Comparison of Compilation Based Diagnosis Approaches: OBDD vs DNNF

Wenfeng Zhang[1] and **Bo Pang**[1, 2], **Qi Zhao**[1], **Gan Zhou**[1], **Xiumei Guan**[1], **Wenquan Feng**[1]
[1]1 School of Electronic and Information Engineering, Beihang University, Beijing, 100191, China
email: zhangwenfeng42153@gmail.com; zhaoqi@buaa.edu.cn;
zhouganterry@hotmail.com; guanxm@buaa.edu.cn; buaafwq@buaa.edu.cn;
[2]Beijing Institute of Spacecraft System Engineering
email: to_pang_bo@yahoo.com

Abstract

To deal with computational intractability in practical diagnosis system, compilation based diagnosis has been emerging as a new direction of model based diagnosis. Although a lot of compilation languages are proposed, most demand too much storage to be practical. This paper discusses the most two extensively used compilation languages: Ordered Binary Decision Diagram (OBDD) which compiles the system model into an elegant topological structure and Decomposable Negative Normal Form (DNNF) that achieves it within ingenious decomposable property. Generally speaking, we 1) develop a uniform framework to study and discuss the two distinct diagnosis approaches; 2) compare and analyze them in both theory and practical application. The theoretical analysis and experimental result reveal that each method has its strengths and weaknesses.

1 Introduction

Automated diagnosis aims to track system behavior including state estimation, fault detection and isolation with reasoning technique. As systems become quite complex, efficient diagnosis plays an important role in ensuring safety and reliability for modern systems with high autonomy.

Origin from Artificial Intelligence (AI), Model Based Diagnosis (MBD) proposed by Reiter[1] describes systems' behavior by propositional logic, first order logic or qualitative process theory first and then computes valid values that is consist with observed variables for all the unobserved. Then, MBD has been elegantly transformed into the satisfiability (SAT) problem which is known to be NP complete[2].

Exponential runtime in size of the system is required to solve SAT problem. Many proposals have been made to boost up diagnosis efficiency during the past three decades. The development process could be divided into three classes roughly. In the first class, researchers focus on rapid methods for pure SAT problem[3], such as local consistency method, decomposition method and look ahead/back method. The second class concentrates on new methods based on special characters of diagnosis. To reduce context switching cost for different assumptions, Kleer and Williams proposes Assumption-based Truth Maintenance System[4]–[6] which is employed to generate conflicts in both General Diagnosis Engine (GDE)[7] and Conflict-Directed A^*[8]. The former gets diagnosis by hitting set but the latter does by A^* search algorithm. Considering that it is not necessary to find all the candidates, Feldman proposes StochAstic Fault diagnosis AlgoRIthm (SAFARI)[9] which trades off guarantees of minimal diagnosis for efficiency. In the third class, we trade space for time by compiling system model into a special formula.

The earliest compilation method is prime implicate[10], [11] in TMS. However, it is impractical for complex systems because there are too many prime implicates to store. Another notable compilation method is Finite State Machine(FSM) based diagnoser proposed by Sampath[12], [13]. Darwiche introduces many common knowledge compilation methods in [14] including Negation Normal Form(NNF), Decomposable Negation Normal Form(DNNF), Free Binary Decision Diagram(FBDD), Ordered Binary Decision Diagram(OBDD), Prime Implicates(PI), Prime Implicants(IP) and others in which OBDD and DNNF are employed in MBD mostly because they are much more compact than others.

OBDD is Directed Acyclic Graph(DAG) representation of Boolean functions[15]. Sztipanovits and Misra[16] propose OBDD based compilation diagnose firstly and Torta[17], [18] completes all the diagnosis schemes and algorithms. In Torta's thesis[17], system model is compiled into OBDD and diagnosis results are also in the form of OBDD. Because most operations of OBDD can be implemented in polynomial time, we could diagnose on line efficiently. DNNF proposed by Darwiche[19] endows NNF decomposability which makes variables independent relatively from each other. DNNF is not used in dynamic systems for exact diagnosis because exponential time is needed. But it is still a good compilation language for approximate diagnosis[20].

This paper makes a comprehensive comparison between OBDD and DNNF based compilation diagnosis methods. In section 2, we introduce the framework of compilation diagnosis methods and basic concepts of OBDD and DNNF. Implements of OBDD/DNNF based diagnosis are illustrated in section 3. Then section 4 analyzes the complexity of OBDD and DNNF and their functionality in diagnosis. Section 5 applies the two approaches to a thermal control system. Finally, section 6 discusses the relationship between OBDD and DNNF and presents the conclusions of this work.

2 Back Ground

In this section, a simple example is given to illustrate how reasoning based MBD works. Then we discuss the basic idea and framework of compilation based MBD.

Figure 1 shows a simple component which has three Boolean I/O variables where x_1 and x_2 are inputs and x_3 is output. We denote by variable *m* the health state of the component. When *m* is *true*, namely the component is normal, the I/O variables obey that $x_1+x_2=x_3$ where '+' means Boolean operation *and*. So we could model the component by Boolean function (1). Because the component is physically realizable, function *f* must be *true*. Reasoning based MBD obtains valid values of unobserved variables by Boolean Constraint Propagation (BCP)[2], [21] or other inference procedures on line. Then we know whether *m* could be *true* or not. However, the intractable inference needs too much resource and time, especially for complex systems.

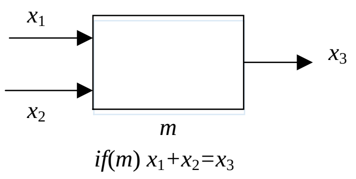


Figure 1 A Simple Component

$$f(m, x_1, x_2, x_3) = m \Rightarrow (x_1 + x_2 = x_3) \quad (1)$$

As illustrated in Figure 2, compilation based diagnosis is composed of two stages: off line and on line. In the off line stage, system model is compiled by diagnostic compiler based on the target language schema, resulting in compiled system model for on line diagnosis (step 1). When we get system observation in on line stage, diagnoser will output all diagnosis result which is usually in form of target language too (step 2). To get preferred diagnosis results, evaluator is needed (step 3). In this case, compiled system model includes both basic behavior model and priori information.

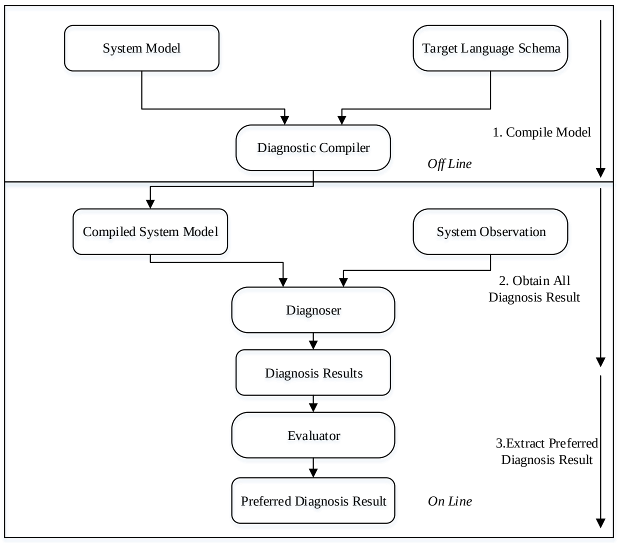


Figure 2 Framework of Compilation Based Diagnosis

Although model based diagnosis is NP-complete[22], compilation based diagnosis accomplishes most computation in off line stage where there are enough software and hardware resources and time. In turn, the size of compiled system model is much larger than the original system model. Both the size and the on line diagnosis cost is determined directly by the target compilation language. Many languages are proposed in knowledge compilation domain[14], however only OBDD and DNNF are employed mostly in MBD because they have a good balance between compiled model size and on line diagnosis cost.

2.1 OBDD

2.1.1 Definition of OBDD

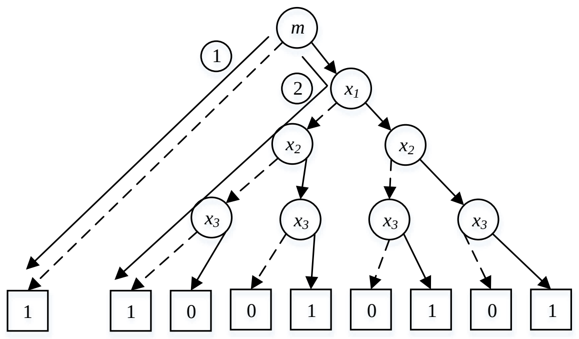


Figure 3 OBDD of function

OBDD is a canonical representation of Boolean functions as directed acyclic graph proposed by Bryant[15]. Each vertex in the DAG represents one variable and has two arcs in which one means the variable is set as *true* and the other means *false*. Any path from root vertex to leaf vertex represents a value assignment for the Boolean function. A leaf vertex is either *true* or *false* representing the value of the Boolean function under assignments of paths from root vertex to it. A path whose leaf vertex is *true* is 1-path or *true*-path, and 0-path or *false*-path in the opposite. Here, we give the formal definition of OBDD.

Definition 1, **Ordered Binary Decision Diagram (OBDD)** is a DAG with the following characters:

- Each nonterminal vertex is labelled by a variable *var*(*v*) and has arcs directed toward two children. The arc directed to *lo*(*v*) corresponds to the case *var*(*v*) is assigned *false* and is usually shown as a dashed line in figure. The other toward *hi*(*v*) corresponds to the case of *true* and is usually shown as a solid line in figure.
- Each terminal vertex is labelled *true*(1) or *false*(0).
- Given a total ordering < over the set of all variables of the function, *var*(*u*) < *var*(*v*) holds for any vertex *u* and it nonterminal child *v*.

Figure 3 illustrates an OBDD representation the system model in Figure 1. Path 1 means when *m* is assigned *false*, no matter what other three variables are assigned, function *f* results in *true*. Path 2 means when *m* is assigned *true* and x_1, x_2, x_3 are assigned *false*, function *f* results in *true*.

Table 1 is the truth table of the same Boolean function *f*. The symbol 'X' means the responding variable could be either *true* or *false*. Table 1 is a little different from common truth table. But it is quite clear that Figure 3 is a graphic representation of Table 1. Compared with truth table, OBDD could be compressed with less storage.

Table 1 Truth Table of function $f(m, x_1, x_2, x_3)$

m	0	1							
x_1	X	0				1			
x_2	X	0		1		0		1	
x_3	X	0	1	0	1	0	1	0	1
f	1	1	0	0	1	0	1	0	1

When diagnosing on line, instead of inferencing valid values of health variable *m*, OBDD based off line compilation diagnosis finds all 1-paths that contains observed values from OBDD system model. The arc types of unobserved variables, including health variables, in 1-paths stand for the valid values.

2.1.2 Operations of OBDD

The whole diagnosis process is implemented by the basic operations of OBDD including *CONSTRUCT*, *APPLY* and *RESTRICT* which are introduced and analyzed below.

CONSTRUCT Operation

CONSTRUCT operation constructs a new OBDD representing Boolean function f based on Shannon expansion (2) where x_i represents any variable.

$$f(x_1, x_2, ..., x_n) = !x_i \bullet f(x_i = 0) + x_i \bullet f(x_i = 1) \qquad (2)$$

Before calling *CONSTRUCT* operation, the order *S* must be given where *S*(*i*) is the i_{th} variable. *CONSTRUCT* operation firstly creates a node *v* labelled by variable *S*(1). Then nodes for $f(S(1)=0)$ and $f(S(1)=1)$ are created and *S*(1)'s two arcs are directed to them respectively. When all variables are assigned values, the recursion is stopped and we get the result of *f* under a special assignment.

APPLY Operation

By applying algebraic operations to other functions, The *APPLY* operation generates new Boolean functions. For Boolean operator <*op*> (such as *AND*/∧ or *OR*/∨), given Boolean function *f* and *g*, *APPLY* operation returns new function *f* <*op*> *g*. *APPLY* operation can also be implemented by "commute" Shannon expansion (3).

$$\begin{aligned} f < op > g = \; & !x \bullet (f|_{x\leftarrow 0} < op > g|_{x\leftarrow 0}) \\ & + x \bullet (f|_{x\leftarrow 1} < op > g|_{x\leftarrow 1}) \end{aligned} \qquad (3)$$

For any Boolean function *f* and *g* with the same ordered variables, Shannon expansion could be applied recursively until <*op*> is applied on two constants. Because the OBDD representation of a Boolean function is a DAG, we identifies the function by the root of the DAG and the function and the root vertex are not distinguished later.

RESTRICT Operation

To restrict variable *x* to value *k*, we can simply delete the invalid arc for vertex *v* having *var*(*v*)=*x*. Any arc into *v* is redirected either to *lo*(*v*) for *k*=0 or to *hi*(*v*) for *k*=1.

2.2 DNNF

2.2.1 Definition of DNNF

NNF is a propositional sentence constructed from literals using only conjoin and disjoin. In practice, NNF is usually represented by a DAG as introduced in the follows. DNNF is a subclass of NNF.

Definition 2, **Decomposable Negation Normal Form (DNNF)** is a NNF with the decomposable property:

For any conjunction $\alpha=\alpha_1\wedge\ldots\wedge\alpha_n$, no atom is shared between α_i and α_j ($i \neq j$), namely $\alpha_i \cap \alpha_j = \emptyset,\ i \neq j$. Although the property seems quite simple, it makes many operations of DNNF quite effective.

Based on rules (4) and (5), then Boolean function in 2.1 can be translated into (6).

$$A \Rightarrow B \ \leftrightarrow\ !A + B \qquad (4)$$

$$A = B \ \leftrightarrow\ !A \bullet !B + A \bullet B \qquad (5)$$

$$f(m, x_1, x_2, x_3) = !m + (x_1 + x_2) \bullet x_3 + !x_1 \bullet !x_2 \bullet !x_3 \qquad (6)$$

Apparently, Boolean function (6) is a DNNF. Figure 4 represents it by a DAG in which there are three kinds of nodes: *literal* node, *and*(∧) node and *or*(∨) node. Labels beside *and* node and *or* node are the proposition the node represents.

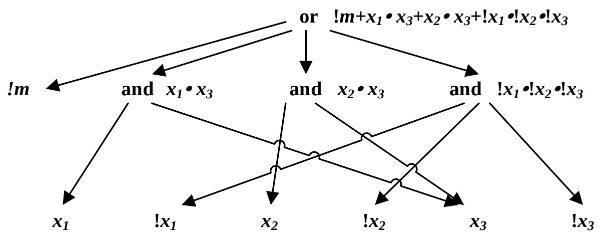


Figure 4 DAG representation of (6)

2.2.2 Operations of DNNF

DNNF's basic operations employed in diagnosis includes *CONSTRUCT*, *PROJEC*, *MCARD*, *MINIMIZE* and *MODELS*.

CONSTRUCT Operation

In [19], Darwiche calls the process *COMPILATON*. In order to unify with operations of OBDD, this paper renames *COMPILATION* as *CONSTRUCT*. It transfers normal NNF into DNNF based on a rule similar to Shannon expansion:

For two DNNFs Δ_1 and Δ_2, let X be atoms(Δ_1) ∩ atoms(Δ_2). Let Δ be the sentence $\vee_\beta((\Delta_1|\beta)\wedge(\Delta_2|\beta)\wedge\beta)$, where β is an assignment to variables in set X. Then Δ is a DNNF and is equivalent to$\Delta_1\wedge\Delta_2$.

Based on the property above, *CONSTRUCT* operation could be implemented in two phases: the first one is translating the sentence into Conjunctive Normal Form (CNF) who is the conjunction of disjunctions of literals and apparently disjunction of literals must be a DNNF; in the second phases, terms of CNF are incorporated successively.

PROJECT Operation

A *projection* of propositional sentence Δ on atom set A is another sentence Γ whose atoms are included in set A (namely, A-sentence) and for any A-sentence β, Γ ⊨ β if and only if Δ ⊨ β. For a DNNF Δ and an atom set A, *PROJECT* operation projecting propositional sentence Δ on atom set A is defined as follows.

(1) $$PROJECT(\Delta,\ \mathrm{A}) \overset{def}{=} \begin{cases} true,\ if\ \ \Delta\ is\ an\ !A-literal \\ \Delta,\ if\ \ \Delta\ is\ an\ A-literal \end{cases}$$

(2) $$PROJECT(\Delta = \wedge_i \alpha_i, \mathrm{A}) \overset{def}{=} \wedge_i\ PROJECT(\alpha_i, \mathrm{A})$$

(3) $$PROJECT(\Delta = \vee_i \alpha_i, \mathrm{A}) \overset{def}{=} \vee_i\ PROJECT(\alpha_i, \mathrm{A})$$

***MCARD* Operation**

The *cardinality* of a truth assignment ω to a propositional sentence is the number of atoms set to *false*, denoted by *CARD*(ω). *Minimum cardinality* of a propositional sentence Δ is defined as *min*{*CARD*(ω) | ω ⊨ Δ}. For a DNNFΔ, *MCARD* operation gets its minimum cardinality and is defined as follows.

(1) $$MCARD(\Delta) = \begin{cases} 0, & if\ \Delta \text{is a positive literal or } true \\ 1, & if\ \Delta \text{ is a negative literal} \\ \infty, & if\ \Delta \text{ is } false \end{cases}$$

(2) $$MCARD(\Delta = \vee_i \alpha_i) = \min_i MCARD(\alpha_i)$$

(3) $$MCARD(\Delta = \wedge_i \alpha_i) = \sum_i MCARD(\alpha_i)$$

***MINIMIZE* Operation**

A *minimization* of propositional sentence Δ is a sentence Γ which satisfies 1) for any truth assignment ω, ω ⊨ Γ if and only if ω ⊨ Δ; and 2) *CARD*(ω) = *MCARD*(Δ). *MINIMIZE* gets *minimization* of Δ and is defined as follows.

(1) $$MINIMIZE(\Delta) \overset{def}{=} \Delta, if\ \Delta\ is\ a\ literal, true\ or\ false$$

(2) $$MINIMIZE(\Delta = \vee_i \alpha_i) \overset{def}{=} \vee_{MCard(\alpha_i)=MCard(\Delta)} MINIMIZE(\alpha_i)$$

(3) $$MINIMIZE(\Delta = \wedge_i \alpha_i) \overset{def}{=} \wedge_i MINIMIZE(\alpha_i)$$

***MODELS* Operation**

MODELS Operation enumerates all truth assignment of a propositional sentence. For a smooth DNNF, *MODELS* is defined as follows.

(1) $$MODELS(\Delta) = \begin{cases} \{\{p = true\}\}, if\ \Delta\ is\ a\ positive\ literal\ p \\ \{\{p = false\}\}, if\ \Delta\ is\ a\ negaitiv eliteral\ !p \\ \{\{\}\}, \quad if\ \Delta\ is\ true \\ \{\}, \quad if\ \Delta\ is\ false \end{cases}$$

(2) $$MODELS(\Delta = \vee_i \alpha_i) = \bigcup_i MODELS(\alpha_i)$$

(3) $$MODELS(\Delta = \wedge_i \alpha_i) = \{\bigcup_i \beta_i : \beta_i \in MODELS(\alpha_i)\}$$

3 OBDD/DNNF based Diagnosis

As show in Figure 2, system model is compiled into OBDD or DNNF in the off-line stage (step 1) and then on-line diagnosis bases on the compiled model (step2 and step 3). No matter what the target language we choose, Off-line compilation is simply implemented by *CONSTRUCT* operation which is not explained here in details.

3.1 Diagnosis Result Computation

Although the basic framework of online diagnosis for the two approaches is similar, operations and their combination are quite different. Figure 5 illustrates how to get all diagnosis results based on OBDD (step 2). Compiled OBDD model (M) is composed of two parts: static model (SM) which represents the constraints between inputs and outputs under each mode and dynamic model (DM) that determines the initial available modes at next time step. *APPLY*(∧) on old modes (OM) at last time step and DM results in model under dynamic constraint (MDC) which contains available modes at the beginning of the current time step. On the other hand, model under static constraint (MSC) which contains modes consistent with observation (OBS) is obtained when *RESTRICT* SM on OBS. Available Models (AM) combines MSC and MDC by *APPLY*ing ∧ on them. Any path directed to *true* in AM is a legal value assignment and Diagnosis Result is the OBDD discarding .all non-mode variables by *PROJECT*.

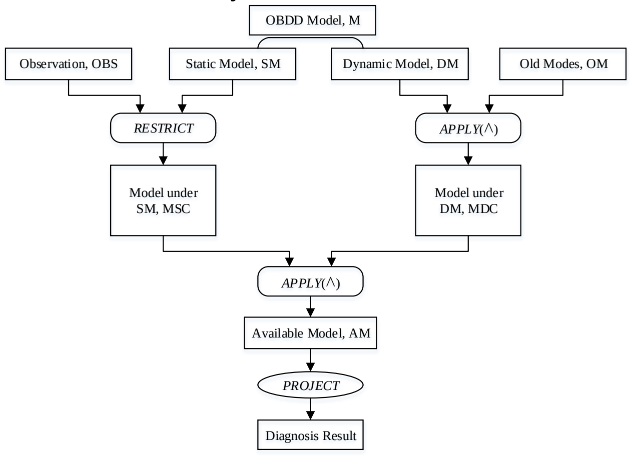


Figure 5 OBDD based on-line diagnosing without extracting perferred diagnosis result

PROJECT is not a basic operation of OBDD, however it could be implemented by *APPLY* operation. Without loss of genericity, projection of OBDD ***O*** on an atom set lacking only one variable *v* could be computed by *APPLY* ∨ on ***O***(*v*=0) and ***O***(*v*=1).

DNNF's version has the following differences: 1) DNNF restricts observation by simply assigning values for each observed variables; 2) to get MDC, DNNF uses *CONSTRUCT* operation to transform DM∧OM into DNNF; 3) Diagnosis Result is obtained by *PROJECT* operation in 2.2.1 instead of combination of other operations.

3.2 Preferred Diagnosis Extraction

In this paper, we just focus on minimal cardinality diagnosis result who has minimal faulty components.

Torta proposes a fault cardinality filter[17] based method which constructs OBDD with particular fault modes first and then *APPLY* ∧ on diagnosis result and the filter for OBDD based diagnosis. This method is able to compute preferred diagnosis results with different number of faults, however it wastes too much computation for multiple faults (*n* faults, for example) because *APPLY*(∧) on diagnosis result and filter[*i*](*i*<*n*) makes no sense.

We propose a search based preferred diagnosis extraction method which employs A* research algorithm[23] with cost function (7) where *g*(*n*) is number of variables that represent fault mode and are assigned *true*, and *h*(*n*) is zero under the assumption that all the unassigned fault mode variables will be assigned *false*.

$$\begin{aligned} &f(n) = g(n) + h(n) \\ &g(n) = \text{number of fault mode variables assigned } true \\ &h(n) = 0 \end{aligned} \qquad (7)$$

It's quite easy for DNNF to extract minimal cardinality diagnosis: 1) compute minimal cardinality by *MCARD* operation; 2) obtain minimized diagnosis result by *MINIMIZE* operation; 3) get all minimal cardinality diagnosis by apply *MODELS* operation on minimized diagnosis result. In case of strong fault mode[22], the first definition rule for *MCARD* must be replaced by (8).

$$MCARD(\Delta) = \begin{cases} 1, \ if\ \Delta \text{ is a positive literal for fault mode} \\ \infty, \ if\ \Delta \text{ is } false \\ 0, \text{others} \end{cases} \quad (8)$$

4 Comparison of OBDD and DNNF

Both OBDD and DNNF could represent knowledge in the form of DAG but with different strategies. As a kind of decision diagram, OBDD terminates checking the rest of a Boolean function once the value of the Boolean function could be determined based on partial variables' values. Taking Boolean function (9) as an example, when x_1 is false, no matter what the values of x_2 and x_3 are, f is false. So OBDD ignores the values of the two left variables.

$$f(x_1, x_2, x_3) = x_1 \bullet (x_2 + x_3) \quad (9)$$

However, DNNF considers more from the aspect of satisfiability. As explained in section 2.2, the most notable property of DNNF is that any conjunctive formulas share no atoms. So any conjunctive formulas in DNNF are independent from each other which means that 1) to ensure the DNNF to be *true*, any term of conjunctions must be *true* and 2) all truth assignments for a conjunction in DNNF is the Cartesian product of truth assignment for each term.

Next, we report our comparison of the two compilation approaches in MBD along two dimensions: complexity and functionality.

4.1 Complexity

Before analyzing the complexity, we summarize the complexity of each operation for OBDD and DNNF. Both OBDD and DNNF don't have a fixed complexity. Considering that the characters of the system model are unknown, we only consider the worst case.

Complexity of OBDD's operations

As *CONSTRUCT* operation could create $(1+2^1+\ldots+2^n) = 2^n-1$ nodes at most based on Shannon expansion, its time and space complexity is $O(2^n)$ and $\leq 2^n$ respectively. *APPLY* operation creates new nodes by comparing node in $O1$ and node in $O2$ where the biggest combination number is $|O_1|\cdot|O_2|$ where $|O|$ means the node number of OBDD O. When restricting a variable, *RESTRICT* may traverse all nodes in OBDD O. We summarize them in Table 2 [24].

Table 2 Operation Complexity of OBDD

Operation	Time Complexity	Space Complexity
CONSTRUCT	$O(2^n)$	$\leq 2^n$
APPLY	$O(\|O_1\|\cdot\|O_2\|)$	$\leq \|O_1\|\cdot\|O_2\|$
RESTRICT	$O(\|O\|)$	$\leq \|O\|$

Complexity of DNNF's operations

In the worst case, system model is translated into a CNF composed of two disjunctions of literals within the same atoms. As β in $\vee_\beta((\Delta_1|\beta)\wedge(\Delta_2|\beta)\wedge\beta)$ must traverse all the variables, time complexity of *CONSTRUCT* for DNNF is also $O(2^n)$ and the corresponding space complexity is $\leq 2^n$.

PROJECT operation traverses all nodes in DNNF and redirects all nodes not in set A to *true*. Its time and space complexity are $O(|D|)$ and $|D|$ respectively where $|D|$ means the node numbers of OBDD D.

MCARD, *MINIMIZE* and *MODELS* operations have the same time complexity with *PROJECT* operation as they are quite similar with it except the different behavior at each node.

Because *MCARD* just returns a number, we don't analyze its space complexity. *MINIMIZE* operation would delete any sub-node with cardinality bigger than other sub-nodes. The space it cost is less than $|D|$. The number of assignments making the DNNF to be *true* is 2^n mostly and space complexity for MODELS operation is $\leq 2^n$. The results are summarized as Table 3.

Table 3 Operation Complexity of DNNF

Operation	Time Complexity	Space Complexity
CONSTRUCT	$O(2^n)$	$\leq 2^n$
PROJECT	$O(\|D\|)$	$\leq\|D\|$
MCARD	$O(\|D\|)$	-
MINIMIZE	$O(\|D\|)$	$\leq\|D\|$
MODELS	$O(\|D\|)$	$\leq 2^n$

4.1.1 Model Compilation

As explained in section 3, both OBDD and DNNF compile system model by *CONSTRUCT* operation. As shown in Table 2 and Table 3, they have the same time and space complexity in the worst case. However, Darwiche[19] has proven that 1) any OBDD could be translated into an equivalent DNNF in linear time and 2) there are Boolean functions which could be represented in DNNF with polynomial time and space but admit only exponential representation using OBDD.

4.1.2 Diagnosis Results Computation

Static Diagnosis

Static diagnosis includes *RESTRICT* and *PROJECT* operations. No matter how many variables to restrict, it could be accomplished by traversing all variables once, with $O(|O|)$ time complexity and $\leq|O|$ space complexity. To project an OBDD on a variable set A, we must discard all variables not included in A. Discarding variable v from OBDD O consists of two steps: 1) restricting O on v=0 and v=1 respectively with $O(|O|)$ time complexity and $\leq|O|$ space complexity; 2) applying $\vee$ on the two restricted OBDD in the last step. Instead of complexity shown in Table 2, applying $\vee$ on $O(v=0)$ and $O(v=1)$ compares $|O|$ times mostly and the new OBDD is less than $|O|$. As a result, the time complexity is $O(2\times|O|)+(n\text{-}k)\times O(|O|^2) = O((n\text{-}k)\times|O|)$ where k is the number of mode variables and $(n\text{-}k)$ is the number of variables to be discarded. Obviously, static diagnosis result is smaller than $|O|$.

To restrict a DNNF, all need to do is traversing all DNNF node and redirecting observed nodes to *true* or *false* node. So the time complexity is $O(|D|)$ and restricted DNNF is smaller than $|D|$. From Table 3, we could conclude that time complexity of static diagnosis of DNNF is $O(|D|)+ O(|D|) = O(2\times|D|)$ and $\leq|D|$ for space complexity.

Dynamic Diagnosis

Compared with static diagnosis, dynamic diagnosis additionally employs two more steps based on *APPLY* or *CONSTRUCT* operation: 1) applying $\wedge$ on DM and OM to get MDC or translating DM $\wedge$OM into DNNF; 2) applying $\wedge$ on MSC and MDC to get available model or translating MSC $\wedge$MDC into DNNF.

Based on the complexity analysis for static diagnosis, time complexity of dynamic diagnosis of OBDD is $O((n\text{-}k)\times|O|)+O(2\times|O|^2) = O(2\times|O|^2)$ and space complexity is $\leq|O|^2$. For DNNF, time complexity is $O(2\times|D|) + 2\times O(2^n) = O(2^{n+1})$ and space complexity is $\leq 2^n$.

4.1.3 Preferred Diagnosis Results Extraction

As introduced in section 4.3, preferred diagnosis result extraction method of OBDD is based on A^* research algorithm. In order to extract all minimal cardinality diagnosis candidates, A^* may traverse all OBDD nodes in the worst case. Time complexity is $O(|O|)$. Space complexity is node analyzed here because it is all the minimal cardinality diagnosis candidates. In the case of DNNF, Table 3 gives $O(|D|)$ time complexity directly.

As a summary, we conclude the analysis results in Table 4 and Table 5.

Table 4 Time Complexity of OBDD and DNNF

Language		OBDD	DNNF
Model Compilation		$O(2^n)$	$O(2^n)$
Compute Diagnosis Result	Static Diagnosis	$O((n\text{-}k)\times|O|)$	$O(2\times|D|)$
	Dynamic Diagnosis	$O(2\times|O|^2)$	$O(2^{n+1})$
Preferred Diagnosis Result Extraction		$O(|O|)$	$O(|D|)$

Table 5 Space Complexity of OBDD and DNNF

Language		OBDD	DNNF
Model Compilation		$\leq 2^n$	$\leq 2^n$
Compute Diagnosis Result	Static Diagnosis	$\leq|O|$	$\leq|D|$
	Dynamic Diagnosis	$\leq|O|^2$	$\leq 2^n$
Preferred Diagnosis Result Extraction		-	-

4.2 Functionality

We analyze the functionality in four dimensions: model type, variable type, system character and property of observation.

4.2.1 Weak and Strong Model

A system model could be a weak one[22] which describes normal behavior only or a strong model that also describes abnormal behaviors. In weak model, *false* assignment for a mode variable means one component is abnormal. In strong model, a component gets fault if the mode variable is assigned a fault mode.

Although, strong model could not be compiled by OBDD or DNNF directly, we can encode strong model with Boolean variables. After that, the system could be compiled into both OBDD and DNNF at the expense of modifying some operations slightly as shown in section 3.2. It should be reminded that the encoding schedule has a strong impact the number Boolean variables to compile which also determines the size of the compiled model in some degree.

4.2.2 Boolean and Multi-value Variable

It's quite similar to the problem discussed in the above except that we may not modify operations if there is no multi-value mode variable.

4.2.3 Static and Dynamic System

In theory, they both have the ability to compile static and dynamic system while the efficiency is different especially for dynamic system.

From Table 4, we know that linear time and space complexity are ensured for both OBDD and DNNF. By contrast, DNNF is better because 1) DNNF may have more compact representation (see section 3.1.1); 2) $(n\text{-}k)>1$.

For dynamic system, however, OBDD could diagnose in polynomial time but DNNF costs exponential time. So DNNF is not practical in dynamic diagnosis.

4.2.4 Uncertain Observation

Sometimes, we may get uncertain observation because of noise. In these cases, observation could not be loaded by *RESTRICT* or variable assignments.

Just like dynamic diagnosis, uncertain observation could be regarded as a special "mode". When diagnosing with OBDD, we could represent the uncertain observation in form of OBDD firstly denoted by OBDD_{OBS} and then apply $\wedge$ on static model and OBDD_{OBS}. Although *APPLY* operation of OBDD admits a polynomial time complexity, compiling observation may cost too much time when the uncertainty is high because *CONSTRUCT* admits exponential complexity. DNNF based method does not need to represent observation in DNNF which is more convenient than OBDD, however translating OBS *and*($\wedge$) static DNNF model makes up the saved time.

In summary, both of them are not capable to diagnose with highly uncertain observation.

5 Case Study: Thermal Control System

This experiment employs a thermal control system in satellite to compare their performance in static diagnosis applied most extensively. The system is composed of three kinds of components: switch, heater and ammeter. Shown as Figure 6, switch S is employed to control the current of all heaters. Each heater h_i is assigned with a switch s_i to open or close it individually. At the return circuit, ammeter A monitors the current.

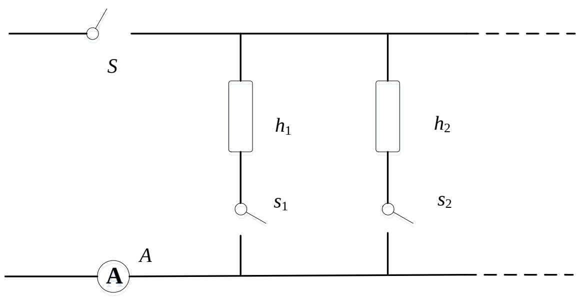


Figure 6 Thermal Control System

To diagnose the continuous system, each component is discretized as follows. A switch has three modes: normal, stuck on and stuck off. When a key is normal, its output is the same with input if commanded on and is *false* if off. It never switches off if stuck on and vice versa. (Figure 7)

To be convenient, we model heater h_i combined with its corresponding switch s_i where both h_i and s_i being faulty is not considered. The combined component includes five modes: normal, heater shorted, heater broken, switch stuck on and switch stuck off. If it is normal, the heater will release energy and output normal current when commanded on. And when command off, there is no released energy and current. If the heater gets shorted and commanded on, it outputs over limited current although still releases heat. If broken, the heater never outputs current or heat. The behaviors of mode switch stuck on and switch stuck off are quite similar with mode heater shorted and heater broken except than switch stuck on will not result in over limited current. (Figure 8)

Ammeter's behavior is quite simpe. Normally, it monitors the current in the return circuit. If all input currents are none, ammeter outputs none. If one of the input current is over, it outputs over. In other cases, normal is monitored. When faulty, it always shows the current is over limited. (Figure 9)

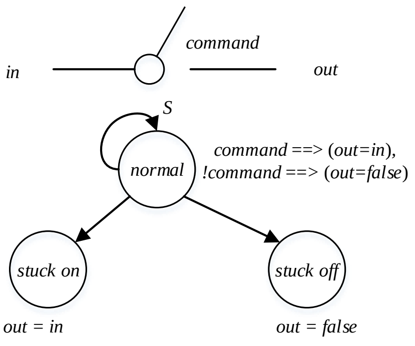
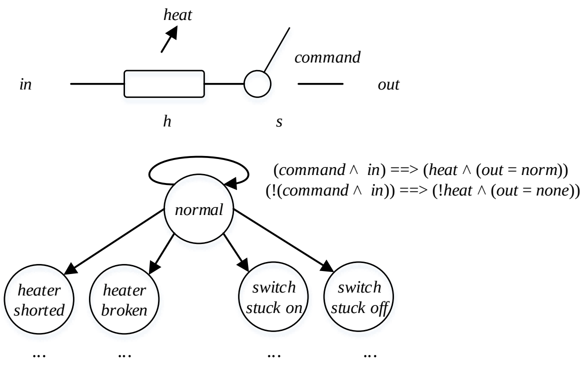
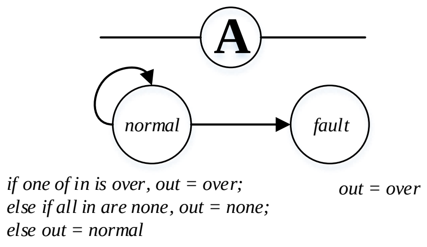


Figure 7 Model of Switch



heat
command
in
out
h
s
normal
(command ∧ in) ==> (heat ∧ (out = norm))
(!(command ∧ in)) ==> (!heat ∧ (out = none))
heater shorted
heater broken
switch stuck on
switch stuck off

Figure 8 Model of Heater



A
normal
fault
if one of in is over, out = over;
else if all in are none, out = none;
else out = normal
out = over

Figure 9 Model of Ammeter

We first apply OBDD and DNNF respectively on each component individually and then on the whole thermal control system where there is only one heater.

Table 6 records node number (space complexity) for different systems where atom number denotes the number of bool variables used to encode each system. We can see that 1) node number trends to increase with more atoms, but it is not ensured because simple system behavior demands fewer nodes to describe; 2) DNNF always has more nodes than OBDD. In fact, unreduced OBDD has much more nodes (2^n) than DNNF while it's quite easy to delete redundant nodes and arcs for OBDD. In contrast, there is no effective mechanism to simply DNNF. It is not conflict with results in section 5.1.1 which refer to worst and some special cases.

Table 6 DAG Node Number for Different Systems

system	atom number	compile method	
		OBDD	DNNF
switch	6	3	28
ammeter	8	10	27
heater	11	5	63
thermal control system	32	88	135

Figure 10 shows the time cost of thermal control system for off-line and on-line stage. It is obvious that DNNF is much quicker than OBDD not only in off-line compilation but also in on-line diagnosis. Although OBDD results in fewer nodes, it creates much more temporary nodes which deplete compilation time. When diagnosing on line, DNNF saves much time because of fast *PROJECT* operation as illustrated in Figure 11.

In summary, DNNF is much better than OBDD in compilation and diagnosis time at the expense of a little more space.

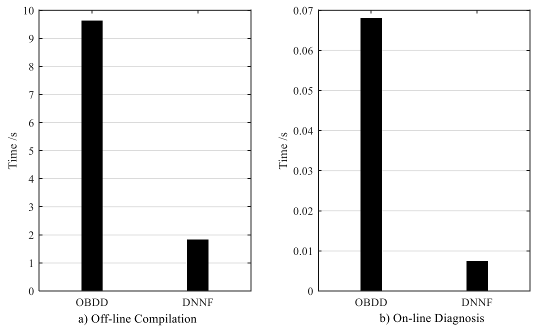


Figure 10 Time Cost of Thermal Control System

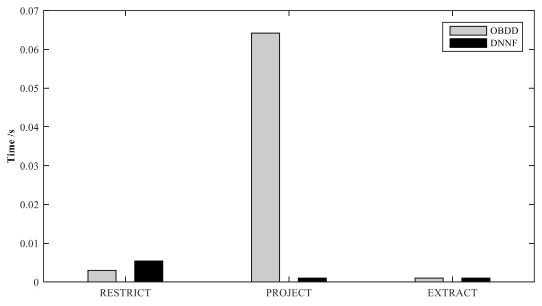


Figure 11 Time Cost for On-line Diagnosis

6 Discussion and Conclusion

In this paper, we study and compare two compilation approaches: OBDD and DNNF.

They both adopt "divide and conquer" strategy. OBDD expands Boolean functions by Shannon expansion according to an ordered variable set successively while DNNF expands it by enumerating all instances of shared atoms. The biggest difference is that Shannon expansion is applied on both conjunction and disjunction however DNNF only decomposes conjunctive sentences with shared atoms. Non-terminal node and terminal node in OBDD correspond to ∨-node and literal node in DNNF respectively while there is no relevant node for ∧-node in OBDD.

In other words, OBDD divides the original problem into "additive" sub-problems (∨-node) recursively but DNNF divides it into both "additive" (∨-node) and "multiplicative" (∧-node). To conquer global solution from additive solutions, we just need to put them together nevertheless the multiplicative must be transferred into a number of additive ones by Cartesian product firstly. Compared with the additive, the multiplicative could generate more global solutions with less local solutions namely multiplicative sub-problems are more compact. That's why a Boolean function may have polynomial DNNF representation while only admit exponential OBDD representation.

To represent sentence $(l_1 \vee l_2 \vee \ldots \vee l_m) \wedge (l_{m+1} \vee l_{m+2} \vee \ldots \vee l_n)$ where $\alpha_1=(l_1 \vee l_2 \vee \ldots \vee l_m)$ and $\alpha_2=(l_{m+1} \vee l_{m+2} \vee \ldots \vee l_n)$ share no atoms, OBDD needs 2^n-1 nodes in the worst case while it could be done by DNNF with (n+3) nodes (n literal nodes, two ∨-nodes and one ∧-node). It is obvious that OBDD could be transferred into DNNF in linear time [19] while transferring DNNF into OBDD many cost exponential time because of literal disjunction.

On the other hand, although DNNF is more compact due to multiplicative sub-problems, it suffers exponential time complexity to combine two DNNFs in turn because the mixture of additive and multiplicative division methods makes it very hard to utilize the compiled DNNF.

In general, combining the analysis and experimental results, we can conclude that:

1) Both OBDD and DNNF are suitable for static diagnosis but DNNF is better because it may have much more compact representation of some systems and its on-line diagnosis is fairly faster;

2) OBDD is capable to diagnose dynamic system while DNNF isn't due to the exponential complexity of *CONSTRUCT* operation.

3) Neither OBDD nor DNNF plays well when the observations are highly uncertain as a) compiling uncertain observation into OBDD and b) transferring model *and*(∧) uncertain observation into DNNF admits exponential complexity.

References

[1] R. Reiter, "A theory of diagnosis from first principles," *Artif. Intell.*, vol. 32, no. 1, pp. 57–95, 1987.

[2] R. Dechter, *Constraint Processing*, vol. 33, no. 2. 2003.

[3] A. Metodi, R. Stern, M. Kalech, and M. Codish, "A novel SAT-based approach to model based diagnosis," *J. Artif. Intell. Res.*, vol. 51, pp. 377–411, 2014.

[4] K. D. Forbus and J. de Kleer, *Building Problem Solvers (Artificial Intelligence)*, no. October. 1993.

[5] J. de Kleer, "An assumption-based TMS," *Artif. Intell.*, vol. 28, no. 2, pp. 127–162, 1986.

[6] J. Doyle, "A Truth Maintenance System," *Readings Nonmonotonic Reason.*, pp. 259–280, 1987.

[7] J. de Kleer and B. C. Williams, "Diagnosing multiple faults," *Artif. Intell.*, vol. 32, no. 1, pp. 97–130, 1987.

[8] B. C. Williams and R. J. Ragno, "Conflict-directed A* and Its Role in Model-based Embedded Systems," *J. Discret. Appl. Math*, no. December 2001, pp. 1562–1595, 2007.

[9] A. Feldman, G. Provan, and A. Van Gemund, "Approximate model-based diagnosis using greedy stochastic search," *J. Artif. Intell. Res.*, vol. 38, pp. 371–413, 2010.

[10] P. Marquis, "Knowledge compilation using theory prime implicates," *Proc. Fourteenth Int. Jt. Conf. Artif. Intell.*, pp. 837–843, 1995.

[11] J. De Kleer, "Compiling Devices: Locality in a TMS," in *Recent advances in qualitative physics*, 1992.

[12] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. C. Teneketzis, "Failure diagnosis using discrete-event models," *IEEE Trans. Control Syst. Technol.*, vol. 4, no. 2, pp. 105–124, 1996.

[13] M. Sampath, K. Sinnamohideen, S. Lafortune, and D. Teneketzis, "Diagnosability of Discrete-Event Systems," *IEEE Trans. Automat. Contr.*, vol. 40, no. 9, pp. 1555–1575, 1995.

[14] A. Darwiche and P. Marquis, "A perspective on knowledge compilation," *IJCAI Int. Jt. Conf. Artif. Intell.*, vol. 17, no. 1, pp. 175–182, 2001.

[15] R. E. Bryant, "Symbolic Boolean manipulation with ordered binary-decision diagrams," *ACM Comput. Surv.*, vol. 24, no. 3, pp. 293–318, 1992.

[16] J. Sztipanovits and A. Misra, "Diagnosis of discrete event systems using ordered binary decision diagrams," *Seventh Int. Work. Princ. Diagnosis*, 1996.

[17] G. Torta, "Compact Representation of Diagnoses for Improving Efficiency in Model Based Diagnosis," 2006.

[18] P. Torasso and G. Torta, "Compact diagnoses representation in diagnostic problem solving," *Comput. Intell.*, vol. 21, no. 1, 2005.

[19] A. Darwiche, "Decomposable Negation Normal Form," *J. ACM*, vol. 48, no. 4, pp. 608–647, 2001.

[20] P. Elliott and B. C. Williams, "DNNF-based Belief State Estimation," *Proc. Natl. Conf. Artif. Intell.*, pp. 36–41, 2006.

[21] A. H. Biere, M. Heule, H. van Maaren, and T. Walsh, *Handbook of Satisfiability*. 2009.

[22] A. Bos, "Choosing the Right Model in Model Based Diagnosis," 1998.

[23] S. Russell and P. Norvig, *Artificial Intelligence A Modern Approach*. 2013.

[24] P. Torasso and G. Torta, "Model-based diagnosis through OBDD compilation: A complexity analysis," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 4155 LNAI, pp. 287–305, 2006.