Implementation of belief change operators using BDDs

Nikos Gorogiannis and Mark D. Ryan ({nkg,mdr}@cs.bham.ac.uk)
School of Computer Science
University of Birmingham
Birmingham B15 2TT
UK

Abstract. While the theory of belief change has attracted a lot of interest from researchers, work on implementing belief change and actually putting it to use in real-world problems is still scarce. In this paper, we present an implementation of propositional belief change using Binary Decision Diagrams. Upper complexity bounds for the algorithm are presented and discussed. The approach is presented both in the general case, as well as on specific belief change operators from the literature. In an effort to gain a better understanding of the empirical efficiency of the algorithms involved, a fault diagnosis problem on combinational circuits is presented, implemented and evaluated.

Keywords: belief revision, binary decision diagrams, fault diagnosis

1. Introduction

When an agent acquires information which contradicts its current beliefs, it is obliged to give up some of its beliefs in order to accommodate the new information and remain consistent. The operation of consistently incorporating new information into a belief state by removing some of the old beliefs is called *belief revision*. The seminal work on belief revision was done by Alchourrón, Gärdenfors and Makinson (see, e.g., [7]). The AGM theory, as it is known, proposes a set K1–K8 of rationality postulates which any belief revision operator ought to satisfy. More recently, a number of subtly different forms of revision have been distinguished, such as *update* [11]. While revision is used to model the evolution of belief about a static world, update models the same process in a changing world.

As well as work on rationality postulates, several authors have presented specific revision or update operators [17, 6, 18, 2, 15]. There has also been work on applications of belief revision beyond the modelling of artificial agents. For example, applications of belief revision in *fault diagnosis* have been proposed [6, 18]. We examine this application later in this paper.

In this paper we are concerned with implementation of belief change operators in a finite, propositional language. An important decision in any implementation of belief change concerns the choice of representation for belief states. Among the desiderata for such representations, one may include their representational compactness, syntax independence and overall efficiency.

The goal of this paper is to explore implementations of belief change operators on propositional logic by means of a data structure known as the *Binary Decision Diagram* (BDD), thus addressing the above criteria. BDDs are widely known because of their use in *model checking*, a hardware verification technique which works by exhaustive state-space exploration. In that context, their usage has led to a dramatic improvement in the efficiency of model checking implementations, and therefore in the size of model that can realistically be explored [13, 5].

The paper is structured as follows. We introduce BDDs and their operations in the next section. In section 3, we review belief revision, and in section 4 we implement some belief revision operators in terms of BDDs, studying their complexity. Section 5 is devoted to a substantial example based on fault diagnosis, and our conclusions are presented in section 6. Due to space limitations, a longer version of this paper is available at `ftp://ftp.cs.bham.ac.uk/pub/authors/M.D.Ryan/01-sl.ps.gz`.

2. Binary Decision Diagrams

2.1. Definitions and Basic Results

Binary Decision Diagrams (BDDs) are a compact and empirically efficient data structure for representing formulas in propositional logic. The *decision tree* for the formula $x \vee y$ is shown in figure 1(a). The dotted lines denote the path to be taken when a node is false, and the solid lines when it is true. The decision tree shows four paths, corresponding to the four possible values of x and y, and the leaves show the resulting truth value of the formula in those cases. Decision trees thus code up the truth-table for the formula. They are not space-efficient, having $2^{n+1}-1$ nodes when the number of atomic propositions in the formula is n.

The BDD for $x \vee y$ is shown in figure 1(b). It is obtained by folding together shared subtrees in the decision trees, and removing redundant decision nodes. BDDs can be much more compact than the corresponding decision trees. For example, the BDD for $((p \vee q) \wedge r) \vee s$, shown in figure 1(c), contains 6 nodes, while the corresponding tree contains 31 nodes. In the worst case, BDDs can still have $O(2^n)$ nodes. However, BDDs have been extensively used in verification where they appear to be a compact representation in practice.

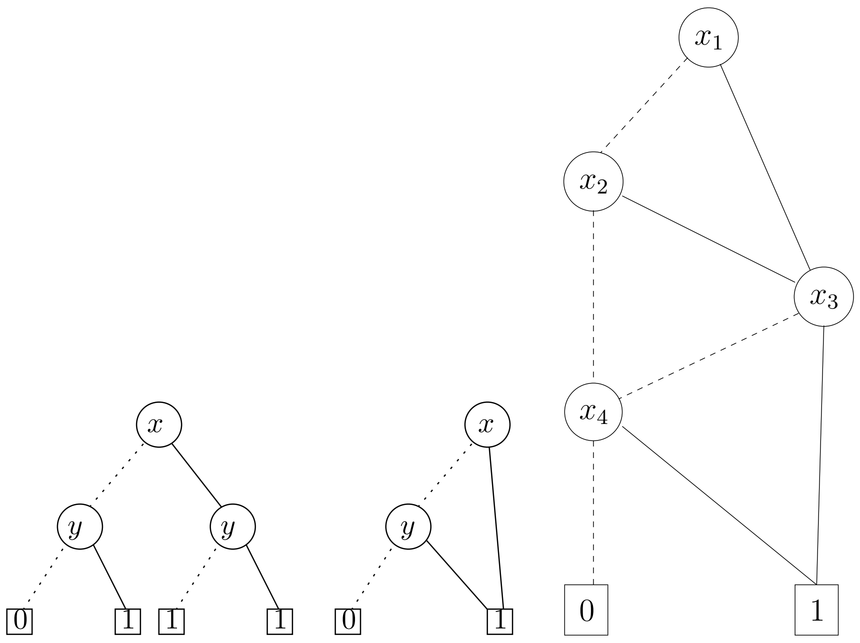

Figure 1. (a) The decision tree and (b) the BDD for the formula $x \vee y$. (c) The BDD for the formula $((p \vee q) \wedge r) \vee s$.

Both decision trees and BDDs assume a fixed ordering of the variables into layers. The size of the decision tree is independent of that ordering, but the size of the BDD is not; the space-economy introduced by sharing subdiagrams can depend on the ordering of the variables.

A BDD is fully *reduced* if it has no redundant decision points and no isomorphic subdiagrams. There is an efficient algorithm, called `reduce`, for reducing a decision tree or partly-reduced BDD into its fully-reduced form. Once reduced, BDDs are *canonical*: that means that there is a unique reduced BDD for a given formula with respect to a fixed variable ordering. More detailed information about BDDs and their algorithms can be found in [1, 5] or the book [8].

2.2. Algorithms on Binary Decision Diagrams

After converting a formula to a BDD, that BDD can be manipulated using several algorithms that implement logical operations. Some of these algorithms are presented below along with their complexity characteristics.

2.2.1. *Tautology, satisfiability and equivalence checking*

Because of the canonicity of BDDs, it is easy to check whether a BDD represents a tautology, or an unsatisfiable formula. Every tautology is represented by the same BDD, namely, the BDD with a single node, the terminal 1. Thus, tautology checking is a constant-time operation.

In the same spirit, a formula is satisfiable if its BDD representation is not the terminal node 0. Again, this results in a constant-time operation. It follows from these observations that the *conversion of a formula to BDD form is NP- and coNP-hard in the length of the formula.* Consequently, since it is widely believed that NP$\neq$coNP, it is probably the case that the problem of converting a formula to a BDD is not a member of NP$\cup$coNP.

The canonicity property of BDDs implies that checking if two formulas are equivalent by comparing their BDDs is very efficient. In BDD packages like CUDD [16] or BuDDy [12], this can be done by pointer comparison (and hence in constant time).

2.2.2. *The algorithms `apply`, `negate` and `restrict`*

Given two BDDs representing the formulas ϕ and ψ (having $|\phi|$ nodes and $|\psi|$ nodes respectively), together with a binary connective $\bullet$, the algorithm `apply` computes the BDD for $\phi \bullet \psi$. The worst-case complexity of `apply` is $O(|\phi| \cdot |\psi|)$ and it is known to be a tight bound [3].

Given the BDD for ϕ, the algorithm `negate` computes the BDD for $\neg\phi$ by using `apply` and the $\rightarrow$ operator: $\neg\phi = \phi \rightarrow \bot$. Thus its complexity is $O(|\phi| \cdot 1) = O(|\phi|)$.[1]

These two algorithms provide a way for converting a formula to a BDD, without creating the decision tree and then reducing it to BDD form. The BDD representation of a propositional variable is a tree with three nodes, the root labelled by the variable and the two terminal nodes, 1 and 0. Using these and the algorithms `apply` and `negate`, a formula can be recursively converted to the equivalent BDD. Indeed, this is the only algorithm for conversion used in practice, since converting a formula to its decision tree is *always* an exponential operation in the number of variables, whereas conversion using `apply` is expensive only in the worst case.

The result $\phi[C/p]$ of the substitution of a variable p by a boolean constant C can be computed with the algorithm `restrict`. The worst-case complexity is $O(|\phi|)$ (see [3]). As noted in the same paper, the

[1] Note that `negate` could be implemented as a constant-time operation, by swapping the terminal nodes. However, for reasons of efficiency, most BDD packages use the same terminal nodes for all stored BDDs, all of which would be negated if the terminal nodes were to be swapped.

algorithm can be modified to perform a specific number of restrictions simultaneously without affecting its complexity.

2.2.3. *The algorithms `exists` and `forall`*

The formulas $\forall p.\, \phi$ and $\exists p.\, \phi$ are defined as

$$\begin{aligned} \forall p.\, \phi &= \phi[\top/p] \wedge \phi[\bot/p] \\ \exists p.\, \phi &= \phi[\top/p] \vee \phi[\bot/p] \end{aligned}$$

The BDDs for $\forall p.\, \phi$ and $\exists p.\, \phi$ can be computed from the BDD for ϕ by the algorithms `apply` and `restrict`, with complexity $O(|\phi|^2)$ (see [4]). Consecutive quantification over k variables using this algorithm results in an upper bound for the worst-case complexity, of $O(|\phi|^{2^k})$.

McMillan, in [13], describes the `andExists` algorithm, for computing an operation that occurs very often in model checking and which plays a central role in our formulation of propositional belief change. Let ϕ and ψ be two BDDs. The algorithm computes the consecutive existential quantification over a specified vector of variables, of the conjunction $\phi \wedge \psi$, but without explicitly forming the BDD for it. An upper bound on the time complexity of this algorithm is $O(|\phi| \cdot |\psi| \cdot 2^{2n})$, where n is the total number of variables appearing in ϕ and ψ. However, intuition and empirical evidence both suggest the existence of a smaller bound. The resulting BDD has a size bounded by the general worst-case of the result, i.e. $O(2^{n-k})$. McMillan also proves that the computation of the BDD expressing an existential quantification over n variables, is NP-complete.

The universal quantification can be computed by using the fact that $\forall \equiv \neg\exists\neg$ and the algorithm `negate`, giving the same complexity. The dual algorithm to `andExists`, `impliesForall`, is derivable from `andExists` and `negate`, having again the same complexity bound.

2.2.4. *The algorithm `replace`*

As we see later, we often need to replace some variables in a BDD by other variables, corresponding to substitution in logic. This is a linear-time operation if the BDD resulting from the substitution obeys the variable ordering chosen. If it does not, then re-ordering is necessary and in general this can take exponential time.

2.3. Expression syntax for BDDs

We use a bold-face logical notation to denote the algorithms of the preceding section, as summarised in the table below. These algorithms will be used to describe belief change operators. We now present some derived algorithms which will be useful for that purpose. These can be thought of as macros.

algorithm (with arguments)	notation
`apply`$(B_1, B_2, \to)$	$B_1 \to B_2$
`negate`(B)	$\neg B$
`exists`$(\mathbf{p}, B)$	$\exists \mathbf{p}.\, B$
`andExists`$(\mathbf{p}, B_1, B_2)$	$\exists \mathbf{p}.\, (B_1 \wedge B_2)$
`replace`$(\mathbf{p}, \mathbf{p}', B)$	$B[\mathbf{p}'/\mathbf{p}]$

We have seen how BDDs represent formulas by representing the set of models that satisfy them. To implement some belief change operators, we need to be able to represent relations on models as BDDs. A relation can be thought of as a function which, given two models, returns a boolean value. Therefore, it can be represented as a BDD over two copies of the atomic propositions, which we call *unprimed* and *primed*, and write as $\mathbf{p}, \mathbf{p}'$.

For example, consider the ordering $\leq$ shown in figure 2 over the four models $\{\overline{pq}, \overline{p}q, p\overline{q}, pq\}$ of the language $\{p, q\}$. Its BDD is also shown in the figure. To determine whether $m \leq m'$, we supply the truth values $\mathbf{p}$ for m and $\mathbf{p}'$ for m' to the BDD and get a boolean value result.

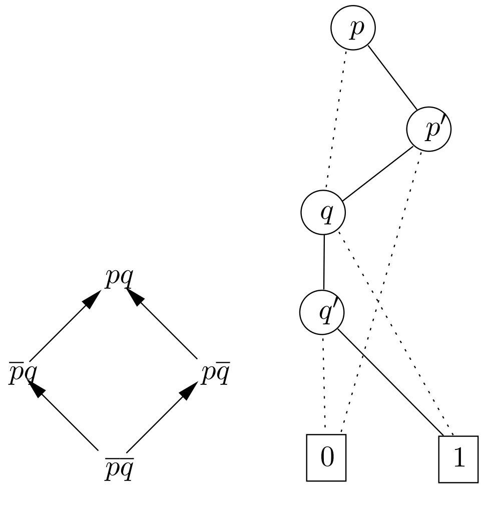

Figure 2. An ordering on the models of the language $\{p, q\}$, and its BDD.

If B_R is a BDD representing a relation R over unprimed and primed variables, then the BDD for the inverse relation is obtained by simultaneously renaming the unprimed variables to primed, and the primed ones to unprimed. We write this as $B_R[\mathbf{p}/\mathbf{p}', \mathbf{p}'/\mathbf{p}]$. The strict counterpart of the relation R is given mathematically as $R \cap \overline{R^{-1}}$. Thus, the

BDD for the strict counterpart is given by

$$\texttt{strict}(B_R) = B_R \land \neg(B_R[\mathbf{p}/\mathbf{p}', \mathbf{p}'/\mathbf{p}])$$

Note that the swapping of the primed and unprimed variables will necessitate re-ordering the variables, and is therefore an expensive operation. This is the only instance of variable replacement in the paper which does not respect the ordering of variables; all the other replacements can be performed in linear-time.

The R-minimal elements of X are defined as

$$\min_R(X) = \{w \in X \mid \forall v \in X\ v\overline{R^<}w\}$$

where $R^<$ is the strict counterpart of R. The BDD algorithm for $\min_R(X)$, in terms of the BDDs B_R, B_X for R and X, can be written as

$$\texttt{min}(B_R, B_X) = ((\forall\mathbf{p}.\,(B_X \rightarrow \neg\texttt{strict}(B_R)))[\mathbf{p}/\mathbf{p}']) \land B_X$$

If the relation R is known to be total, then the minimal set of elements can be written more simply, and this permits an optimisation in the way we calculate the BDD for `min`. If R is total, then

$$\min_R(X) = \{w \in X \mid \forall v \in X\ wRv\}$$

and therefore the BDD for `min` need not use `strict`:

$$\texttt{min}(B_R, B_X) = (\forall\mathbf{p}'.\,(B_X[\mathbf{p}'/\mathbf{p}] \rightarrow B_R)) \land B_X$$

2.4. Upper Bounds of BDD Size Based on Circuit Implementations

The main theorem for proving upper bounds of the size of some BDDs that appear in the following sections, proved in [13], is presented in this section.

Let ϕ be an n-ary boolean function and suppose a logical circuit computing ϕ is given. This circuit will contain a number m of blocks that are either gates (binary or otherwise) or primary inputs (inputs are counted as blocks with zero inputs and one output). Let a *linear order* of the circuit be a numbering of the blocks from 1 to m, with the block producing the primary output numbered last. Then, the *forward cross section at block i* is the total number of wires from an output of a block j such that $j < i$ to an input of a block k such that $i \leq k$. The *forward width w_f of the circuit* (with respect to the linear order chosen) is defined as the maximum forward cross section for all blocks.

Similarly, the *reverse cross section at block i* is the total number of wires from an output of a block j such that $j > i$ to an input of a block k such that $i \geq k$. The *reverse width w_r of the circuit* (again with respect to the linear order) is defined as the maximum reverse cross section at any block. Then, the following theorem holds:

THEOREM 1 ([13]). *If a circuit computing function ϕ has forward width w_f and reverse width w_r for some linear order L, then there is a BDD representing function ϕ of size bounded by $n2^{w_f 2^{w_r}}$, where n is the number of inputs of the circuit.*

3. Belief Change

3.1. Belief Revision

Belief revision refers to the process of incorporating new knowledge in an agent's prior beliefs, even when the new information contradicts the previous ones. Agents are said to be in an epistemic state, representing their beliefs and any other relevant epistemic information. The change of epistemic state in the light of new information is the phenomenon that revision is supposed to explain.

The seminal work in belief revision is that of Alchourrón, Gärdenfors and Makinson (see, e.g., [7]). They modelled epistemic states as sets of formulas closed under consequence, and proposed a set of rationality postulates K1–K8 which they argue any revision operator ought to satisfy.

Katsuno and Mendelzon [9] have studied the case where the propositional language is finite. In that case, epistemic states may be modelled as propositional formulas instead of consequence-closed theories. This setting is rather simpler; since we are interested in implementations, and any implementation necessarily involves only finitely many atomic propositions, we adopt the setting of Katsuno and Mendelzon.

The revision operator $\circ : \mathcal{L} \times \mathcal{L} \to \mathcal{L}$ takes two formulas and returns another formula. The formula $\phi \circ \psi$ represents the epistemic state resulting from revising ϕ with ψ; intuitively, this is intended to be ψ together with whatever 'parts' of ϕ can be consistently retained. Katsuno and Mendelzon formulate a set of postulates R1–R6 which, for finite languages, are equivalent to the AGM postulates K1–K8. The intention of the postulates is to encode minimal change, and this can be made precise by the following theorem. Consider a function that assigns to each formula ψ a total pre-order $\leq_\psi$ on interpretations, that is, a binary relation on the set of interpretations $\mathcal{U}$ that is transitive, reflexive and total. This function is called a faithful assignment if and only if the following hold (where mod(ψ) is the set of models of ψ):

F1. If $w, v \in \text{mod}(\psi)$, then $w <_\psi v$ does not hold.

F2. If $w \in \text{mod}(\psi)$ and $v \notin \text{mod}(\psi)$ then $w <_\psi v$ holds.

F3. If $\psi \equiv \phi$ then $\leq_\psi = \leq_\phi$.

Then, Katsuno and Mendelzon prove the following representation theorem:

THEOREM 2 ([10]). *A revision operator $\circ$ satisfies conditions R1–R6 if and only if there exists a faithful assignment that maps each formula ψ to a total pre-order $\leq_\psi$ such that*

$$\text{mod}(\psi \circ \mu) = \min_{\leq_\psi}(\text{mod}(\mu))$$

3.2. Belief Update

In [11], Katsuno and Mendelzon make an important distinction on the meaning of some belief change operators. They argued that not all belief changes are revisions, i.e. incorporation of new information about a static world. They identified and characterised with a set of postulates and a representation theorem, the form of belief change they call belief update. This kind of belief change aims to integrate new information with an agent's prior beliefs about a *changing* world. In the spirit of the generic approach, they list a number of postulates U1–U8 that any update operator $\diamond$ should satisfy [11].

In the context of belief updates, a function that maps each model w to a *partial* pre-order $\leq_w$ is called a faithful assignment if it satisfies the following condition:

– For any models $w, v \in \mathcal{U}$, if $w \neq v$ then $w <_w v$.

Under this framework, the following representation theorem holds:

THEOREM 3 ([11]). *An update operator $\diamond$ satisfies conditions U1–U8 if and only if there exists a faithful assignment that maps each interpretation w to a partial pre-order $\leq_w$ such that*

$$\text{mod}(\psi \diamond \mu) = \bigcup_{w \in \text{mod}(\psi)} \min_{\leq_w}(\text{mod}(\mu))$$

4. Implementing belief change operators as BDD algorithms

We saw in section 2 that propositional formulas may be represented as BDDs, and that the `apply` algorithm can be used to implement the binary operators $\wedge$, $\rightarrow$ etc, while the `negate` algorithm implements $\neg$. This section is concerned with the implementation of revision operators and update operators.

4.1. Revision defined by faithful assignment

Theorem 2 tells us how to compute $\phi \circ \psi$, given a faithful assignment. To implement this definition, we represent relations on models as BDDs in the manner described in section 2.3. Theorem 2 assumes a faithful assignment which, given a formula ϕ, returns an ordering $\leq_\phi$. Therefore, we assume an operation `fa` taking a BDD over $\mathbf{p}$, which represents ϕ, and returning a BDD over $\mathbf{p}, \mathbf{p}'$, representing $\leq_\phi$.

By the theorem, $\text{mod}(\psi \circ \mu) = \min_{\leq_\psi}(\text{mod}(\mu))$. Therefore, given BDDs B_ψ, B_μ for formulas ϕ, μ, we can compute the BDD for $\psi \circ \mu$ as

$$\texttt{min}(\texttt{fa}(B_\psi), B_\mu)$$

where the operator `min` on BDDs is described in section 2 *for total relations.*

Suppose that the number of propositional variables is n (*in each copy of the variables*), the worst-case time complexity of the operation `fa` is given as $|\texttt{fa}|$ and also, the size of the resulting BDD as $|\texttt{fa}(B_\psi)|$. By expanding the macros in the above formula we get:

$$B_\mu \wedge \forall \mathbf{p}'.\, (B_\mu[\mathbf{p}'/\mathbf{p}] \rightarrow \texttt{fa}(B_\psi))$$

An upper bound for the worst-case complexity of the revision can be computed as follows:

Operation	Time Complexity	Result Size
$B_\mu[\mathbf{p}'/\mathbf{p}]$	$O(\|B_\mu\|)$	$O(\|B_\mu\|)$
`fa`	$\|\texttt{fa}\|$	$\|\texttt{fa}(B_\psi)\|$
$\forall \mathbf{p}'.\, (\cdot \rightarrow \cdot)$	$O(\|B_\mu\| \cdot \|\texttt{fa}(B_\psi)\| \cdot 2^{4n})$	$O(2^n)$
$B_\mu \wedge$	$O(\|B_\mu\| \cdot 2^n)$	$O(2^n)$

Thus, an upper bound of the complexity of the whole operation is:

$$O(\max\{|\texttt{fa}|,\ |B_\mu| \cdot |\texttt{fa}(B_\psi)| \cdot 2^{4n}\})$$

This upper bound measure may not be indicative of the true situation because:

- Empirical evidence in the context of model checking indicates that the average-case complexity of these operations is much lower than their worst case, but it is very hard to formulate in a precise sense what exactly the average case is.

- This measure depends crucially on the complexity of the BDD representation of the ordering on models. All of the model-based operators proposed define orderings on models that require at least one quantification (for examples see the following sections). Thus, different specific revision strategies will yield very different concrete complexities.

In the following subsections, we look at some specific belief revision operators defined in the literature.

4.2. Borgida

An interpretation v can be thought as a set containing only the propositional variables that hold in v. The symmetric set-difference $v \triangle w$ of two interpretations v and w, is the set containing all the propositional variables whose values differ in v and in w. Given a formula μ and an interpretation v, the set of differences of v and μ can be defined as:

$$\mathrm{diff}(v, \mu) \overset{\mathrm{def}}{=} \{v \triangle w \mid w \in \mathrm{mod}(\mu)\}$$

Borgida introduced a revision operator in [2] that orders interpretations according to the set-inclusion of symmetric set-differences. The definition of $\psi \circ \mu$ has two main parts:

- If $\psi \wedge \mu$ is consistent, then $\psi \circ \mu = \psi \wedge \mu$ (R2).

- Otherwise, w is a model of $\psi \circ \mu$ if there is a model v of ψ, such that
$$v \triangle w \in \min_{\subseteq}(\mathrm{diff}(v, \mu))$$

Borgida's revision is known to satisfy R1-R5 but not R6 (see [10]). As such, it is not definable by a faithful assignment. Let us look how this is implemented in BDDs.

If $\psi \wedge \mu$ is inconsistent then

$$\begin{aligned}\mathrm{mod}(\psi \circ \mu) &= \{w \mid \exists v.\, (v \in \mathrm{mod}(\psi) \wedge v \triangle w \in \min_{\subseteq}(\mathrm{diff}(v, \mu)))\} \\ &= \{w \mid w \in \mathrm{mod}(\mu) \wedge \exists v.\, (v \in \mathrm{mod}(\psi) \wedge \\ &\qquad \forall z.\, (z \in \mathrm{mod}(\mu) \rightarrow v \triangle z \not\subset v \triangle w))\}\end{aligned}$$

The symmetric set-difference of two interpretations can be expressed as a boolean operation (where $(v \triangle w)_i$ is the i-th propositional variable of the symmetrical set-difference between v and w)

$$(v \triangle w)_i = \neg(v_i \leftrightarrow w_i)$$

Set-inclusion of symmetric set-differences is, then, expressed as

$$v \triangle z \subseteq v \triangle w \ \text{ iff } \bigwedge_{i=1}^{n} \neg(v_i \leftrightarrow z_i) \rightarrow \neg(v_i \leftrightarrow w_i)$$

and consequently, strict inclusion as

$$\begin{aligned} v \triangle z \subset v \triangle w \ \text{ iff } & \left(\bigwedge_{i=1}^{n} \neg(v_i \leftrightarrow z_i) \rightarrow \neg(v_i \leftrightarrow w_i) \right) \wedge \\ & \neg \left(\bigwedge_{i=1}^{n} \neg(v_i \leftrightarrow w_i) \rightarrow \neg(v_i \rightarrow z_i) \right) \end{aligned}$$

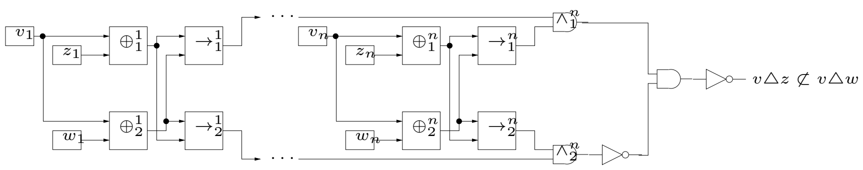

Figure 3. Circuit to decide $v \triangle z \not\subset v \triangle w$.

A circuit to compute $v \triangle z \not\subset v \triangle w$ based on these equations is presented in figure 3, where $\oplus$ is the xor-gate and $\rightarrow$ the implies-gate. We define the following ordering on gates and inputs:

$$\begin{array}{llllllllll} v_1, & z_1, & \oplus_1^1, & w_1, & \oplus_2^1, & \rightarrow_1^1, & \rightarrow_2^1, & & & \\ v_2, & z_2, & \oplus_1^2, & w_2, & \oplus_2^2, & \rightarrow_1^2, & \rightarrow_2^2, & \wedge_1^2, & \wedge_2^2, & \\ \vdots & & & & & & & & & \\ v_{n-1}, & z_{n-1}, & \oplus_1^{n-1}, & w_{n-1}, & \oplus_2^{n-1}, & \rightarrow_1^{n-1}, & \rightarrow_2^{n-1}, & \wedge_1^{n-1}, & \wedge_2^{n-1}, & \\ v_n, & z_n, & \oplus_1^n, & w_n, & \oplus_2^n, & \rightarrow_1^n, & \rightarrow_2^n, & \wedge_1^n, & \wedge_2^n, & \wedge, \neg, \wedge, \neg \end{array}$$

It is easy to check that the forward cross section at each gate or input of the circuit is at most C where C is a constant, that is, it does not depend in any way on n. Thus, by the theorem in section 2.4, there exists a BDD B_R representing this circuit, i.e. the negation of the strict part of the ordering, of size $O(n)$.

Assuming that the BDD B_R has variables $\mathbf{p}, \mathbf{p}', \mathbf{p}''$ for w, v, z respectively, the BDD algorithm implementing Borgida's revision will be:

1. Compute $B_\psi \wedge B_\mu$ and check it for consistency. If it is consistent, then this is also the result of the revision (overall complexity is $O(|B_\psi| \cdot |B_\mu|)$.

2. Otherwise, the result will be:

$$B_\mu \wedge \exists \mathbf{p}'. (B_\psi[\mathbf{p}'/\mathbf{p}] \wedge \forall \mathbf{p}''. (B_\mu[\mathbf{p}''/\mathbf{p}] \rightarrow B_R))$$

Upper bounds for the complexity of these operations are shown below:

Operation	Time Complexity	Result Size
$B_\mu[\mathbf{p}''/\mathbf{p}]$	$O(\|B_\mu\|)$	$O(\|B_\mu\|)$
$\forall \mathbf{p}''. (\cdot \rightarrow \cdot)$	$O(\|B_\mu\| \cdot n \cdot 2^{6n})$	$O(2^{2^n})$
$B_\psi[\mathbf{p}'/\mathbf{p}]$	$O(\|B_\psi\|)$	$O(\|B_\psi\|)$
$\exists \mathbf{p}'. (\cdot \wedge \cdot)$	$O(\|B_\psi\| \cdot 2^{2n} \cdot 2^{4n})$	$O(2^n)$
$B_\mu \wedge \cdot$	$O(\|B_\mu\| \cdot 2^n)$	$O(2^n)$

Therefore, the worst-case time complexity of Borgida's revision is at most $O(|B_\mu| \cdot n2^{6n})$.

4.3. Satoh

Given two formulas ψ and μ, the set of differences of ψ and μ is defined as

$$\text{diff}(\psi, \mu) \overset{\text{def}}{=} \bigcup_{v \in \text{mod}(\psi)} \text{diff}(v, \mu)$$

The revision operator proposed by Satoh in [15] is defined in first-order logic. Its restriction to finite propositional logic, as described in [10] is a "global" version of Borgida's revision. When revising ψ by μ, instead of considering individually the models of ψ, Satoh's notion of minimality relies on both ψ and μ simultaneously. An interpretation w is a model of $\psi \circ \mu$ if there exists a model v of ψ such that $v \triangle w$ is a minimal element of diff(ψ, μ). Satoh's revision is known to satisfy R1-R5 but not R6 (proved in [10]).

It is easy to express Satoh's revision as a BDD algorithm, using much of the construction presented above for Borgida's operator. The set of minimal pairs $\min_\subseteq(\text{diff}(\psi, \mu))$ can be expressed as

$$\begin{aligned}\min_\subseteq(\text{diff}(\psi, \mu)) \;=\; & \{v \triangle w \mid v \in \text{mod}(\psi) \wedge w \in \text{mod}(\mu) \wedge \\ & \forall x \forall y. (x \in \text{mod}(\psi) \wedge y \in \text{mod}(\mu) \rightarrow x \triangle y \not\subset v \triangle w)\}\end{aligned}$$

Therefore the models of the revision are:

$$\begin{aligned}\text{mod}(\psi \circ \mu) \;=\; & \{w \mid w \in \text{mod}(\mu) \wedge \exists v. (v \in \text{mod}(\psi) \wedge \\ & \forall x \forall y. (x \in \text{mod}(\psi) \wedge y \in \text{mod}(\mu) \rightarrow x \triangle y \not\subset v \triangle w))\}\end{aligned}$$

It is trivial to modify the circuit for Borgida's ordering to produce one that decides $x\triangle y \not\subset w\triangle v$. Therefore, there exists a BDD B_R of size $O(n)$ that represents this ordering. We assume that B_R contains variables $\mathbf{p}, \mathbf{p}', \mathbf{p}'', \mathbf{p}'''$ that correspond to w, v, x, y respectively. Then, the BDD algorithm is

$$B_\mu \wedge \exists \mathbf{p}'.\, (B_\psi[\mathbf{p}'/\mathbf{p}] \wedge \forall \mathbf{p}'', \mathbf{p}'''.\, (B_\psi[\mathbf{p}''/\mathbf{p}] \wedge B_\mu[\mathbf{p}'''/\mathbf{p}] \rightarrow B_R))$$

Operation	Time Complexity	Result Size
$B_\psi[\mathbf{p}''/\mathbf{p}], B_\psi[\mathbf{p}'/\mathbf{p}]$	$O(\lvert B_\psi\rvert)$	$O(\lvert B_\psi\rvert)$
$B_\mu[\mathbf{p}'''/\mathbf{p}]$	$O(\lvert B_\mu\rvert)$	$O(\lvert B_\mu\rvert)$
$B_\psi[\mathbf{p}''/\mathbf{p}] \wedge B_\mu[\mathbf{p}'''/\mathbf{p}]$	$O(\lvert B_\psi\rvert \cdot \lvert B_\mu\rvert)$	$O(\lvert B_\psi\rvert \cdot \lvert B_\mu\rvert)$
$\forall \mathbf{p}'', \mathbf{p}'''.\, (\cdot \rightarrow \cdot)$	$O(\lvert B_\psi\rvert \cdot \lvert B_\mu\rvert \cdot n \cdot 2^{8n})$	$O(2^{2n})$
$\exists \mathbf{p}'.\, (\cdot \wedge \cdot)$	$O(\lvert B_\psi\rvert \cdot 2^{2n} \cdot 2^{4n})$	$O(2^n)$
$B_\mu \wedge \cdot$	$O(\lvert B_\mu\rvert \cdot 2^n)$	$O(2^n)$

an upper bound for the worst-case complexity of which is $O(|B_\psi| \cdot |B_\mu| \cdot n2^{8n})$.

4.4. Dalal

The revision operator proposed in [6] takes the distance between two interpretations to be the cardinality of their symmetric set-difference (also known as the *Hamming distance*):

$$d(w, v) \overset{\text{def}}{=} |w \triangle v|$$

where the $|\cdot|$ operator is set-cardinality. The distance of a formula ψ and an interpretation v to be:

$$d(\psi, v) \overset{\text{def}}{=} \min\{d(w, v) \mid w \in \mathrm{mod}(\psi)\}$$

Using this notion of distance, a faithful assignment can be defined as

$$w \leq_\psi v \quad \text{iff} \quad d(\psi, w) \leq d(\psi, v)$$

The induced ordering is clearly total, reflexive and transitive and thus, the operator is a revision by the representation theorem for revisions.

The idea behind the BDD formulation of Dalal's operator comes from the construction of a circuit that, when given four interpretations w, v, x and y in the form of binary vectors, decides whether $d(w, x) \leq d(v, y)$ by 0 the appropriate boolean value. Thus, in order to compare $|w\triangle x|$ and $|v\triangle y|$ we need a way to *count* how many propositional variables are true in each set-difference and compare those

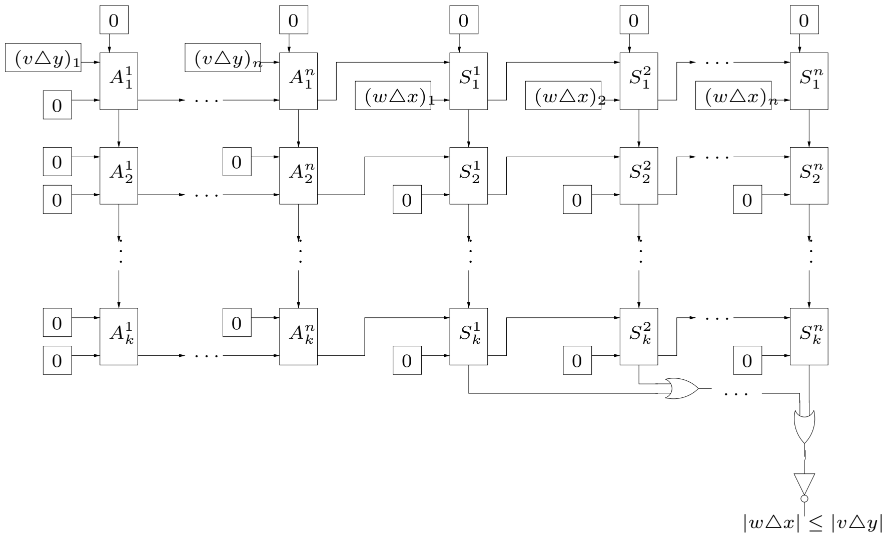

Figure 4. Circuit to decide $|w\triangle x| \leq |v\triangle y|$.

counts. These counts will be binary numbers representing how many 1s occur in those differences. The maximum number of differences possible is obviously n, thus these binary numbers need only have $k = \lceil \log_2 n \rceil$ bits.

A construction made of n k-bit adders in sequence can be used to do the counting of bits set to 1 in $v\triangle y$ (see left-half of figure 4). Blocks labelled A_j^i are *full-adders*. These blocks are simple binary circuits that, given two input bits a, b and a carry bit c, they calculate the sum o and the produced carry bit c':

$$\begin{aligned} o &= \neg(\neg(a \leftrightarrow b) \leftrightarrow c) \\ c' &= (a \wedge b) \vee (c \wedge \neg(a \leftrightarrow b)) \end{aligned}$$

Each column in the first-half of figure 4 forms a k-bit adder. By connecting zeros to all bits of the first argument except the first one, where $(v\triangle y)_i$ is connected, we ensure that the i-th bit of the difference is added to the second argument, which holds the results of the counting so far.

In order to compare the count we get from the left-half of figure, we use a structure made from subtracters S_j^i in order to *count down* the 1s in $w\triangle x$, seen in the right-half of figure 4. Similar to the full-adder, the unit S_j^i is a boolean circuit that given inputs a, b and an input carry c calculates the difference o and the produced carry bit c':

$$o = \neg(\neg(a \leftrightarrow b) \leftrightarrow c)$$

$$c' = (b \wedge c) \vee (\neg a \wedge \neg(b \leftrightarrow c))$$

If, while counting down, the subtraction produces a carry bit then we know that $|w \triangle x| > |v \triangle y|$. Thus, we preserve the existence of a carry bit by taking the disjunction of all carry bits produced by the subtraction stages and by inverting that value the circuit decides $|w \triangle v| \leq |v \triangle y|$.

In order to apply the theorem in section 2.4, we define an ordering over the blocks of the circuit:

$$\begin{array}{lllll}
(v\triangle y)_1, & A_1^1, & \ldots, & A_k^1, & \\
 & & \vdots & & \\
(v\triangle y)_n, & A_1^n, & \ldots, & A_k^n, & \\
(w\triangle x)_1, & S_1^1, & \ldots, & S_k^1, & \vee, \\
 & & \vdots & & \\
(w\triangle x)_n, & S_1^n, & \ldots, & S_k^n, & \vee, \ \neg
\end{array}$$

It is easy to see that on each block, the forward cross section is at most $k + C$ where C is a constant and that the reverse cross section is always zero. Thus, the forward width of the circuit is $k + C$ and the bound given by the theorem is $4n2^{k+C} = O(n^2)$, because $k = \lceil \log_2 n \rceil$.[2] Therefore there exists a BDD of size $O(n^2)$ that represents $|w \triangle x| \leq |v \triangle y|$.

In order to express Dalal's revision as an operation on BDDs, we have to construct the BDD operation representing the faithful assignment. By its definition we have:

$$\begin{aligned}
w \leq_\psi v \ &\text{iff} \ d(\psi, w) \leq d(\psi, v) \\
&\text{iff} \ \exists x. \, (x \in \text{mod}(\psi) \wedge \forall y. \, (y \in \text{mod}(\psi) \rightarrow |w \triangle x| \leq |v \triangle y|))
\end{aligned}$$

Since Dalal's revision is known to satisfy R1-R6 (see [10]) the above faithful assignment determines uniquely the revision operator. Assuming that the ordering is represented by a BDD B_R with variables $\mathbf{p}, \mathbf{p}', \mathbf{p}'', \mathbf{p}'''$ corresponding to w, v, x, y respectively, then the BDD algorithm for the faithful assignment is:

$$\exists \mathbf{p}'''. \, (B_\psi[\mathbf{p}'''/\mathbf{p}] \wedge \forall \mathbf{p}''''. \, (B_\psi[\mathbf{p}''''/\mathbf{p}] \rightarrow B_R))$$

Operation	Time Complexity	Result Size
$B_\psi[\mathbf{p}''''/\mathbf{p}], B_\psi[\mathbf{p}'''/\mathbf{p}]$	$O(\lvert B_\psi \rvert)$	$O(\lvert B_\psi \rvert)$
$\forall \mathbf{p}''''. \, (\cdot \rightarrow \cdot)$	$O(\lvert B_\psi \rvert \cdot n^2 \cdot 2^{8n})$	$O(2^{3n})$
$\exists \mathbf{p}'''. \, (\cdot \wedge \cdot)$	$O(\lvert B_\psi \rvert \cdot 2^{3n} \cdot 2^{6n})$	$O(2^{2n})$

[2] Several optimisations can be made on the circuit appearing in figure 4, by replacing blocks with known output with appropriate constants. The forward width of the circuit, however, does not change.

Therefore, an upper bound for the worst-case time complexity of the BDD algorithm for the faithful assignment is $O(|B_\psi|\cdot 2^{9n})$. Thus, in view of the result of section 4.1, the derived upper bound for the worst-case time complexity of the revision is $O(\max\{|B_\psi|\cdot 2^{9n},\ |B_\mu|\cdot 2^{2n}\cdot 2^{4n}\}) = O(|B_\psi|\cdot 2^{9n})$.

4.5. Update defined by faithful assignment

Theorem 3 tells us how to compute $\phi\diamond\psi$, given a faithful assignment for updates. Instead of representing the faithful assignment as an operation `fa` from BDDs representing ϕ to a BDD representing $\leq_\phi$, as we did for revisions, we represent the faithful assignment as a BDD over $\mathbf{p}, \mathbf{p}', \mathbf{p}''$. This possibility is available to us in the case of updates, but not in the case of revisions, because the faithful assignment is indexed by a model in updates and by a set of models in revisions.

$B_\leq$ is a BDD over $\mathbf{p}, \mathbf{p}', \mathbf{p}''$. Given values for $\mathbf{p}''$, it would become a BDD over $\mathbf{p}, \mathbf{p}'$, representing a simple (partial) ordering. For example, the expression

$$\exists \mathbf{p}''.\,(B_\phi[\mathbf{p}''/\mathbf{p}] \wedge B_\leq)$$

returns the relation

$$\bigcup_{w\in \mathrm{mod}(\phi)} \leq_w .$$

That is not what we want, however. Theorem 3 asks us to compute

$$\mathrm{mod}(\psi\diamond\mu) = \bigcup_{w\in \mathrm{mod}(\psi)} \min_{\leq_w}(\mathrm{mod}(\mu))$$

$B_\leq$ can be fed its inputs in any order, and it is convenient to manipulate its $\mathbf{p}, \mathbf{p}'$ parameters first. Using the definition of `min` is section 2.3 *for partial orderings*, we may calculate the BDD

$$\mathtt{min}(B_\leq, B_\mu)$$

which, given $\mathbf{p}''$ representing w, computes $\min_{\leq_w}(\mathrm{mod}(\psi))$. This BDD is still parameterised by $\mathbf{p}''$, since we need to take the union over all $w \in \mathrm{mod}(\psi)$. The final answer for the BDD representing $\psi\diamond\mu$ in terms of the BDDs $B_\leq$, B_ψ and B_μ is therefore

$$B_\mu \wedge \exists \mathbf{p}''.\,(B_\psi[\mathbf{p}''/\mathbf{p}] \wedge \mathtt{min}(B_\leq, B_\mu)).$$

which when expanded gives (where $B_{\not<} = \neg(B_\leq \wedge \neg B_\leq[\mathbf{p}'/\mathbf{p}, \mathbf{p}/\mathbf{p}'])$)

$$B_\mu \wedge \exists \mathbf{p}''.\,(B_\psi[\mathbf{p}''/\mathbf{p}] \wedge \forall \mathbf{p}'.(B_\mu[\mathbf{p}'/\mathbf{p}] \rightarrow B_{\not<}))$$

An upper bound for the complexity of the double replacement is $O(|B_\leq|\cdot n2^{6n})$. The actual complexity is probably much lower, but a lower bound would not change the overall complexity of the algorithm, computed below.

Operation	Time Complexity	Result Size
$B_{\leq}[\mathbf{p}'/\mathbf{p}, \mathbf{p}/\mathbf{p}']$	$O(\lvert B_{\leq}\rvert \cdot n2^{6n})$	$O(2^{3n})$
$B_{\not<}$	$O(\lvert B_{\leq}\rvert \cdot 2^{3n})$	$O(2^{3n})$
$\forall\mathbf{p}'.\,(\cdot \rightarrow \cdot)$	$O(\lvert B_{\mu}\rvert \cdot 2^{3n} \cdot 2^{6n})$	$O(2^{2n})$
$\exists\mathbf{p}''.\,(\cdot \wedge \cdot)$	$O(\lvert B_{\psi}\rvert \cdot 2^{2n} \cdot 2^{4n})$	$O(2^{n})$

Thus, an upper bound for the worst case complexity of update is $O(|B_\mu| \cdot 2^{9n})$.

4.6. Winslett

Winslett introduced an update operator in [18]. The ordering used in this operator is defined using the set-inclusion of symmetric set-differences

$$a \leq_w b \quad \text{iff} \quad w \triangle a \subseteq w \triangle b$$

which is, clearly, a partial order and the mapping is a faithful assignment. As noted in [10], Winslett's operator coincides with Borgida's when ψ and μ are inconsistent. In other words, in Winslett's update the second step of the algorithm for Borgida's revision is always used, so our results in section 4.2 carry over here unchanged.

5. Fault Diagnosis

In this section we present a formulation of fault diagnosis as a special kind of belief revision, along with experimental results gathered from an implementation of that algorithm. Our goal is not to formulate a fully-fledged theory for fault diagnosis, nor to prove that the best method for diagnosis is by belief revision. What we aim at is to demonstrate the BDD algorithms we have presented, in a medium-sized example. To that end, we formulate a method for fault diagnosis that works in a well-studied class of systems, combinational boolean circuits, and investigate its complexity in practice.

5.1. Fault Diagnosis of Boolean Combinational Circuits

Physical systems can develop faults that make them deviate from their specifications. Given a description of a physical system and an observation of the system (usually, an input-output observation) that is inconsistent with the specification, the problem of fault diagnosis is to deduce which components of the system are faulty.

To use revision for fault diagnosis [14, 18, 6], we let ψ_1 be a formula describing the circuit's operation on the assumption that the components are not faulty, and ψ_2 be the assertion that the components are not faulty. Let μ be the observation made of the system. We perform the revision $(\psi_1 \wedge \psi_2) \circ \mu$ subject to the *integrity constraint* ψ_1. The result is a set of possible fault cases.

Since we are interested in implementations of belief change in a finite propositional language, a natural application is fault diagnosis of combinational boolean circuits. Such a circuit consists of a finite number g of unary or binary gates.[3] We define n_I propositional variables I_i $(1 \leq i \leq n_I)$ corresponding to the primary inputs of the circuit (at most $2g$). For each gate i, we define a propositional variable N_i, its *normality predicate*, as well as O_i, its *output value*[4]. The input(s) of each gate will either be primary input(s) or output(s) of other gates. The circuit has also n_{PO} primary outputs, denoted as PO_i $(1 \leq i \leq n_{PO})$, which form a subset of the output values O_i (at most g if no repetitions of results are allowed). Output values of gates not belonging to the set of primary outputs are called intermediate results.

Therefore, for a circuit of g gates we define $n_I + 2g$ (at most $4g$) propositional variables. However, not each valuation of those $n_I + 2g$ variables is a possible state the circuit can be found in; if, in some valuation, the normality predicate of a gate is true then its behaviour is uniquely determined and thus, its output can only assume one value out of the two possible. The set of interpretations allowed under the specification of the circuit is the set of valuations that satisfy its integrity constraints

$$\mathrm{IC} \overset{\mathrm{def}}{=} \bigwedge_{i=1}^{g} N_i \to (F_i \leftrightarrow O_i)$$

where F_i is a boolean expression defining the expected output in terms of the inputs of gate i. The integrity constraints for a circuit that computes $\neg(I_1 \wedge I_2)$, for example, are

$$\mathrm{IC} = (N_{\mathrm{AND}} \to (I_1 \wedge I_2 \leftrightarrow O_{\mathrm{AND}})) \wedge (N_{\mathrm{NOT}} \to (\neg O_{\mathrm{AND}} \leftrightarrow O_{\mathrm{NOT}}))$$

The initial belief will be the conjunction of the integrity constraints and of the belief that *all gates are not faulty*:

$$\mathrm{IB} \overset{\mathrm{def}}{=} \mathrm{IC} \wedge \left(\bigwedge_{i=1}^{g} N_i \right)$$

[3] The presented method can be easily generalised for gates of any (constant) arity and of any (constant) number of outputs.

[4] Since the gate may be faulty, its output value need not be uniquely determined by its inputs. Thus we do need a separate propositional variable for its output value.

An observation is a description of observed primary input and primary output values

$$\mathrm{OBS} \overset{\text{def}}{=} \bigwedge_{i=1}^{n_I} (I_i \text{ or } \neg I_i) \wedge \bigwedge_{j=1}^{n_{PO}} (PO_j \text{ or } \neg PO_j)$$

Our goal is to define a revision operator that given an initial belief and an observation of the above forms, returns an epistemic state describing which gates if taken as faulty, explain the given observation. Of course, the returned formula need not indicate only one combination of faulty gates; there could be several ways in which a faulty circuit can produce a given output.

We define the revision operator using a suitable notion of minimality. Intuitively, we want to select all those interpretations that are models of the observed behaviour, while making the *smallest change* to the *persistent* information about the circuit, i.e., the normality predicates. Thus, a suitable notion of minimality is the set-inclusion of differences, but restricted on normality predicates. We do not use a variant of Dalal's operator because that would imply that we are only interested in the minimum *number* of faults necessary to explain the observation.

We choose a variant of Borgida's operator to model this notion of closeness. This variant is identical to Borgida's version, except for the ordering on interpretations. If x, y, z are interpretations, the ordering is defined as:

$$x \leq_z y \ \text{ iff } \left(\bigwedge_{i=1}^{n_I} I_i(x) \leftrightarrow I_i(y)\right) \wedge \left(\bigwedge_{j=1}^{n_{PO}} PO_j(x) \leftrightarrow PO_j(y)\right) \wedge$$
$$\left(\bigwedge_{k=1}^{g} \neg(N_k(z) \leftrightarrow N_k(x)) \rightarrow \neg(N_k(z) \leftrightarrow N_k(y))\right)$$

where $N_i(x)$ denotes the value of N_i at the interpretation x, and similarly for other propositional variables. Thus, for two interpretations to be comparable, they should imply the same input-output behaviour, hence the first two conjuncts of the above formula. Note that intermediate results do not appear in the definition of the ordering, as they are not observable. The third conjunct formalises our notion of minimal changes; we are interested in the minimal set of gates (with respect to set-inclusion) that, when faulty, concord with the observation.

Under this revision operator, protection of integrity constraints is achieved by revising our initial belief not just with the observation, but with the conjunction $\mathrm{IC} \wedge \mathrm{OBS}$.

The result of the revision will include information about the particular observation we revised with, in view of the axiom R2. In particular, the values of primary inputs and outputs in the observation will be implied by the resulting epistemic state. Since in fault diagnosis we are only interested in information about the normality predicates of the circuit, we need to eliminate from the resulting belief all knowledge about propositional variables other than normality predicates. We use (boolean) existential quantification to eliminate all propositional variables that carry irrelevant information from the result of the revision. This operation, called *elimination*, is described in [9].

5.2. BDD Formulation

The ordering and the negation of its strict counterpart are easily constructible as circuits similar to the one presented in section 4.2. Thus, by the theorem in section 2.4, the BDD $B_{\not<}$ representing $\not<$ (where $\leq$ is defined by the formula above) is of size $O(n_I + 2g) = O(g)$.

The BDD B_{IC} is, of course, dependent on the specific circuit in question. Therefore we cannot give a bound on its size. However, by using the variable ordering $I_1, \ldots, I_{n_I}, N_1, O_1, \ldots, N_g, O_g$ we ensure an empirically compact representation of B_{IC}.

The BDD for the conjunction of all normality predicates N_i can be easily shown to have a size of $O(g)$ irrespective of the variable ordering used. Thus the BDD for the initial belief B_{IB} will be of size $O(|B_{\mathrm{IC}}| \cdot g)$, by the `apply` algorithm. Similarly, the BDD B_{OBS} for the observation will have a size of $O(g)$ and the conjunction with the integrity constraints $B_{\mathrm{IC}} \wedge B_{\mathrm{OBS}}$ will be of size $O(|B_{\mathrm{IC}}| \cdot g)$.

Therefore, the models of the revision are

$$\begin{aligned}\mathrm{mod}(\mathrm{IB} \circ (\mathrm{IC} \wedge \mathrm{OBS})) = \{w \mid w \in \mathrm{mod}(\mathrm{IC} \wedge \mathrm{OBS}) \wedge \\ \exists v.\, (v \in \mathrm{mod}(\mathrm{IB}) \wedge \forall z.\, (z \in \mathrm{mod}(\mathrm{IC} \wedge \mathrm{OBS}) \rightarrow z \not<_v w))\}\end{aligned}$$

and the models of the diagnosis are obtained by quantifying away all variables except normality predicates:

$$\mathrm{mod}(\mathrm{DIAG}) = \exists I_1 \ldots \exists I_{n_I} \exists O_1 \ldots \exists O_g.\, \mathrm{mod}(\mathrm{IB} \circ (\mathrm{IC} \wedge \mathrm{OBS}))$$

The BDD algorithm is:

$$\begin{aligned}\exists \mathbf{p}^{\langle I_1,\ldots,I_{n_I},O_1,\ldots,O_g\rangle}.\, ((B_{\mathrm{IC}} \wedge B_{\mathrm{OBS}}) \wedge \\ \exists \mathbf{p}'.\, (B_{\mathrm{IB}}[\mathbf{p}'/\mathbf{p}] \wedge \forall \mathbf{p}''.\, ((B_{\mathrm{IC}} \wedge B_{\mathrm{OBS}})[\mathbf{p}''/\mathbf{p}] \rightarrow B_{\not<})))\end{aligned}$$

An upper bound for the worst-case time complexity of which is $O(|B_{\mathrm{IC}}| \cdot g^2 2^{24g})$.

5.3. Implementation and Experimental Results

As mentioned earlier, belief change operations are known to be very expensive in the worst case. Since the complexities reported up to this point concern only the worst-case, and taking into account the fact that the known bounds on BDD operations such as quantification are still not tight, we proceeded with a medium-scale implementation of the algorithm presented in the previous section. Results regarding the complexity of that algorithm were gathered by trying diagnosis of random observations. Those results along with details about the implementation are presented below.

The implementation uses the BuDDy package for the manipulation and construction of BDDs [12]. This package, as most of the packages available, offers heuristics for automatically re-ordering the variables of BDDs in order to attain lower space complexities. This capability is very important in an application for fault diagnosis on arbitrary circuits, since the BDD for the integrity constraints is of an unpredictable size and can benefit from automatic re-ordering. However, having chosen a specific circuit to perform our experiments on, an n-bit adder, we did not make use of this feature.

We have assembled a small collection of fairly general tools that can be used for diagnosis of any combinational boolean circuit. The chosen circuit we have tested, the n-bit adder, is a circuit that leads to low complexities of the BDDs involved in diagnosis. However, if we had selected a circuit that knowingly lead to exponential complexities, then the average case for diagnosis (and indeed, *any* case) would be provably exponential. The BDD for the integrity constraints B_{IC}, has linear complexity for the n-bit adder. Using a standard design for adders, each bit of the adder amounts to 5 gates, thus an upper bound for the worst-case complexity of the fault-diagnosis algorithm on an n-bit adder is $O(n^3 2^{120n})$, in view of the result in the previous section.

We attempted two kinds of tests. In the first one, n-bit adders of successively larger n were generated, and each one was diagnosed with a set of uniformly distributed random observations. The number of possible observations for n bits is 2^{3n}. A constant percentage of those 2^{3n} observations were sampled and fed to the diagnosis algorithm. The space complexity of each diagnosis was recorded and an average complexity for each n was produced.

The above-mentioned algorithm, when fed with an observation that is *consistent* with the integrity constraints, does not revise the normality predicates. In this special case, it can be easily proved that there is just one model of $\mathrm{IC} \wedge \mathrm{OBS}$, and that the algorithm exhibits a complexity much lower than in the worst-case. However, the number of consistent observations for an n-bit adder is 2^{2n}, thus the ratio of consistent to all observations is $1/2^n$. Therefore, as n increases, the probability of us sampling a consistent observation decreases exponentially, thus increasing the measured average complexity.

Unfortunately, sampling a fixed percentage of an exponentially sized population leads in exponential time taken for the tests. Indeed, at 7 bits, diagnosing 10% of the total number of observations possible amounts to running the diagnosis algorithm 209716 times, and for 8 bits this number is multiplied by 8. Due to the excessive time taken for the fixed-percentage tests, we could only run them for up to 7 bits. The results for this test are shown below (space complexity is measured in BDD nodes produced):

Bits	Average Space Complexity	Number of Samples
1	234	1
2	886	7
3	1906	52
4	3229	410
5	4861	3277
6	6804	26215
7	9052	209716

In order to get an idea of the complexities concerned in larger circuits, we tried a second test by running our algorithm on 1000 samples for each bit-size. Admittedly, this approach reduces exponentially the accuracy of our averages in the number of bits. However, by running this test multiple times we have empirically verified that the variance of the results is not significant.[5] The average space complexity (in BDD nodes produced) and the average time complexity (in ms) are shown in figures 5 and 5 respectively.

As with any empirical investigation, these results cannot be taken as conclusive evidence of tractability or intractability. However, we did use a nonlinear least squares method (Marquardt-Levenberg algorithm) to fit a number of classes of functions to the above curves. To our best knowledge, the best fit was a quadratic function, being significantly better that exponential and sub-exponential non-polynomial ones. In addition, in the case of space complexity, the resulting quadratic function from fitting the first or last 15 data points predicts reasonably well the remaining 45.

[5] In addition, the ratio of standard deviation to average never exceeded 5% in the case of space complexity and 10% in the case of time complexity.

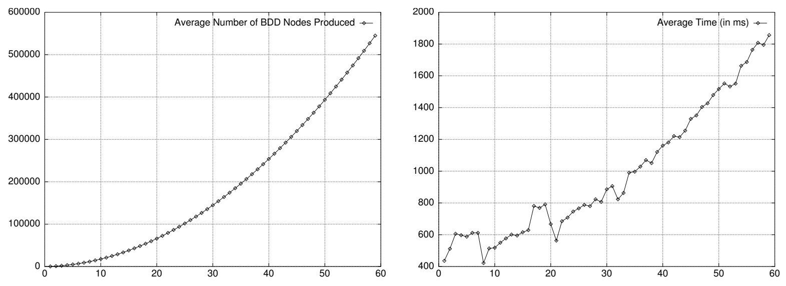

Figure 5. Average number of BDD nodes produced and average time (in ms) spent per diagnosis in the number of bits.

What has been demonstrated, we believe, is the practicality of our approach, contrary to the order of magnitude of the upper bounds calculated in the previous sections; for fault diagnosis, with a moderately-sized computer system one can perform single runs of our algorithm on reasonably large circuits. In particular, we have successfully run our program several times on adders of 1000 bits, with each run taking 20 minutes on average. Note that in the model presented above, a 1000-bit adder leads to a state-space of 2^{12000} interpretations.

6. Conclusions and Further Work

We presented a formulation of a variety of belief change operators on a finite propositional language as algorithms on BDDs. Moreover, upper bounds for their worst-case complexities were calculated. These bounds do not provide conclusive evidence for the efficiency of the proposed methods and can only do so when tight bounds of the worst-case complexity of BDD algorithms (especially `andExists`) are proved.

In the context of verification, the use of BDDs has greatly extended the size of systems that can be practically verified, in spite of the worst-case complexity of model-checking (the problem of model-checking a CTL formula is PSPACE-complete). This fact warranted us to investigate empirically the average-case complexities of the presented algorithms in a fault-diagnosis case study. We found them to be much better than their expected worst case.

An obvious impediment to the applicability of the methods we have presented is the limited expressive power of propositional logic. As already mentioned, BDD algorithms are routinely used in verification, in which the underlying language is CTL, a temporal modal logic. In general, many systems of modal logic give rise to possible applications of belief change (concept revision in description logics, feature integration in temporal modal logics as belief update, knowledge revision in epistemic logics).

Moreover, the expressivity of several modal logics seems to be a satisfactory compromise between propositional and first-order logic, while retaining several important characteristics such as decidability. Thus, a natural continuation of our current work is the study of belief change with a modal logic as base language and in particular the connections of belief change with the semantics of the underlying logic.

References

1. Andersen, H. R.: 1998, 'An Introduction to Binary Decision Diagrams'. `http://www.it.dtu.dk/~hra`. Department of Information Technology, Technical University of Denmark, Lecture Notes.
2. Borgida, A.: 1984, 'Intelligent Handling of Exceptions in Information Systems'. In: *Workshop on Expert Database Systems, University of South Carolina, 1984*, Vol. 2.
3. Bryant, R. E.: 1986, 'Graph-Based Algorithms for Boolean Function Manipulation'. *IEEE Transactions on Computers* **C-35**(8), 677–691.
4. Bryant, R. E.: 1992, 'Symbolic Boolean Manipulation with Ordered Binary-Decision Diagrams'. *ACM Computing Surveys* **24**(3), 293–318.
5. Burch, J. R., J. M. Clarke, K. L. McMillan, D. L. Dill, and J. Hwang: 1990, 'Symbolic model checking: 10^{20} states and beyond'. In: *IEEE Symposium on Logic in Computer Science.*
6. Dalal, M.: 1988, 'Investigations Into a Theory of Knowledge Base Revision: Preliminary Report'. In: *Proceedings of the Seventh National Conference on Artificial Intelligence*, Vol. 2. St. Paul, Minnesota, pp. 475–479.
7. Gärdenfors, P.: 1988, *Knowledge in Flux: Modeling the Dynamics of Epistemic States.* Cambridge, MA: MIT Press, Bradford Books.
8. Huth, M. R. and M. D. Ryan: 2000, *Logic in Computer Science: Modelling and Reasoning about Systems.* Cambridge University Press.
9. Katsuno, H. and A. O. Mendelzon: 1989, 'A Unified View of Propositional Knowledge Base Updates'. In: N. S. Sridharan (ed.): *Proceedings of the 11th International Joint Conference on Artificial Intelligence.* Detroit, MI, USA, pp. 1413–1419.
10. Katsuno, H. and A. O. Mendelzon: 1991, 'Propositional Knowledge Base Revision and Minimal Change'. *Artificial Intelligence* **52**(3), 263–294.
11. Katsuno, H. and A. O. Mendelzon: 1992, 'On the Difference between Updating a Knowledge Base annd Revising it'. In: P. Gärdenfors (ed.): *Belief Revision.* Cambridge University Press, pp. 183–203.
12. Lind-Nielsen, J., 'BuDDy: Binary Decision Diagram Package Release 1.8'. `http://www.it.dtu.dk/research/buddy`.
13. McMillan, K. L.: 1993, *Symbolic Model Checking.* Kluwer Academic Publishers.
14. Reiter, R.: 1987, 'A theory of diagnosis from first principles'. *Artificial Intelligence* **32**, 57–95. Reprinted in in *Readings in Nonmonotonic Reasoning*, M. L. Ginsberg (ed.), Morgan Kaufman, San Francisco, CA. 1987, pp. 352–371.
15. Satoh, K.: 1988, 'Nonmonotonic Reasoning by Minimal Belief Revision'. In: I. for New Generation Computer Technology (ICOT) (ed.): *Proceedings of the*

International Conference on Fifth Generation Computer Systems. Volume 2. Berlin, FRG, pp. 455–462.

16. Somenzi, F., 'CUDD: CU Decision Diagram Package Release 2.3.0'. `http://vlsi.colorado.edu/~fabio/CUDD/cuddIntro.html`.
17. Williams, M.-A.: 1997, 'Anytime Belief Revision'. In: *Fifteenth International Joint Conference on Artifical Intelligence*. pp. 74–81.
18. Winslett, M.: 1988, 'Reasoning About Action Using a Possible Models Approach'. In: *Proceedings of the Seventh National Conference on Artificial Intelligence*. pp. 89–93.