table 1. The second column of the table gives the size of the CNF formula encoding the circuit in terms of the number of variables and the number of clauses; the timings are given in seconds based on a 2.4GHz CPU. For a more meaningful comparison, the size of the compilation, for both languages, has been given in terms of the number of edges in the DNNF DAG (each internal OBDD node contributes 6 edges to the DNNF DAG as we illustrated earlier). We observe that for these circuits the DNNF compilation was orders-of-magnitude faster than the OBDD compilation, and the compiled results are between one and two orders of magnitude smaller in DNNF than in OBDD.

In summary, both theoretical and empirical results suggest that the choice of DNNF over OBDD as the target language for compiling the system model will help improve the efficiency and scalability of the off-line phase of the diagnosis.

The Complexities of Diagnostic Queries

The job is but half done after successful compilation of the system model. It remains to be seen whether the diagnostic task can be carried out efficiently on this compiled knowledge base. We consider first the problem of computing the complete set of diagnoses, then that of computing the set of minimum-cardinality diagnoses only. As we shall see, although DNNF allows fewer tractable queries, its weaker representational requirement leads to lower complexities for computing and representing the diagnoses than if the OBDD representation is used.

Computing All Diagnoses

Given a system description $(\Delta, \mathbf{A}, \mathbf{O})$, a compilation Δ' for the system model Δ, and a system observation α, the complete set of diagnoses Δ_d can be characterized as follows:

$$\Delta_d = \mathsf{Project}(\mathsf{Restrict}(\Delta', \alpha), \mathbf{A}). \qquad (2)$$

That is, we restrict the compiled system model Δ' by incorporating the observation α, and then project the result on the assumables $\mathbf{A}$. We will define these two operations more formally later, but let us first illustrate them with a concrete example.

Let us consider again the diagnosis of the circuit in Figure 2 under the observation $\alpha \equiv \neg A \wedge B \wedge \neg D$. Figure 3a depicts the compiled system model in DNNF. For the Restrict operation, all one need do is visit the leaves of Figure 3a and replace A with 0, $\neg A$ with 1, B with 1, $\neg B$ with 0, D with 0, and $\neg D$ with 1. For the Project operation, again one visits the leaves of the DNNF but, this time, replaces both C and $\neg C$ with 1. The result of this process is shown in Figure 3b. The DNNF is then simplified in the straightforward way, yielding a smaller DAG representing the set of all diagnoses, shown in Figure 3c, which basically says: Gate X is abnormal, or gate Y is abnormal (or both). In general, one would obtain a more complex DNNF DAG, which characterizes the set of all diagnoses. One can also enumerate the diagnoses easily from the DNNF, but their number could be large unless filtered further as shown below.

As we have illustrated with this example, restricting a DNNF sentence, regardless of the number of variables involved in the instantiation, amounts to replacing the relevant leaves with their given values. Restrict can therefore be performed in time linear in the size of the DNNF. We note that OBDDs do just as well for this particular operation: Restricting an OBDD, also regardless of the number of variables instantiated, amounts to a single traversal of its nodes where pointers to nodes labeled with instantiated variables are updated (Bryant 1986).

Recall that the projection of a propositional theory Γ on a set of variables $\mathbf{A}$ is the result of *existentially quantifying* the complement set of variables $\overline{\mathbf{A}}$ from Γ. Existential quantification of a single variable can be defined as:

$$\exists x \Gamma \equiv \mathsf{Restrict}(\Gamma, x) \vee \mathsf{Restrict}(\Gamma, \neg x). \quad (3)$$

In a typical OBDD package, existential quantification is computed exactly according this definition. Since disjunction, as well as other binary Boolean operations, can potentially take quadratic time and produce a result of quadratic size, using the *Apply* operator (Bryant 1986), it is well known that when multiple variables are being quantified one can encounter an exponential complexity.[4]

Existential quantification (and, hence, projection) is a much easier task for DNNF as we have seen in the example. To existentially quantify a set of variables, regardless of their number, one need only traverse the DNNF once, substituting the constant 1 for all leaves (literals) that mention a quantified variable and eliminating redundant nodes that result (Darwiche 2001).[5]

The Compilation Gets Even Smaller

Because of the low complexity of projection on DNNF, after compiling the system model into DNNF Δ', one can immediately project it on the observables $\mathbf{O}$ and assumables $\mathbf{A}$ (i.e., existentially quantify out the nonobservables) and obtain a simplified representation Δ'':

$$\Delta'' = \mathsf{Project}(\Delta', \mathbf{O} \cup \mathbf{A}). \quad (4)$$

The diagnoses Δ_d can then be computed simply as:

$$\Delta_d = \mathsf{Restrict}(\Delta'', \alpha). \quad (5)$$

In the example of Figure 3a, applying this early projection would have removed the two nodes that mention variable C. Each subsequent diagnostic case would then be solved on the simpler representation. Since the compiled system model will generally be used many times to answer diagnostic queries on-line, a smaller size to start with helps further improve the efficiency of this phase of diagnosis.

The same technique is not always feasible with OBDDs. Because of the potentially exponential complexity, the projection of Equation 4 may not be practical, or may result in an OBDD much larger than the original. Under these circumstances, one would prefer to postpone the projection until the compiled system model has been restricted to a given observation (at which point the OBDD will be smaller and the projection will therefore be more likely to succeed), as prescribed by Equation 2.

Computing Minimum-Cardinality Diagnoses

In cases where the set of diagnoses is large, one may be interested in only those diagnoses, called *preferred diagnoses*, that satisfy one or more particular criteria. *Minimum cardinality* is a typical criterion used for this purpose. A diagnosis of minimum cardinality is one that identifies the smallest possible number of faulty components.

Suppose that we have already obtained the complete set of diagnoses Δ_d using either Equation 2 or Equation 5. Computing the minimum-cardinality diagnoses will then amount to an operation known as *minimization* (Darwiche 2001), which refers to a revision of Δ_d so that it encodes only a subset of its original models (i.e., satisfying variable assignments) where the number of variables assigned 0 is the smallest possible (without loss of generality, we assume that an assumable, when set to 0, represents the abnormality of the corresponding system component).

Minimization can be carried out efficiently on DNNF by traversing the DAG exactly three times (Darwiche 2001). In the first traversal, the DNNF is *smoothed* so that disjuncts of any disjunction mention the same set of variables; see Figure 3d. Smoothing takes $O(|\Delta_d| \cdot |\mathbf{A}|)$ time and results in a new DAG of size $O(|\Delta_d| \cdot |\mathbf{A}|)$.[6] The second traversal computes the mc (minimum cardinality) for each node of the DAG: The mc of a positive (negative) literal is 0 (1); the mc of an or-node (and-node) is the minimum (sum) of the mc's of its children; see Figure 3d. This computation takes

[4](Torta & Torasso 2004) contains a result that if the observables $\mathbf{O}$ and nonobservables $\mathbf{N}$ are logically dependent on the assumables $\mathbf{A}$ and the system *context*, then Equation 2 can be computed in $O((|\mathbf{O}| + |\mathbf{N}|) \cdot |\Delta'|^2)$ time for OBDDs.

[5]Since all OBDDs are in DNNF, one can do the same on an OBDD; the result, however, will not be an OBDD. A similar argument applies to the minimization operation which we discuss later.

[6]One can omit this smoothing step; the final DNNF for minimum-cardinality diagnoses should then be interpreted differently: When enumerating the models (diagnoses) of the DNNF, assume that variables with unspecified values are assigned 1. This alternative method can be slightly more efficient in practice, and was used for the experiments reported later.

Table 2: The complexity of diagnosis: A comparison of state-of-the-art algorithms.

Operation	OBDD	DNNF
$\mathsf{Restrict}(\Delta', \alpha)$	$O(\lvert\Delta'\rvert)$	$O(\lvert\Delta'\rvert)$
$\mathsf{Project}(\Gamma, \mathbf{A})$	exponential	$O(\lvert\Gamma\rvert)$
$\mathsf{Minimize}(\Delta_d)$	$O(mc \cdot \lvert\Delta_d\rvert \cdot \lvert\mathbf{A}\rvert^2)$	$O(\lvert\Delta_d\rvert \cdot \lvert\mathbf{A}\rvert)$

time linear in the size of the smoothed DNNF. The final traversal visits every or-node and cuts off all its children whose mc is greater than that of their parent. This simplification produces a DAG that encodes exactly the diagnoses of minimum cardinality, again taking time linear in the size of the smoothed DNNF. The overall complexity of minimization is therefore $O(|\Delta_d| \cdot |\mathbf{A}|)$.

Figure 3e depicts the result of applying minimization to Figure 3c. The number of diagnoses is now down to 2—the diagnosis with two abnormal gates has been wiped out.

Minimization of OBDDs is not as trivial. The most recent algorithm for this purpose appeared in (Torta & Torasso 2004); it works by constructing a series of OBDDs $Filter[k]$ so that when $Filter[k]$ is conjoined with the OBDD Δ_d encoding the complete set of diagnoses, the resulting OBDD

$$\mathsf{Apply}(\wedge, \Delta_d, Filter[k]) \quad (6)$$

will characterize exactly all diagnoses of cardinality k. The algorithm then iteratively runs Equation 6 for increasing values of k, starting from 0, until the resulting OBDD characterizes a nonempty set of diagnoses (i.e., is not the 0-sink).

According to (Torta & Torasso 2004), all the OBDD filters $Filter[k]$ have size $O(|\mathbf{A}|^2)$ regardless of k. Hence computing Equation 6 takes $O(|\Delta_d| \cdot |\mathbf{A}|^2)$ time and the overall algorithm, having to run mc (minimum cardinality) iterations, takes $O(mc \cdot |\Delta_d| \cdot |\mathbf{A}|^2)$ time, which is clearly greater than the $O(|\Delta_d| \cdot |\mathbf{A}|)$ complexity for DNNF minimization.

We close this section by summarizing in Table 2 the main complexity results that we have compared for diagnosis-related operations on OBDD and DNNF.

A DNNF-based Tool for Diagnosis

We implemented the DNNF diagnostic algorithms, linked them with the DNNF compiler of (Darwiche 2004) (extended to allow e-clauses), and produced a tool for efficient model-based diagnosis (available at http://reasoning.cs.ucla.edu/). Given a system description $(\Delta, \mathbf{A}, \mathbf{O})$ where Δ is in CNF, and an observation on variables $\mathbf{O}$, this tool computes consistency-based diagnoses of minimum cardinality as described in the previous section. In addition, it has the following two features:

- For particularly large systems where the compilation of CNF Δ may be difficult, an alternative is offered where Δ is first simplified by setting all the $\mathbf{O}$ variables to their observed values, and then compiled into DNNF. The result is again projected on the assumables $\mathbf{A}$ to produce the diagnoses as before. The drawback of this method is that compilation has to be done for each diagnostic case, even though the system model remains unchanged. Scalability can be significantly extended, however, because instantiating the observables in Δ can substantially reduce its size, making compilation more likely to succeed.
- An efficient verification component is built into the tool, which invokes an independent SAT solver (zChaff-04) to verify the correctness of the diagnoses computed: A diagnosis d is proved correct for a system model Δ and observation α if $\Delta \wedge \alpha \wedge d$ is found satisfiable by the SAT solver. The minimum-cardinality diagnoses we report in the next section have all been verified in this manner. (We have also verified that they are indeed of minimum cardinality by using the SAT solver, in a brute-force search, to prove that no diagnosis of lower cardinality exists.)

Table 3: Using DNNF vs. OBDD on a diagnosis benchmark.

Diagnosis in	Δ'		Δ_d	Δ_{mcd}	
3 Steps →	Size	Time	Size	Size	Time
OBDD-S2	375500×6	12.19	2678×6	79×6	0.051
OBDD-S3	59000×6	1.81	3229×6	77×6	0.035
DNNF	3750	0.05	85	42	0.011

An Empirical Study

Our goal in this section is to empirically compare the DNNF-based and OBDD-based approaches on a real-world diagnostic problem. Since (Torta & Torasso 2004) contains an empirical evaluation of the OBDD-based approach, we took the benchmark that was used there, and applied the tool that we described above to solve the same diagnostic tasks—computing the diagnoses as well as minimum-cardinality diagnoses. The experiments were run on a 2.4GHz processor and allocated 512MB of memory.

The said benchmark consists of a model of an industrial plant and 3000 diagnostic cases each “containing between 1 and 3 faults.” After translating the system model, originally given in XML, into CNF, we obtained a formula with 279 Boolean variables and 386 clauses, of which 36 are e-clauses (i.e., 36 of the variables in the original system model are multi-valued).

Our task was then completed in three steps. First, we compiled the CNF formula into DNNF. The compilation took only 0.05 second and the resulting DNNF had 1600 nodes and 3750 edges.[7] In the second step, and for each of the 3000 diagnostic cases (one of these turned out to be invalid and was not used), we computed the set of all diagnoses (of any cardinality) as a DNNF DAG. The average running time per case was 0.0111 second and the average DNNF had 65 nodes and 85 edges. Finally, we minimized the DNNF to produce the set of minimum-cardinality diagnoses. This took an extra 0.0002 second and resulted in a DNNF DAG with 40 nodes and 42 edges on average.

[7] Using Equation 4 would knock it down to 830 nodes and 1605 edges without noticeable increase in the running time. Our subsequent computation was based on this simplified compilation; we report the size of the full compilation here for the later comparison with its OBDD counterpart.

The same benchmark posed an initial challenge for the OBDD-based approach (Torta & Torasso 2004): Much effort was required to obtain a suitable variable ordering and even the two best orderings found led to relatively large OBDDs. See Table 3: The first two rows of data correspond to the two sets of results reported in (Torta & Torasso 2004) based on the two best OBDD variable orderings found,[8] and the last row summarizes our corresponding results for DNNF described above. The three columns of data correspond to the three diagnosis steps described above. The DAG size again reflects the number of DNNF edges (recall from the second section that an internal OBDD node contributes 6 DNNF edges).

It can be clearly seen that on this benchmark the use of DNNF has led to faster compilation and diagnosis, as well as smaller representations for the system model and the sets of diagnoses. Together with the data in Table 1 and (Darwiche 2004), these results empirically support the role of DNNF compilation in improving the efficiency and scalability of model-based diagnosis.

Conclusion

We systematically compared the use of OBDD and DNNF as target compilation languages in model-based diagnosis. In particular, we demonstrated theoretically that using the less constrained DNNF language can lead to more efficient and scalable compilation as well as diagnosis. We presented a DNNF-based diagnosis tool (available at http://reasoning.cs.ucla.edu/), and empirically compared the DNNF-based and OBDD-based approaches on a diagnosis benchmark.

Acknowledgments

We would like to thank the authors of (Torta & Torasso 2004) for generously providing their benchmark and the two unpublished timings and kindly answering our questions about the benchmark. Thanks also to the anonymous reviewers for their helpful comments. This work has been partially supported by Air Force grant FA9550-05-1-0075 and MURI grant N00014-00-1-0617.

References

Aloul, F.; Markov, I.; and Sakallah, K. 2001. Faster SAT and smaller BDDs via common function structure. In *International Conference on Computer Aided Design (ICCAD)*, 443–448.

Bryant, R. E. 1986. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers* C-35:677–691.

Darwiche, A., and Marquis, P. 2002. A knowledge compilation map. *Journal of Artificial Intelligence Research* 17:229–264.

Darwiche, A. 1998. Model-based diagnosis using structured system descriptions. *Journal of Artificial Intelligence Research* 8:165–222.

Darwiche, A. 2001. Decomposable negation normal form. *Journal of the ACM* 48(4):608–647.

Darwiche, A. 2004. New advances in compiling CNF into decomposable negation normal form. In *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI)*, 328–332.

de Kleer, J. 1990. Compiling devices and processes. Presented at the Fourth International Workshop on Qualitative Physics, Lugano, Switzerland.

Dechter, R., and Mateescu, R. 2004a. The impact of AND/OR search spaces on constraint satisfaction and counting. In *Proceedings of the Tenth International Conference on Principles and Practice of Constraint Programming (CP)*, 731–736.

Dechter, R., and Mateescu, R. 2004b. Mixtures of deterministic-probabilistic networks and their AND/OR search spaces. In *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence (UAI)*, 120–129.

Forbus, K. D., and de Kleer, J. 1993. *Building Problem Solvers*. MIT Press.

Huang, J., and Darwiche, A. 2004. Using DPLL for efficient OBDD construction. In *Proceedings of the Seventh International Conference on Theory and Applications of Satisfiability Testing (SAT)*, 127–136.

McMillan, K. L. 2002. Applying SAT methods in unbounded symbolic model checking. In *Proceedings of the 14th International Conference on Computer Aided Verification (CAV)*, 250–264.

Somenzi, F. Release 2.4.0. CUDD: CU Decision Diagram Package. http://vlsi.colorado.edu/~fabio/CUDD/cuddIntro.html.

Sztipanovits, J., and Misra, A. 1996. Diagnosis of discrete-event systems using ordered binary decision diagrams. In *Seventh International Workshop on Principles of Diagnosis (DX)*.

Torasso, P., and Torta, G. 2003. Computing minimum-cardinality diagnoses using OBDDs. In *Proceedings of the 26th Annual German Conference on Artificial Intelligence (KI)*, 224–238.

Torta, G., and Torasso, P. 2004. The role of OBDDs in controlling the complexity of model-based diagnosis. In *15th International Workshop on Principles of Diagnosis (DX)*.

[8] The timings for computing minimum-cardinality diagnoses using OBDDs were reported in (Torta & Torasso 2004) as 0.125 and 0.09 second based on a 933MHz processor; the timings for the compilation of the system model into OBDD have been provided by Gianluca Torta as 20.9 and 3.1 seconds based on a 1.4GHz processor. We have scaled these numbers accordingly to reflect the speed of our 2.4GHz processor.