Model-Based Diagnosis with Sentential Decision Diagrams

Gianluca Torta and Francesco Schembri
1Dipartimento di Informatica, Università di Torino
e-mail: gianluca.torta@unito.it, francesco.schembr@edu.unito.it

Abstract

Knowledge-compilation has been advocated in the literature as a way to obtain theoretical guarantees and practically speed-up Model-Based Diagnosis (MBD). In this paper we consider a recent canonical form introduced by Adnan Darwiche and its team, namely Sentential Decision Diagrams (SDDs), as the target for model compilation within the MBD task. SDDs have been proved to be a subset of DNNFs (Decomposable Negation Normal Forms) and a superset of OBDDs (Ordered Binary Decision Diagrams), two target compilation languages already exploited in MBD. We discuss model compilation and some diagnostic applications of SDDs and compare them with their OBDD and DNNF counterparts.

1 Introduction

It is well known that the MBD task, even in its most basic form defined by Reiter [1], suffers from two potential combinatorial problems: computing a diagnosis is NP-hard, and the number of diagnoses can be exponential in the system size.

One way to mitigate these problems consists in the compilation of the system model into a target language that supports efficient operations to obtain a compact encoding of all the diagnoses. Starting from such an encoding, it should also be possible to perform post-processing operations, the most obvious of which are the computation and enumeration of preferred diagnoses according to common preference criteria (minimal cardinality, minimality w.r.t. set inclusion).

Ordered Binary Decision Diagrams (OBDDs) [2] are a target compilation language that has been shown to be effective in MBD. In particular, OBDDs have been used for the MBD of both static systems [3] and discrete-event systems (DES) [4; 5; 6; 7], and for diagnosability analysis of DES [8]. A nice feature of OBDDs is that, once the OBDD representing all the diagnoses has been computed, it can be post-processed with operations at the symbolic level (i.e., operations that transform the OBDD into another OBDD), which is essential in many tasks including MBD of dynamic systems. On the other hand, the OBDD language is quite restrictive, which can lead to an explosion of the size of the compiled model, especially when dealing with large and complex systems.

An alternative target language, Decomposable Negation Normal Form (DNNF) [9] has been shown to be much more compact than OBDDs for some large benchmark circuits [10]. Moreover, DNNFs support efficient operations for computing and extracting minimum-cardinality (mc) diagnoses of static systems, and have been shown to significantly outperform OBDDs for such a task [11]. Unfortunately, DNNFs lack the symbolic level post-processing operations that allow, e.g., to compute the diagnoses of dynamic systems.

More recently, Darwiche has introduced Sentential Decision Diagrams (SDDs) [12], a target language that maintains most of the benefits of OBDDs while being significantly more compact on several benchmark models [13]. In the present work, we aim to study the application of SDDs to several MBD tasks. In particular, we first consider model compilation, which requires a careful ordering of the variables and clauses of the system model; then, we study the computation of the mc-diagnoses of static systems; and, finally, the application of SDDs to further MBD tasks including the computation of minimal diagnoses w.r.t. set inclusion and the diagnosis of dynamic systems. As we detail the use of SDDs, we compare them to OBDDs and DNNFs by pointing out their respective theoretical properties. Moreover, for model compilation and computation of static MC diagnoses, we present and discuss experimental results obtained on some combinatorial circuit models from the ISCAS85 benchmark.

The paper is structured as follows. We first cover background information in section 2. Then, we address compilation (section 4), computation of static mc-diagnoses (section 5), and of minimal diagnoses (section 6). After covering other MBD tasks in section 7, we draw some conclusions and point to further research directions in section 8.

2 Background on Knowledge Compilation

2.1 Ordered Binary Decision Diagrams

Ordered Binary Decision Diagrams (OBDDs) are a target language for the compilation of propositional KBs, introduced in [2], which, among others, has already been applied successfully to the MBD task, e.g., [3; 4]. As customary in the literature, OBDDs are considered as an encoding of a Boolean function \(f()\), where variables and the function itself take values in \(\{0, 1\}\); we shall stick to this notation, with the understanding that a Boolean function is equivalent to a propositional theory, and that values \(\{0, 1\}\) correspond to logical values \(\{\bot, \top\}\).

Given a variables order \(x_1, \ldots, x_n\), a Boolean function \(K(x_1, \ldots, x_n)\) is compiled to a rooted DAG whose nodes
include two terminal nodes labeled 0 and 1 and non-terminal nodes each labeled with one of the \(x_i\) variables. Every internal node \(x_i\) has exactly two successors low and high (if \(x_i\) is successor of \(x_i\) in the DAG then \(x_i\) must precede \(x_j\) in the ordering). We write:

\[
ite(x, \alpha, \beta)
\]

for an OBDD with root \(x\), \(\text{high} = \alpha\), and \(\text{low} = \beta\). Every path \(P\) from the root to node 1 can be viewed as an assignment to the variables involved in \(P\) \((x_i = 1\) if \(\text{high}(x_i) \in P\) and \(x_i = 0\) if \(\text{low}(x_i) \in P\)) which guarantees that \(K\) has value 1. The size of an OBDD is defined as the number of its nodes. It is known that the OBDD of minimal size is unique for a given Boolean function \(f\) and a fixed variable ordering (canonicity). Variable ordering choice is a prominent issue when encoding a Boolean function as an OBDD, and since finding the optimal ordering is NP-hard, heuristic methods are usually adopted [14; 15].

Figure 1 shows the OBDD corresponding to \(K = (A \land B) \lor (C \land D)\) given the ordering \(A, B, C, D\). OBDDs have several properties, that are desirable in many tasks including diagnosis [2; 16]:

- An apply\((op, B_1, B_2)\) operation can compute the OBDD \(B\) resulting from applying the logical operator \(op\) to the Boolean functions encoded by \(B_1, B_2\) in time and space proportional to \(|B_1| \cdot |B_2|\).
- Canonicity ensures that checking whether \(B_1 \equiv B_2\) (i.e. if \(B_1\) and \(B_2\) encode the same function) can be done in constant time; even more importantly, the search for a compact OBDD for encoding a given function \(K\) can be reduced to a (heuristic) search over the variable orders: indeed, for each given variable order there’s only one OBDD encoding \(K\).
- Conditioning OBDD \(B\) on the value of a variable \(v\) (called restrict in OBDDs terminology), namely substituting \(v\) with a constant value 0 or 1, takes linear time w.r.t. \(|B|\).
- Model enumeration of the Boolean function \(K\) represented by \(B\) (i.e., enumeration of the assignments to the \(x_i\) variables that yield \(K = 1\)) can be done in output-polynomial time.

2.2 Decomposable Negation Normal Form

Decomposable Negation Normal Form (DNNF) is another target language, introduced in [9]. Also DNNF have already been applied successfully to the MBD task [11]. Essentially, a DNNF encodes a KB \(K(x_1, \ldots, x_n)\) as an and/or graph where:

- the leaves are literals of the variables in \(K\) or the constants \(\top, \bot\)
- each internal and node \(n\) satisfies decomposability: no variable is shared between the subgraphs rooted at the children of \(n\).

DNNFs do not prescribe a variable order, and do not have a canonical form for representing a given KB.

Figure 2 shows a DNNF corresponding to \(K = (A \land B) \lor (C \land D)\). While less restrictive than OBDDs (and, thus, potentially much more compact), DNNFs still have several properties that are desirable for diagnosis [16]:

- Conditioning DNNF \(N\) on the value of a variable takes linear time w.r.t. \(|N|\).
- Projection of DNNF \(N\) on a subset of its variables takes linear time w.r.t. \(|N|\).
- Minimal model (w.r.t. the number of true variables) enumeration of the KB represented by \(N\) can be done in output-polynomial time.

An interesting relation between OBDD and DNNF is that OBDDs are a subset of DNNFs [16]. Therefore, if convenient, an OBDD can always be considered as a DNNF at some point of the solution of the problem at hand, and processed with the DNNF operations from then on.

2.3 Sentential Decision Diagrams

The last target compilation language considered in the present work are Sentential Decision Diagrams (SDDs), introduced in [12]. As we shall see, SDDs are a generalization of OBDDs and a restriction of DNNFs.

In order to briefly describe SDDs (all the details are in [12]), we need to introduce two main concepts: \(X\)-partitions and \(v\)trees. Let us consider an \((X, Y)\) decomposition of a KB \(K(X, Y)\) with \(X \cap Y = \emptyset\), i.e., a rewriting of \(K\) as:

\[
K(X, Y) = (p_1(X) \land s_1(Y)) \lor \ldots \lor (p_n(X) \land s_n(Y))
\]

where the \(p_i\)s are called primes, and the \(s_i\)s are called subs. An \((X, Y)\) decomposition of \(K(X, Y)\) is called \(X\)-partition if:

- it is strongly deterministic: \(p_i \land p_j = \bot\) for \(i \neq j\)
- the \(p_i\)s induce a partition of the assignments to the \(X\) variables.

The \(X\)-partition is further said to be compressed if \(s_i \neq s_j\) for \(i \neq j\).

A \(v\)tree for variables \(X\) is a full binary tree whose leaves are in 1:1 correspondence with the variables in \(X\) (Figure
3. A vtree induces an order on the $X$ variables through a depth-first search (indexes in Figure 3).

A vtree plays for an SDD the role that a variable or an OBDD. Given a vtree $v$ for variables $A-D$, there is a unique compressed SDD $SD,COMPS$ (canonicity), which ensures that checking whether $\Delta$ is consistent with $\neg okC : C \in \Delta$ takes linear time w.r.t. $|\Delta|$ on the value of a variable takes exponential time [17].

While potentially much more compact, SDDs share with OBDDs some interesting properties:

- an apply($op$, $S_1$, $S_2$) operation can compute the SDD $S$ resulting from applying the logical operator $op$ to the KBs encoded by $S_1$, $S_2$ in time and space proportional to $|S_1| \cdot |S_2|$.
- given a vtree, there is a unique compressed SDD (canonicity), which ensures that checking whether $S_1 \equiv S_2$ can be done in constant time, and that the search for a compact SDD for encoding a given KB $K$ can be reduced to a (heuristic) search over the vtrees. However, contrary to OBDDs, it is necessary to choose between the polynomiality of apply and the canonicity: given two compressed SDDs $S_1$, $S_2$, computing a compressed result of an apply on $S_1$, $S_2$ may take exponential time [17].

- conditioning SDD $S$ on the value of a variable takes linear time w.r.t. $|S|$.
- model enumeration of the KB represented by $S$ can be done in output-polynomial time.

3 Background on Model-Based Diagnosis

3.1 Static MBD

Following the basic formulation of MBD in the seminal work by Reiter [11], we consider a system as a pair $(SD, COMPS)$ where $SD$ is the System Description (for the purposes of this paper, a theory in propositional logic) and $COMPS$ is a set of components. For each component $C \in COMPS$, we assume that there is an atom $okC$ that is used in $SD$ to describe the correct behavior of $C$.[1]. Given observations $Obs$, a diagnosis $\Delta$ for $(SD, COMPS)$ is a set of components $C_1, \ldots, C_k$ s.t.:

$$SD \cup Obs \cup \{\neg okC : C \in \Delta\} \not\models \bot$$

We consider two preference criteria for ranking diagnoses: minimum cardinality (diagnoses that have a minimum number of components) and minimality (diagnoses that are minimal w.r.t. set inclusion).

In the present paper we focus on weak models, i.e., models that describe only the nominal behavior of components. Extension of the discussion to strong models should be straightforward.

3.2 Dynamic MBD

There are many definitions of diagnosis for dynamic systems. Here, we follow the approach described in [18] for the diagnosis of DESs (Discrete Event Systems) with OBDDs. The system model is a tuple $(\Pi, T, D, C, M_\Sigma, M_\tau)$ where:

- $\Pi$ is the set of state variables $y_i$ ²
- $T$ is the set of transition variables $\tau_{t,y}$ for state variables $y_i$
- $D$ is the set of dependent variables
- $C$ is the set of command variables
- $M_\Sigma$ is a propositional formula over $\Pi_t$ and $D_t$; it specifies the admissible assignments to the dependent variables given the current state
- $M_\tau$ is a propositional formula over $\Pi_t$, $D_t$, $C_t$, $T_t$ and $\Pi_{t+1}$; it specifies the admissible assignment to the transition and next state variables given the current state, dependent variables and commands

Intuitively, formula $M_\Sigma$ is analogous to the $SD$ of static diagnosis, and describes the instantaneous effects of the system status. While, formula $M_\tau$ describes the (discrete) state transitions of the system.

Let $O$ be the subset of the dependent variables $D$ whose assignment corresponds to a measurement from the process. A diagnosis, is then a trajectory of states $s_0, s_1, \ldots, s_m$ such that each $s_i$ is consistent with $M_\Sigma$ and $O_i$, and $s_i \cup \tau_{t+1}$ is consistent with $M_\tau$ and $D$. A preference among diagnoses can be defined by assigning ranks (non-negative integer costs) to the transition variables $\tau_{t,y}$ and assigning to a trajectory a cost that is the sum of all the costs of the transition variables that appear in it.

[1] When convenient and clear from the context, we shall refer to $okC$ as the component $C$.

[2] State and other variables are subscripted with the discrete time point they refer to, e.g., $y_i \in \Pi_t$. 
4 Static Model Compilation

In this section, we first describe the model compilation into SDD for the static MBD task. Then, we briefly describe the compilation into OBDD and DNNF, and show some experimental results.

4.1 Compilation into SDDs

Let us assume that the System Description SD is a propositional theory in CNF (Conjunctive Normal Form). Since SDDs support an apply operation, it is possible to build the SDD S_{SD} for SD bottom-up, adding one clause at a time through an apply of the ∨ operator.

As pointed out in section 2.3, however, we have to choose whether we want to compute a compressed SDD, giving up the guarantee of polynomiality of apply, or we prefer to compute a non-compressed SDD. Extensive experiments of compilations reported in [17] give empirical evidence that it is significantly more efficient to compile a KB into a compressed SDD, and this approach is also hardcoded into the software package we have used for our experiments [19] (see below).

One of the main challenges for an effective compilation of SD is the choice of the vtree to use for building the SDD. In [13] the authors describe a dynamic vtree search algorithm analogous to the dynamic reordering algorithms exploited to find a good variables order for building OBDDs [14]. Also in this case, we have exploited the implementation of the algorithm available in the SDD software package [19]. It is possible to let the package decide when to invoke the dynamic vtree search algorithm, or invoke it manually. We have chosen to invoke dynamic minimization whenever the last apply made the SDD size grow more than 20%.

Even if using dynamic search can significantly improve the vtree while new clauses are added, it is also important to start the compilation process with a good vtree. In our experiments we have compared a basic strategy Π_{nat} that starts with a balanced vtree over the natural variable order (i.e., the order in which variables are found in the CNF) and a more sophisticated strategy Π_{nat}, which exploits the fact that compiling SDVs, i.e., the SD under the assumption that all components work, is much easier than compiling SD itself. The latter strategy requires that we first compile SD_V starting from a balanced vtree v_0 over the natural order of the variables, obtaining an improved vtree v_1 at the end of the compilation, thanks to dynamic search; then, we start the compilation of SD with vtree v_1.

A final remark concerns the order in which the clauses of SD are compiled. We follow the heuristic reported in [13], according to which the clauses are mapped to the nodes of the current vtree, so that each clause γ is associated with the lowest node of the vtree whose leaves cover all the literals in γ. Then, the clauses are compiled in an order induced by a depth-first visit of the vtree, so that we first compile clauses that share variables. Note that, as the vtree changes during compilation because of the dynamic search, come clauses need to be associated to the different nodes.

4.2 Comparison with OBDDs and DNNFs

We have compared the size of the compilation into SDD of some system models from the ISCAS85 benchmark with the corresponding OBDD and DNNF compilations. Note that the ISCAS85 circuits are much easier to compile if we do not model components and their possible failures; e.g., in [13], the authors report the compilation of circuit c432 into an SDD of size 9.9224, while it was not possible to fully compile the same circuit into an SDD when modeling the ok.X variables for each component.

Table 1: Compilation size of some circuits from the ISCAS85 benchmark into SDD (different starting vtrees).

<table>
<thead>
<tr>
<th>circuit</th>
<th>size</th>
<th>SDD (Π_{nat})</th>
<th>SDD (Π_{nat})</th>
</tr>
</thead>
<tbody>
<tr>
<td>c17</td>
<td>29/18</td>
<td>185</td>
<td>159</td>
</tr>
<tr>
<td>c74182</td>
<td>89/75</td>
<td>1019</td>
<td>1374</td>
</tr>
<tr>
<td>c74283</td>
<td>140/122</td>
<td>3526</td>
<td>2630</td>
</tr>
<tr>
<td>c432</td>
<td>592/254</td>
<td>2797</td>
<td>13630</td>
</tr>
<tr>
<td>c432</td>
<td>592/254</td>
<td>3965</td>
<td>35851</td>
</tr>
<tr>
<td>c432</td>
<td>592/314</td>
<td>18502</td>
<td>30566</td>
</tr>
<tr>
<td>c432</td>
<td>592/374</td>
<td>60642</td>
<td>44555</td>
</tr>
<tr>
<td>c432</td>
<td>592/394</td>
<td>35485</td>
<td>134097</td>
</tr>
<tr>
<td>c432</td>
<td>592/414</td>
<td>111155</td>
<td>2634760</td>
</tr>
<tr>
<td>c432</td>
<td>592/514</td>
<td>OOM</td>
<td>OOM</td>
</tr>
</tbody>
</table>

Table 1 shows the following information:

- circuit name (the c432^n circuits are obtained from c432 by deleting n of the 514 clauses of the complete circuit)
- number of variables and clauses
- SDD size with strategy Π_{nat}
- SDD size with strategy Π_{Y}

Unfortunately, the Π_{Y} strategy did not yield any benefits compared to the simpler Π_{nat} (it was actually harmful), so we have adopted the latter in the remainder of the experiments.

For OBDDs, we have started the compilation with two different variable orders: the natural variable order Π_{nat}; and the order Π_{SDD} corresponding to the final vtree produced by the SDD compilation. We have also used the CUDDD package [21], enabling dynamic reordering of the variables during the compilation process. Finally, for DNNFs, we have used the c2d package [22].

Table 2 shows the following information:

- circuit name
- size of the SDD produced with strategy Π_{nat}
- OBDD size with strategy Π_{nat}
- OBDD size with strategy Π_{SD}

Some comments are in order. First of all, we note that the size of the OBDDs quickly gets out of control as we move from small systems to the easiest simplified versions of c432; this confirms our expectation that SDDs can be much more succinct than OBDDs. Second, contrary to SDDs, OBDDs benefit from a good starting order (Π_{SDD} instead of Π_{nat}). Finally, we note that the DNNF sizes are not significantly better (i.e., smaller) than the SDD sizes, except for the fact that we could obtain a similar result with our system.

\[1\] We could obtain a similar result with our system.

\[2\] The c2d package actually compiles a CNF into d-DNNF form, a strict subset of DNNF.
Table 2: Compilation size of some circuits from the ISCAS85 benchmark.

<table>
<thead>
<tr>
<th>circuit</th>
<th>SDD</th>
<th>OBDD</th>
<th>OBDD</th>
<th>DNNF</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>πsat</td>
<td></td>
<td>ΠSDD</td>
<td></td>
</tr>
<tr>
<td>c17</td>
<td>185</td>
<td>109</td>
<td>133</td>
<td>174</td>
</tr>
<tr>
<td>c74182</td>
<td>1019</td>
<td>1863</td>
<td>1456</td>
<td>1742</td>
</tr>
<tr>
<td>c74283</td>
<td>3526</td>
<td>25938</td>
<td>9394</td>
<td>3692</td>
</tr>
<tr>
<td>c432^200</td>
<td>2797</td>
<td>244781</td>
<td>116888</td>
<td>6921</td>
</tr>
<tr>
<td>c432^200</td>
<td>3965</td>
<td>OOM</td>
<td>5469461</td>
<td></td>
</tr>
<tr>
<td>c432^200</td>
<td>18502</td>
<td>OOM</td>
<td>OOM</td>
<td>20465</td>
</tr>
<tr>
<td>c432^140</td>
<td>60642</td>
<td>OOM</td>
<td>OOM</td>
<td>71121</td>
</tr>
<tr>
<td>c432^120</td>
<td>35485</td>
<td>OOM</td>
<td>OOM</td>
<td>60676</td>
</tr>
<tr>
<td>c432^100</td>
<td>111155</td>
<td>OOM</td>
<td>OOM</td>
<td>898579</td>
</tr>
<tr>
<td>c432</td>
<td>OOM</td>
<td>OOM</td>
<td>OOM</td>
<td>7808820</td>
</tr>
</tbody>
</table>

Figure 5: Computation of the Filters[].

we succeeded in compiling c432 as a DNNF, but not as an SDD. Unfortunately, on 32bit OSes, the JVM has limitations on the heap size that it can use, regardless of the available physical memory; this may possibly explain the OOM we obtained when trying to compile the complete c432 system into an SDD.

5 Static Minimum Cardinality Diagnosis

In this section we describe and compare the computation of minimum-cardinality (mc) diagnoses using OBDDs and SDDs. We haven’t considered DNNFs since there is no publicly available package exposing the operations needed for DNNF diagnosis.

The computation of mc-diagnoses with OBDDs has been described in [3], and consists of the following steps:

- compile the SD into an OBDD BS
- assert the observations Obs with the restrict operator
- project the result on the COMPS variables
- compute the cardinality k of mc-diagnoses
- minimize the result of the projection with the Filters[k] OBDD, which encodes all the diagnoses of cardinality k
- enumerate the models of the resulting OBDD

The computation of Filters[] is sketched in Figure 5. The first filter Filter[0] is just the all-nominal assignment (0 faults). The next filter Filter[k] is then computed from Filter[k−1] by flipping one component from okC to ¬okC in each assignment contained in Filter[k−1].

The minimum cardinality mc(B) is simply computed by applying the recursive formula in Figure 6, where B is an OBDD representing complete assignments to the COMPS variables. If a component variable okX is ⊥, we add 1 to the cardinality count and continue with low subtree; otherwise we just continue with the high subtree; the minimum cardinality for B is the maximum between the two resulting values. Note that each node n of the OBDD graph needs only be visited once, by caching the value of mc(·) at n.

From the theoretical point of view, the most expensive step is the project of BS on the COMPS variables, since it is known that forgetting m variables is worst-time exponential in m for OBDDs, while the other steps are all guaranteed to be polynomial [16].

The computation of mc-diagnoses with SDDs can be done in a similar way to the one described for OBDDs, since SDDs also have an apply operator. In the present work, the minimize step for SDDs has been implemented in two alternative ways: with filters, as for OBDDs; and with a minimizeCardinality function available in the package [19] that performs the minimization by visiting the SDD instead of using filters. Also for SDDs, the most expensive step is project, which is worst-time exponential in the number of forgotten variables [17].

It is worth noting that, since both OBDDs and SDDs are also DNNFs, one may in principle adopt the algorithm for the static MBD of DNNFs described in [11]. Such an algorithm does not suffer from the potential blow-up of the project step, since projection is polynomial for DNNFs, and the other steps have no higher complexity than projection. In this work we could not test such an approach because of lack of an implementation of DNNF diagnosis operations. Moreover, it is important to note that, by adopting the DNNF-based algorithms, one gets the set of diagnoses (and mc-diagnoses) encoded as a DNNF even if the BS was encoded as an OBDD or SDD. This prevents post-processing manipulations of the set of diagnoses with the apply operator (see section 7).

Table 3 shows the following information:

- circuit name
- number of observed variables
- average computation time (msec) for computing mc-diagnoses with SDDs, using the minimizeCardinality function

Table 3: Average time (msec) for computing mc-diagnoses with OBDDs and SDDs.

<table>
<thead>
<tr>
<th>circuit</th>
<th>obs</th>
<th>SDD</th>
<th>SDD (filters)</th>
<th>OBDD</th>
</tr>
</thead>
<tbody>
<tr>
<td>c17</td>
<td>7</td>
<td>14.4</td>
<td>22</td>
<td>23.2</td>
</tr>
<tr>
<td>c74182</td>
<td>14</td>
<td>23.2</td>
<td>26.8</td>
<td>27</td>
</tr>
<tr>
<td>c74283</td>
<td>14</td>
<td>37.2</td>
<td>25.6</td>
<td>30.6</td>
</tr>
<tr>
<td>c432^200</td>
<td>52</td>
<td>39.4</td>
<td>39</td>
<td>110</td>
</tr>
<tr>
<td>c432^200</td>
<td>52</td>
<td>54.2</td>
<td>50.8</td>
<td>3437</td>
</tr>
<tr>
<td>c432^140</td>
<td>52</td>
<td>145.8</td>
<td>142</td>
<td>N/A</td>
</tr>
<tr>
<td>c432^120</td>
<td>52</td>
<td>262.8</td>
<td>251</td>
<td>N/A</td>
</tr>
<tr>
<td>c432^100</td>
<td>52</td>
<td>183.8</td>
<td>242.8</td>
<td>N/A</td>
</tr>
<tr>
<td>c432</td>
<td>52</td>
<td>6879.4</td>
<td>6750.2</td>
<td>N/A</td>
</tr>
</tbody>
</table>
$B = \bot$
$B = \text{ite}(\{okX, \alpha, \beta\}) \neg \{okX\} \times MD(\beta) \cup \{okX\} \times (MD(\alpha) \backslash D(\beta))$

Figure 7: Computation of the Minimal Cardinality Diagnoses $MD(B)$ of an OBDD $B$.

- average computation time (msec) for computing mc-diagnoses with SDDs, using filters
- average computation time (msec) for computing mc-diagnoses with OBDDs, using filters

We note that the computation times for SDDs do not differ much between the minimizeCardinality and filters-based versions of the diagnostic algorithm. The computation times for the smaller systems are also similar for OBDDs. However, as soon as the OBDD size is significant larger than the SDD size (resp. 2797 and 116888 for circuit $c432^{260}$), also the computation times diverge, as expected. However, the time for OBDDs raises to just 3.5sec for $c432^{260}$ (size 5M), while the time for SDDs raises to almost 7sec for $c432^{100}$ (size 1M). This suggests that the operations on OBDDs are faster than those on SDDs w.r.t. to the size of their operands. However, in our experiments, the gains in succinctness obtained with SDDs are more than compensate for the slower operations.

6 Minimal Diagnoses

Minimal diagnoses (w.r.t. set inclusion) were introduced in [11] for weak system models: a diagnosis $\Delta$ is minimal iff no diagnosis $\Delta'$ is a proper subset of $\Delta$. For weak models, minimal diagnoses make sense, since all the supersets of a diagnosis are necessarily diagnoses, thus it is useless to report non-minimal diagnoses.

Unfortunately, the number of minimal diagnoses can be much larger than that of mc-diagnoses, and exponential in the system size, so a way to compute all of them as an OBDD/SDD is particularly appealing. Indeed, it not only enables the user to extract them in cardinality order, from lowest to highest, stopping at any desired level; but it also allows querying the set of minimal diagnoses (through symbolic operations) to figure out, e.g., if there are any diagnoses with a given cardinality $k$, or if there’s any minimal diagnosis involving a given component $C$.

OBDDs have been shown to support many symbolic operations that rely on a recursive visit of the BDD, e.g., finding prime implicants [23] or prime irredundant covers of the encoded function [24]. Figure 7 shows a possible recursive formula for computing minimal diagnoses $MD(B)$ of an OBDD $B$ that encodes all the diagnoses (in the formula, $D(\cdot)$ denotes the set of all diagnoses encoded by an OBDD). Note that we use set operations such as $\cap$, $\cup$, $\times$ in the formula, since it is well known that they can be implemented with OBDDs [25].

An interesting research question is in asking whether a similar recursive algorithm could be devised for SDDs. Since primes play for SDDs the role of decision variables for OBDDs, the recursive calls should likely be based on primes, but it is not at all obvious whether such a modification is feasible in a meaningful and efficient way. Luckily, it is also possible to obtain the minimal diagnoses through symbolic operations that exploit filters (section 5).

Figure 8: Computation of the Minimum Cardinality Diagnoses with filters.

7 Other MBD Tasks

In this section, we first consider MBD when the same static system is observed in different contexts, and then move to fully dynamic diagnosis. To the best of our knowledge, also these tasks cannot be addressed with DNNF compilation, so they represent opportunities to exploit potential benefits of the succinctness of SDDs compared to OBDDs.

7.1 Multiple observations

The simplest variation on the static MBD problem is a multiple-observations diagnostic problem, where a single $SD$ (assumed to be static) is observed several times under different contexts (e.g. system inputs):

$DP^* = (SD, \{Obs_1, \ldots, Obs_k\})$

Clearly, a diagnosis for $DP^*$ must be a diagnosis for the sub-problems $DP_i = (SD, Obs_i), \ldots, DP_k = (SD, Obs_k)$. Assuming model compilation into OBDD/SDD, we must perform the following steps:

- compile the system model into OBDD/SDD
- for each observation $Obs_i$, compute the OBDD/SDD representing all the diagnoses for the given observation
- intersect the resulting OBDDs/SDDs by using the apply operator
The renaming requires the use of a linear-time operator following high-level steps:

- set of assignments to the state variables
- OBDD/SDD operators, including
- apply
- start by computing the belief state $B$ through the manipulation of and on the computation of minimum cardinality diagnoses resulting trajectories (see [7]).
- operations followed by an output-linear enumeration of the
- $S$ and $\Pi$ and on the initial state of the system (i.e., set of assignments to the state variables $\Pi$ at time $t$) with the following high-level steps:

- create an OBDD/SDD representing the belief state $B_{\text{cur}}$ at time 0 (possibly by exploiting constraints given on the initial state of the system)
- compute an OBDD/SDD representing the set of consistent assignments $\Sigma_{\text{cur}}$ to $\Pi_{\text{cur}} \cup D_{\text{cur}}$ variables by asserting states $B_{\text{cur}}$, observations $O_{\text{cur}}$ and commands $C_{\text{cur}}$ on model $M$; this is done in a similar way as the computation of diagnoses for static systems
- initialize an OBDD/SDD representing the belief state $B_{\text{next}}$ to time 1 by intersecting $\Sigma_{\text{cur}}$ with $M_T$, and forgetting the variables except the new state variables $\Pi_{\text{next}}$
- rename each variable $y_{\text{next}}$ in $B_{\text{next}}$ to $y_{\text{cur}}$, so that the process can be iterated without introducing fresh variables for each time instant
- iterate the process for each time point when the system has been observed

The above steps can be performed by using the basic OBDD/SDD operators, including apply (see [7] for details). The renaming requires the use of a linear-time operator rename that is available both for OBDDs and SDDs.

As said above, the outlined steps compute a sequence of belief states $B_1, \ldots, B_t$ in symbolic form (OBDD or SDD), but diagnoses are defined as sequences of states in section 3.2. Extracting the ( preferred) trajectories from the sequence of belief states can also be done with symbolic operations followed by an output-linear enumeration of the resulting trajectories (see [7]).

8 Conclusions

In the present paper we have analyzed the application of SDDs to the MBD task. In particular, we have focused on the compilation of a static system model into an SDD $S_{SD}$ and on the computation of minimum cardinality diagnoses through the manipulation of $S_{SD}$. For these tasks, we have done some comparisons with OBDDs, showing that the succinctness of SDDs can more than repay the slightly inferior efficiency of SDD operations. Though we haven’t implemented the mc-diagnosis task with DNNFs, if we restrict our attention to it the winner are likely to be DNNFs, since they are more succinct than SDDs, and support all the needed operations very efficiently.

Things change, however, when we consider more complex tasks, in particular those that require the use of the apply operator to manipulate the set of diagnoses at the symbolic level: such an operator is in fact supported by OBDDs and SDDs, but not DNNFs. While we have discussed some of the more complex tasks in detail, there are many more related to MBD that could be mentioned, e.g., diagnosability analysis, sensor placement, and model abstraction, to name a few. Analyzing the computation of minimal diagnoses w.r.t. set inclusion has been particularly interesting, since it has pointed out that some methods used for OBDDs, that rely on recursion guided by the decision variable at the root, do not have an obvious translation into equivalent ones with SDDs.

To the best of our knowledge, this is the first work considering the use of SDDs for the MBD task. Previous related works have compared DNNFs with OBDDs [11; 26], reaching the conclusion that DNNFs are significantly better at the mc-diagnosis task. However, there are many interesting research questions left unanswered, that should be addressed in future works. First of all, the compilation with SDDs and OBDDs should be compared with more comprehensive experiments, and also target the compilation of dynamic system models. Second, we should compare the performance of SDDs and OBDDs on more complex tasks than mc-diagnosis of static systems, especially when the difference in size between the two is not too big. Finally, but not less interestingly, we should study in depth, both theoretically and experimentally, the computation of minimal diagnoses with SDDs versus OBDDs.

References


---

We assume that commands are also known.


