Implementing Semantic Merging Operators using Binary Decision Diagrams

Nikos Gorogiannis and Anthony Hunter

13th July 2012

Abstract

There is a well-recognised need in diverse applications for reasoning with multiple, potentially inconsistent sources of information. One approach is to represent each source of information by a set of formulae and then use a merging operator to produce a set of formulae as output. A valuable range of model-based operators have been proposed that conform to interesting and intuitive properties. However, the implementation of such operators has remained unaddressed, partly due to the considerable computational complexity of the proposals. To address this, we propose a methodology for implementing model-based merging operators using the notion of dilation and a type of data structure called a Binary Decision Diagram. We apply this method by implementing four merging operators from the literature and experimentally evaluating their average-case performance. The results indicate that while the complexity is indeed significant, problems of modest size can be treated using commodity hardware and short computation times.

Keywords: knowledge merging, knowledge representation, automated reasoning

1 Introduction

The increasing availability of distributed sources of information creates the potential for applications that intelligently combine these sources. However, the sources may conflict, and therefore, the question arises of how we can combine them usefully whilst avoiding the problems arising from inconsistencies in the pooled information. *Logic-based knowledge merging* aims to address these issues by proposing ways of combining multiple sources of potentially conflicting information. For this, a range of *merging operators* has been proposed. These are functions that take a number of knowledge bases each representing an information source, and merge them into a single knowledge base that is consistent and that preserves, in some sense, as much information content as possible. Several proposals can be found in the literature, [6, 7, 25, 26, 22, 24, 21]. These proposals can generally be divided into two categories: the *semantic* (or model-based)

operators, which select models that are somehow closest to the knowledge bases to be merged, and the *syntactic* (or formula-based) operators, which work by selecting formulae from the knowledge bases to be merged.

Despite the increasing interest in knowledge merging, there is a lack of available implementations. This can partly be attributed to the considerable computational complexity of the decision problems associated with knowledge merging, generally situated at the second level of the polynomial hierarchy or above [21]. However, many decision problems in computer science that are generally intractable have been found to be less intractable in their "average"-case complexity with the help of heuristics or succinct data structures (e.g., the problems encountered as part of formal verification, as well as propositional satisfiability).

The aim of this paper is, therefore, to propose and experimentally evaluate algorithms that implement some of the semantic merging operators from the literature. To this end, we employ:

- Four model-based operators defined by Konieczny et al. [22, 23, 24].
- The concept of *dilation* of a set of models, an application of mathematical morphology in the field of logic [8].
- Data structures called *Binary Decision Diagrams* (BDDs), that originate in the field of formal verification, and allow compact representation and manipulation of propositional logic formulae [9].

BDDs have been used for implementing algorithms for several problems in AI including deduction, abduction [28], belief revision [31, 15] and as a general language for compiling knowledge bases [19, 11]. This increasing usage of BDDs in AI indicates that implementing knowledge merging with BDDs is a promising research direction.

The contribution of this paper is two-fold. First, we propose algorithms for implementing four important model-based merging operators from the literature, and prove their correctness. Second, we implement and experimentally evaluate these algorithms using randomly generated knowledge bases; we subsequently analyse the results and attempt to pinpoint the sources of computational complexity.

The rest of the paper is organised as follows. The knowledge merging operators we implement are presented in Section 2.1. The concept of dilation and its relation to knowledge merging is reviewed in Section 2.2. BDDs and their associated algorithms are introduced in Section 2.3. In Section 3, we present algorithms implementing the semantic knowledge merging operators reviewed in Section 2.1. Experimental data from the execution of these algorithms are presented and analysed in Section 4. Finally, we discuss the issues we encountered and suggest possibilities for further research in Section 5.

2 Background

We use a (finite) propositional language $\mathcal{L}$ with $\top$, $\bot$ as constants and the usual connectives $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$. The set of models of the language is denoted by $\mathcal{M}$ and the set of models of a formula $\phi \in \mathcal{L}$ is denoted by $\text{mod}(\phi)$. If $u \in \mathcal{M}$, then $u \models \phi$ means u satisfies ϕ. The number of propositional atoms is denoted by p and we use α_i with $1 \leq i \leq p$ for the corresponding atom. A literal is an atom, possibly negated. Given a model u we will often refer to the value of α_i in u, as u_i. As usual, $\phi \vdash \psi$ means that the formula ϕ entails the formula ψ.

A vector (or sequence) is a finite ordered list denoted by $\langle a, b, \ldots \rangle$, and V_i denotes the i-th element of V. The algorithms presented use one-dimensional arrays, which are essentially vectors, and two-dimensional arrays, which are essentially tables. If A is a one-dimensional array, A_i is the i-th element and if A is a two-dimensional array, then $A_{i,j}$ is the element at the intersection of the i-th row and j-th column.

We will denote the cardinality of a set S as $|S|$. In the following we will often treat a model u as a vector of the truth-values of the atoms on u, $\langle u_1, \ldots, u_p \rangle$. The lexicographic ordering of vectors of numbers is denoted by $\leq_L$ and is defined as $A <_L B$ iff there exists $1 \leq j$ such that for all $1 \leq i < j$, $A_i = B_i$ and $A_j < B_j$. So, for example, $\langle 0, 0, 2 \rangle <_L \langle 0, 1, 2 \rangle$. Then, $A \leq_L B$ iff $A <_L B$ or $A = B$.

A *composition*[1] of a number n into k parts, where $0 \leq n$ and $1 \leq k$ is a vector $\langle a_1, \ldots, a_k \rangle$ such that $0 \leq a_i$ and $\sum_{i=1}^{k} a_i = n$. The set of all compositions of n into k parts is denoted by $\mathit{compositions}(n, k)$. A permutation of a vector V is a vector of length $|V|$ which contains each (appearance of an) element of V exactly once. We denote the set of all such permutations as $\mathit{permutations}(V)$.

2.1 Knowledge merging

A profile $E = \langle \phi_1, \ldots, \phi_k \rangle$ is a sequence of k knowledge bases, represented by (consistent) formulae. A merging operator (defined below) requires a profile along with a formula μ expressing the integrity constraints. In the following we will always use E to refer to the profile under consideration, μ for the integrity constraints, k for the number of formulae in the profile and ϕ_i with $1 \leq i \leq k$ for the formulae of E.

A knowledge merging operator $\triangle_\mu$ is a function from profiles to formulae. The intuition is that, if the formulae in the profile are inconsistent when taken together, then we would like to produce a new formula that is consistent with the integrity constraints and somehow summarises or merges the knowledge contained in the formulae of the profile. There is, of course, no single way of producing this result, therefore there exists a range of merging operator definitions in the literature, each predicated on a different set of assumptions. In a similar manner to belief revision operators [2], merging operators can be characterised according to a set of rationality postulates [22, 24, 21].

A number of merging operators are defined in a model-theoretic way, usually

[1] Also known as weak composition.

employing a notion of distance. We summarise the required definitions here. A distance function is a function from a pair of models to a totally-ordered set. The most commonly used distance is the number of atoms on which the two models disagree. This is called the Hamming, or Dalal, distance [18, 10].

$$d(v, u) = |\{\, i \mid v_i \neq u_i \text{ and } 1 \leq i \leq p \,\}|$$

The Hamming distance is the only distance between models we will examine. Using this distance as a basis, the distance between a model and a formula can then be defined by minimising over the models of the formula.

$$d(\phi, u) = \min \{\, d(v, u) \mid v \in \mathrm{mod}(\phi) \,\}$$

Various notions of *aggregate* distance between a model u and a profile E, $d_x(E, u)$, can then be defined. Using such a notion, a merging operator $\triangle^x_\mu$ is then defined through a process of minimisation:

$$\mathrm{mod}(\triangle^x_\mu(E)) = \{\, u \in \mathrm{mod}(\mu) \mid d_x(E, u) \text{ is minimal} \,\}.$$

We will use the symbol d for all the types of distance functions: model-to-model, formula-to-model and profile-to-model. The arguments will make clear which one is intended. We now look at four different notions of profile-to-model distances defined by Konieczny et al., namely d_{Max}, d_Σ, d_{Gmax} and d_{Gmin}, that give rise to four merging operators, $\triangle^{\mathrm{Max}}_\mu$, $\triangle^\Sigma_\mu$, $\triangle^{\mathrm{Gmax}}_\mu$ and $\triangle^{\mathrm{Gmin}}_\mu$ respectively.

To illustrate the behaviour of the operators reviewed here, we present a running example. Suppose that we want to merge the following three formulae:

$$\begin{aligned}
\phi_1 &= (a \wedge \neg b \wedge c) \vee (a \wedge b \wedge \neg c \wedge \neg d) \\
\phi_2 &= (\neg a \wedge \neg c \wedge d) \vee (\neg a \wedge \neg b \wedge c \wedge \neg d) \vee (a \wedge c \wedge \neg d) \\
\phi_3 &= (\neg b \wedge \neg c \wedge d) \vee (\neg a \wedge b \wedge c \wedge \neg d) \vee (a \wedge b \wedge c \wedge d)
\end{aligned}$$

We want to merge these formulae using the following integrity constraints:

$$\mu = (\neg a \wedge \neg b \wedge \neg c) \vee (\neg a \wedge c \wedge \neg d) \vee (a \wedge \neg b \wedge \neg c) \vee (a \wedge b \wedge c)$$

To encode these formulae as a profile E, we set $p = 4$, $k = 3$, $\alpha_1 = a$, $\alpha_2 = b$, $\alpha_3 = c$ and $\alpha_4 = d$. Table 1 presents the models of the language. Rows with a grey background correspond to models of μ. The rest of the table is discussed in the following sections.

2.1.1 The operator $\triangle^{\mathrm{Max}}_\mu$

In [22], Konieczny and Pino Pérez define an operator originating in [29]. This operator uses the maximum distance from each profile formula as the aggregate distance.

$$d_{\mathrm{Max}}(E, u) = \max \{\, d(\phi_i, u) \mid 1 \leq i \leq k \,\}$$

Then, the operator $\triangle^{\mathrm{Max}}_\mu$ can be defined as follows.

$$\mathrm{mod}(\triangle^{\mathrm{Max}}_\mu(E)) = \{\, u \in \mathrm{mod}(\mu) \mid d_{\mathrm{Max}}(E, u) \text{ is minimal} \,\}$$

model u	$d(\phi_1, u)$	$d(\phi_2, u)$	$d(\phi_3, u)$	d_{Max}	d_{Σ}	d_{Gmax}	d_{Gmin}
(0,0,0,0)	2	1	1	2	4	$\langle 2,1,1\rangle$	$\langle 1,1,2\rangle$
(0,0,0,1)	2	0	0	2	2	$\langle 2,0,0\rangle$	$\langle 0,0,2\rangle$
(0,0,1,0)	1	0	1	1	2	$\langle 1,1,0\rangle$	$\langle 0,1,1\rangle$
(0,0,1,1)	1	1	1	1	3	$\langle 1,1,1\rangle$	$\langle 1,1,1\rangle$
(0,1,0,0)	1	1	1	1	3	$\langle 1,1,1\rangle$	$\langle 1,1,1\rangle$
(0,1,0,1)	2	0	1	2	3	$\langle 2,1,0\rangle$	$\langle 0,1,2\rangle$
(0,1,1,0)	2	1	0	2	3	$\langle 2,1,0\rangle$	$\langle 0,1,2\rangle$
(0,1,1,1)	2	1	1	2	4	$\langle 2,1,1\rangle$	$\langle 1,1,2\rangle$
(1,0,0,0)	1	1	1	1	3	$\langle 1,1,1\rangle$	$\langle 1,1,1\rangle$
(1,0,0,1)	1	1	0	1	2	$\langle 1,1,0\rangle$	$\langle 0,1,1\rangle$
(1,0,1,0)	0	0	2	2	2	$\langle 2,0,0\rangle$	$\langle 0,0,2\rangle$
(1,0,1,1)	0	1	1	1	2	$\langle 1,1,0\rangle$	$\langle 0,1,1\rangle$
(1,1,0,0)	0	1	2	2	3	$\langle 2,1,0\rangle$	$\langle 0,1,2\rangle$
(1,1,0,1)	1	1	1	1	3	$\langle 1,1,1\rangle$	$\langle 1,1,1\rangle$
(1,1,1,0)	1	0	1	1	2	$\langle 1,1,0\rangle$	$\langle 0,1,1\rangle$
(1,1,1,1)	1	1	0	1	2	$\langle 1,1,0\rangle$	$\langle 0,1,1\rangle$

Table 1: Models and distances involved in the running example.

Note that the original definition does not employ integrity constraints, which were introduced in [24].

The operator $\triangle_{\mu}^{\mathrm{Max}}$ is an arbitration operator, i.e., it is insensitive to whether there is a majority of formulae in the profile that are consistent. In general, $\triangle_{\mu}^{\mathrm{Max}}$ can be thought of as trying to minimise the maximum degree of compromise any formula in the profile will have to undergo before reaching agreement.

In Table 1, the columns labelled $d(\phi_1, u)$, $d(\phi_2, u)$ and $d(\phi_3, u)$ list the distances of the model in each row to each profile formula. According to the definition of d_{Max}, its value is the maximum of these three numbers. It is easy to see that the minimum value for this is 1 in the table, since there is no model at distance 0 to E. This means that all models of μ whose maximum distance from any profile formula is less or equal to 1 are the models of the merged formulae, i.e., $\mathrm{mod}(\triangle_{\mu}^{\mathrm{Max}}(E)) = \{(0,0,1,0), (1,0,0,0), (1,0,0,1), (1,1,1,0), (1,1,1,1)\}$.

2.1.2 The operator $\triangle_{\mu}^{\Sigma}$

Konieczny and Pino Pérez propose another operator, $\triangle_{\mu}^{\Sigma}$, defined originally in [22] and extended to employ integrity constraints in [23]. $\triangle_{\mu}^{\Sigma}$ uses the sum of the formula-to-model distances as the aggregate distance.

$$d_{\Sigma}(E, u) = \sum_{i=1}^{k} d(\phi_i, u)$$

Using d_{Σ}, the operator $\triangle_{\mu}^{\Sigma}$ can be defined as follows.

$$\mathrm{mod}(\triangle_{\mu}^{\Sigma}(E)) = \{\, u \in \mathrm{mod}(\mu) \mid d_{\Sigma}(E, u) \text{ is minimal} \,\}$$

The operator $\triangle_\mu^\Sigma$ tries to minimise the sum of the degree of compromise the formulae in the profile will have to undergo before reaching total agreement. In this sense, $\triangle_\mu^\Sigma$ is a majority operator which means that if a set of formulae in the profile agree, then $\triangle_\mu^\Sigma$ will attempt to compromise them less than the rest.

The distance d_Σ for the example profile is shown in the corresponding column in Table 1. It can be seen that the minimal d_Σ is 2 and that the models of μ that have minimal d_Σ are the models of the merge, i.e., $\mathrm{mod}(\triangle_\mu^\Sigma(E)) = \{(0,0,0,1),(0,0,1,0),(1,0,0,1),(1,1,1,0),(1,1,1,1)\}$.

2.1.3 The operator $\triangle_\mu^{\mathrm{Gmax}}$

$\triangle_\mu^{\mathrm{Gmax}}$ is third operator proposed in [23] by Konieczny and Pino Pérez (originally in [22], without integrity constraints). The aggregate distance used is the vector of individual model-to-formulae distances, sorted in descending order. In other words, if the vector of distances of u to the formulae in the profile is $V = \langle d(\phi_1, u), \ldots, d(\phi_k, u)\rangle$ and V^d is the vector obtained by sorting in descending order the elements of V, then $d_{\mathrm{Gmax}}(E,u) = V^d$. Using the lexicographic ordering $\leq_L$ for comparing vectors, the merging operator is defined as follows.

$$\mathrm{mod}(\triangle_\mu^{\mathrm{Gmax}}(E)) = \{\, u \in \mathrm{mod}(\mu) \mid d_{\mathrm{Gmax}}(E,u) \text{ is } \leq_L \text{-minimal} \,\}$$

The operator $\triangle_\mu^{\mathrm{Gmax}}$ attempts to minimise the maximum degree of compromise on any formula in the profile, and as such it is an arbitration operator. It is also known that $\triangle_\mu^{\mathrm{Gmax}}(E) \vdash \triangle_\mu^{\mathrm{Max}}(E)$ [22].

The column with the heading d_{Gmax} of Table 1 lists distance vectors, sorted in descending order. It is easy to see that the minimum distance vector sorted in descending order is $\langle 1,1,0\rangle$ and therefore, the models of the merge are $\mathrm{mod}(\triangle_\mu^{\mathrm{Gmax}}(E)) = \{(0,0,1,0),(1,0,0,1),(1,1,1,0),(1,1,1,1)\}$.

2.1.4 The operator $\triangle_\mu^{\mathrm{Gmin}}$

The operator $\triangle_\mu^{\mathrm{Gmin}}$ is defined by Everaere et al. in [13]. $\triangle_\mu^{\mathrm{Gmin}}$ is closely related to $\triangle_\mu^{\mathrm{Gmax}}$ in definitional terms, as the only difference is that the order in which the distance vectors are sorted is ascending as opposed to descending. Hence, if $V = \langle d(\phi_1, u), \ldots, d(\phi_k, u)\rangle$ is the distance vector and V^a is obtained by sorting in ascending order the elements of V, then $d_{\mathrm{Gmin}}(E,u) = V^a$. The merging operator is defined similarly.

$$\mathrm{mod}(\triangle_\mu^{\mathrm{Gmin}}(E)) = \{\, u \in \mathrm{mod}(\mu) \mid d_{\mathrm{Gmin}}(E,u) \text{ is } \leq_L \text{-minimal} \,\}$$

The operator $\triangle_\mu^{\mathrm{Gmin}}$ is a majority operator: like the $\triangle_\mu^\Sigma$ operator, if a set of formulae in the profile agree without needing to compromise, then this set will be kept intact and the rest of the profile is going to be compromised until consistency has been achieved.

The distances of the models to the example profile according to d_{Gmin} are listed in the corresponding column in Table 1. Even though the sorted vectors under d_{Gmin} are just reversed versions of the vectors under d_{Gmax}, we can see

Decision problem		Complexity class
$\triangle_\mu^{\text{Max}}(E)$	$\vdash \psi$	Θ_2^p-complete
$\triangle_\mu^{\Sigma}(E)$	$\vdash \psi$	Θ_2^p-complete
$\triangle_\mu^{\text{Gmax}}(E)$	$\vdash \psi$	Δ_2^p-complete
$\triangle_\mu^{\text{Gmin}}(E)$	$\vdash \psi$	Δ_2^p-complete

Table 2: Worst-case complexity results for the query answering problem with the four operators in this paper. These results originate in [21].

that the minimal distance vector is $\langle 0, 0, 2\rangle$ and that, as a consequence, the merged profile is different, i.e., $\text{mod}(\triangle_\mu^{\text{Gmin}}(E)) = \{(0, 0, 0, 1)\}$.

2.1.5 Complexity of merging

It is well known that merging operators are closely related to revision operators, and therefore, we cannot hope to improve on complexity results found in the belief revision literature [12]. In fact, as Table 2 illustrates, most operators are to be found at the second level of the polynomial hierarchy [21]. Δ_2^p is the class of problems that can be solved in polynomial time given access to an oracle that decides on any NP problem in constant time; the class Θ_2^p contains problems in Δ_2^p that require a number of calls to the NP oracle that is bounded by a logarithmic function of the input size.

2.2 Dilations and sphere semantics

Bloch and Lang, in [8], explore how some operations from mathematical morphology translate into a logical framework. One of the most basic operations is the dilation of a set, which translates into the dilation of a formula (or its set of models). Let $B : \mathcal{M} \to 2^{\mathcal{M}}$ be a function from a model to a set of models, called the *structuring element*. Then, the dilation $D_B(\phi)$ is defined as the function with the following property.

$$\text{mod}(D_B(\phi)) = \{\, u \mid B(u) \cap \text{mod}(\phi) \neq \emptyset \,\}$$

The most common structuring element, and the one we will exclusively use in the rest of this paper, is the unit ball using the Hamming distance.

$$B(u) = \{\, v \mid d(u, v) \leq 1 \,\}$$

We will omit the structuring element from now on. Iterated dilations are defined inductively, $D^n(\phi) = D(D^{n-1}(\phi))$ and $D^0(\phi) = \phi$. We call n the dilation degree.

Bloch and Lang proceed to explore how knowledge merging and belief revision can be expressed in terms of dilations of formulae. They examine Dalal's operator $\circ$, [10], and show that $\phi \circ \psi = D^n(\phi) \wedge \psi$, where n is the smallest number for which $D^n(\phi) \wedge \psi$ is consistent. On knowledge merging, they also

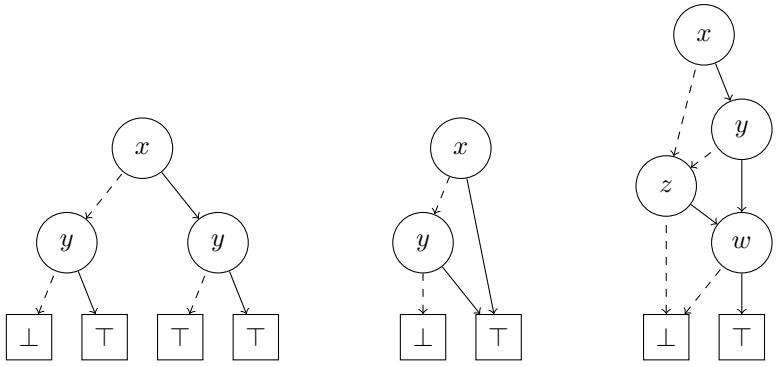

Figure 1: (a) decision tree for $x \vee y$ (b) BDD for $x \vee y$ (c) BDD for $((x \wedge y) \vee z) \wedge w$.

conclude that $\triangle_{\mu}^{\mathrm{Max}}(E) = \mu \wedge D^{d_{\min}}(\phi_1) \wedge \ldots \wedge D^{d_{\min}}(\phi_k)$ where $d_{\min}$ is the smallest number such that $\mu \wedge D^{d_{\min}}(\phi_1) \wedge \ldots \wedge D^{d_{\min}}(\phi_k)$ is consistent.[2]

The idea to use dilations for theory change and knowledge merging is closely related to the semantics by Grove in [17]. Grove provided a semantics for belief revision operators $*$ based on a collection of sets of models S_ϕ (a *system of spheres centred on* ϕ) which is totally-ordered by set-inclusion, as follows.

$$\begin{aligned} \mathrm{mod}(\phi * \psi) \quad = \quad & \{\, u \in \mathrm{mod}(\psi) \mid \exists X \in S_\phi, u \in X, X \cap \mathrm{mod}(\psi) \neq \emptyset, \\ & \nexists Y \in S_\phi, Y \subset X, Y \cap \mathrm{mod}(\psi) \neq \emptyset \,\} \end{aligned}$$

The original formulation is more complex as it is expressed in a logic-agnostic way, involving complete and consistent theories instead of models, whereas here, we have restricted it to propositional logic.

2.3 Binary decision diagrams

2.3.1 Definitions

Binary decision diagrams (BDDs, also known as *reduced ordered binary decision diagrams*) are data structures for the representation of boolean functions, and by implication, propositional formulae. They originate in circuit verification and symbolic model checking. For an accessible introduction see [3] and for one of the seminal papers, see [9].

We will first look at a simpler structure, the decision tree (DT). The DT of the formula $x \vee y$ is shown in Figure 1a. Formally, a DT is a full binary tree of height $p + 1$ consisting of internal nodes, each labelled with an atom (*variable nodes*) and leaf nodes, each labelled with a logical constant (*terminal nodes*). Each level of internal nodes in a DT is labelled with the same atom and no other

[2] We have modified the results by Bloch and Lang by incorporating integrity constraints which were not present in the original formulation.

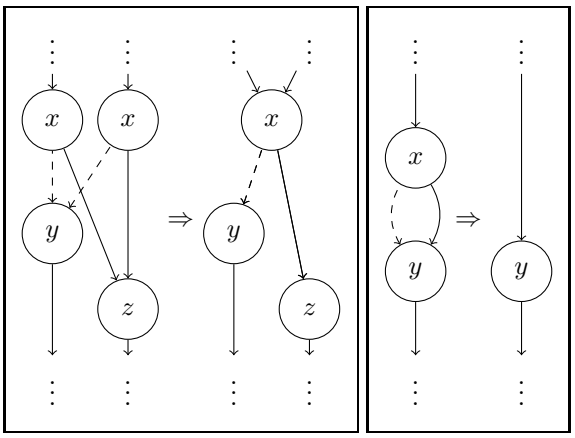

Figure 2: (a) Reduction rule 1, (b) Reduction rule 2

level is labelled with that atom. Each variable node has two outgoing edges, corresponding to the evaluation path taken according to the truth-value of the atom labelling the node; the solid edge corresponds to the path taken when the atom is true and the dashed edge to the path taken when the atom is false. In BDD jargon, the descendant of the dashed edge is called the *low successor* and the descendant of the solid edge, the *high successor*.

A DT encodes the truth-table of a formula and, therefore, allows checking whether a model satisfies a formula by starting at the root of the DT and following the dashed edges when a variable is false and the solid edges when it is true; the path terminates at the node labelled with $\top$ iff the model satisfies the formula. Thus, DTs represent the set of models satisfying a formula; one can enumerate all paths from the root that terminate at $\top$-nodes. A DT always has $2^{p+1} - 1$ nodes and is therefore too costly to use as a practical data structure.

The BDD for $x \vee y$ is shown in Figure 1b. It can be obtained from the DT by folding together shared subtrees and removing nodes whose successors are identical. More formally, BDDs are rooted, directed-acyclic graphs obeying the restrictions for DTs, plus the following two rules (cf. Figure 2).

1. Any two distinct variable nodes that are labelled with the same variable and the same corresponding low and high successors are merged (Figure 2a). Similarly, all copies of the terminal nodes $\top$ and $\bot$ are replaced with only one occurrence of each.

2. Any variable node with identical low and high successors is removed, and its parent is linked to its successor correspondingly (Figure 2b).

These rules are called reduction rules since they can be used to transform (reduce) a DT into a logically equivalent BDD.

2.3.2 Properties and Algorithms

A BDD can be much more compact than its corresponding DT. Figure 1c shows the BDD for the formula $((x \wedge y) \vee w) \wedge z$ which contains 6 nodes as opposed to 31 in the DT. BDDs can still have exponential size in the worst case and their compactness may depend on the choice of variable ordering (i.e., the sequence of atoms on any branch). Some properties of interest are summarised here.

BDDs have a *strong canonicity* property, i.e., any two equivalent formulae are represented by isomorphic BDDs (under a given variable ordering). Consequently, there is exactly one BDD for the constant $\top$ and similarly for $\bot$. Therefore, it is possible to check a formula represented by a BDD for satisfiability and validity in polynomial time (thus converting a formula to a BDD is NP- and coNP-hard in the length of the formula). In addition, implementations of BDD library packages use hash-tables for the storage of BDD nodes, making equivalence checking an amortised constant time operation.

Efficient algorithms for performing logical operations on BDDs exist. The algorithm `apply(`B_ϕ`,` B_ψ`,` $\bullet$`)` takes two BDDs representing ϕ and ψ and computes a BDD representing $\phi \bullet \psi$ where $\bullet$ is any binary boolean connective. The worst-case space and time complexity is $\mathcal{O}(|B_\phi| \cdot |B_\psi|)$, where $|B|$ denotes the number of nodes in the BDD B. Converting a formula to a BDD can be done by consecutive applications of `apply`, using the BDDs for the propositional variables (these consist of one variable node and the two terminal nodes in the obvious arrangement). By using `apply(`B_ϕ`,` $B_\top$`,` $\nleftrightarrow$`)`, the negation of a formula can be computed in linear time and space in the size of the input BDD. There is also a linear-time algorithm called `restrict`(B_ϕ, x, c) that computes the BDD resulting from the substitution of a variable x by a constant c in the BDD for a formula ϕ, with the result denoted as $\phi|_{x=c}$.

BDDs can also be viewed as an *if-then-else normal form* (INF). INF employs the if-then-else operator, a ternary boolean operator, $x \rightarrow \phi, \psi$, where x is a propositional variable and ϕ and ψ are formulae. Its intuitive meaning is that if x is true, then the truth-value of the formula is that of ϕ, and if x is false, then the truth-value of the formula is that of ψ. In other words, $x \rightarrow \phi, \psi$ is equivalent to $(x \wedge \phi|_{x=\top}) \vee (\neg x \wedge \psi|_{x=\bot})$. It can be shown that the if-then-else operator is complete (i.e., all usual connectives can be expressed using the if-then-else operator and the constants $\top$ and $\bot$) and that BDDs effectively encode the INF of a formula in a manner that avoids redundancy by merging identical sub-expressions. We will omit the application of restriction of ϕ and ψ to x whenever it is clear that x does not appear in ϕ and ψ, which is the case for BDDs due to the requirement that variable nodes are labelled in accordance to the variable ordering. Finally, we will use `ite`(x, ϕ, ψ) for the (constant-time) algorithm that constructs the BDD for $x \rightarrow \phi, \psi$.

3 Algorithms for merging

The merging operators we consider use a concept of distance between a model and the set of models of a formula. We use the notion of dilation in order to generate the sets of models of increasing distance from a formula. These sets would be intractable to encode explicitly and this is why we use BDDs to represent and manipulate them.

The algorithms we present compute a representation of the result of the merging, as opposed to answering a query (i.e., whether the merged knowledge base entails a given formula). The motivation for this choice is that, especially when using BDDs but also in other cases, it may be expensive to compute the merged formula but relatively inexpensive to represent it and to subsequently reason with it. Therefore, the computed results of our algorithms are to be retained in BDD form and used as compiled knowledge; in this way, if B is a BDD we can use it to answer questions of the form $B \models \phi$ either by converting ϕ into a BDD and using it accordingly,[3] or by using specialised algorithms (when ϕ is in conjunctive normal form (CNF), $B \models \phi$ can be decided in running time linear in both the size of B and of ϕ).

From now on we will sometimes abuse notation and make no distinction between a formula and its BDD, or between a logical connective and the corresponding BDD algorithm for computing that connective. We will also use equality, $=$, interchangeably with equivalence, $\leftrightarrow$, something allowed by the strong canonicity of BDDs.

3.1 A BDD algorithm for dilation

A key component to our approach in implementing the knowledge merging operators presented in Section 2.1 is an algorithm that, given a BDD for a formula, computes the BDD that represents the Dalal dilation of that formula. In this section we present such an algorithm.

We will now explain the algorithm and show its correctness. First, assume that the input formula χ is represented by a BDD, under a given variable ordering, of the form $x \rightarrow \phi, \psi$ as in Figure 3a. Assume also that a model u is a model of the dilation of χ, $u \models D(\chi)$. We distinguish two cases. First, it may be that $u \models \chi$ in which case u is by definition a model of the dilation. Second, if $u \not\models \chi$ it must be that there exists a propositional variable x' such that if its value in u is inverted, we obtain another model u' such that $u' \models \chi$.

The second case can be decomposed further into three sub-cases. First, it could be that $x = x'$, in which case we need to construct the BDD which admits the same models as χ but with inverted truth-values for x. This can be achieved by swapping places for ϕ and ψ, effectively constructing the BDD for $x \rightarrow \psi, \phi$.

The second and third sub-cases are dual and occur when $x \neq x'$. If $u \models x$ then we know that u is obtained by taking a model u' that sets x to $\top$, satisfies

[3] For example, by computing $\texttt{apply}(B, B_\phi, \rightarrow)$ and then comparing the result with $B_\top$.

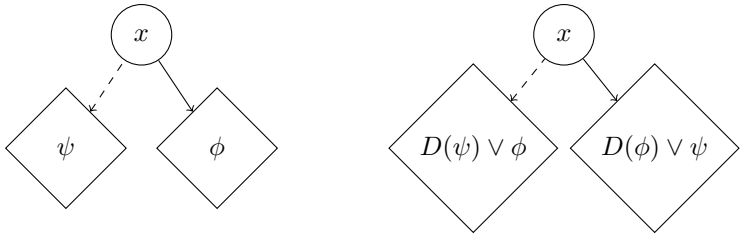

Figure 3: (a) Left, the input BDD χ; (b) Right, the result of computing $D(\chi)$.

ϕ and that is at distance one from u, i.e., $u \models x \wedge D(\phi)$. Similarly, if $u \models \neg x$ then it must that $u \models \neg x \wedge D(\psi)$.

Putting these cases together we obtain a formulation for dilation:

$$D(\chi) \leftrightarrow (x \rightarrow \phi, \psi) \vee (x \rightarrow \psi, \phi) \vee (x \rightarrow D(\phi), D(\psi))$$

By making use of the fact that $x \rightarrow \phi, \psi$ is equivalent to $(x \wedge \phi) \vee (\neg x \wedge \psi)$ and that for all a, $a \vdash D(a)$ we obtain:

$$D(\chi) \leftrightarrow (x \rightarrow (D(\phi) \vee \psi), (D(\psi) \vee \phi))$$

This expression, along with the termination conditions $D(\top) = \top$ and $D(\bot) = \bot$ gives us a recursive algorithm for computing the BDD for the dilation of a formula. The resulting BDD is shown in Figure 3b.

The potential compactness of BDDs rests heavily on sharing sub-graphs in the graph of the BDD. This means that the naïve implementation of the algorithm would re-compute the dilation of a sub-graph multiple times, as many as the number of arcs that lead to it. In order to avoid this and to improve efficiency we use a standard technique for BDD algorithms which consists in the use of a hashmap for the storing of already-computed dilations of sub-BDDs. Algorithm 1 uses the following sub-routines with regards to the hashmap. The routine `is-in-hashmap`(a) returns the answer to whether the dilation of a has been computed already and is in the hashmap. The routine `retrieve-from-hashmap`(a) retrieves the dilation of a assuming it has already been computed. The routine `store-in-hashmap`(a, a') associates the BDD a with its dilation a' within the hashmap.

The complexity of the problem of computing the BDD for the dilation of another BDD is not known. It is known that computing the dilation of a formula expressed in disjunctive normal form (DNF) is polynomial [8], but the complexity of the general problem is unclear. Another related result has to do with the complexity of DISTANCE-SAT, the problem of deciding whether a model is at distance at most d from the models of a given formula [5]. While DISTANCE-SAT is NP-complete in the general case, it is of polynomial complexity when the formula is represented by a BDD.

Algorithm 1 $D(\chi)$

if $\chi = \top$ or $\chi = \bot$ **then**
    return χ
if `is-in-hashmap`(χ) **then**
    return `retrieve-from-hashmap`(χ)
Let x, ϕ, ψ be such that $\chi = x \rightarrow \phi, \psi$
$r \leftarrow$ `ite`$(x, D(\phi) \vee \psi, D(\psi) \vee \phi)$
`store-in-hashmap`(χ, r)
return r

3.2 An algorithm for $\triangle_\mu^{\text{Max}}$

Algorithm 2 for $\triangle_\mu^{\text{Max}}$ is a straightforward implementation of the results by Bloch and Lang. We dilate each formula in the profile once, take the conjunction of the dilations with the integrity constraints and check for consistency. If the result is consistent then that is the result of $\triangle_\mu^{\text{Max}}$, otherwise we repeat the process. Note that we use the variable A as a vector, or one-dimensional array and refer to its i-th value as A_i.

Lemma 1. *Algorithm 2 computes* $\triangle_\mu^{\text{Max}}$.

Proof. By definition, $u \in \text{mod}(\triangle_\mu^{\text{Max}}(E))$ iff $u \in \text{mod}(\mu)$ and for any $v \in \text{mod}(\mu)$, $d_{\text{Max}}(E, u) \leq d_{\text{Max}}(E, v)$. Clearly, there exists a minimum distance $d_{\min}$ such that for any $u \in \text{mod}(\triangle_\mu^{\text{Max}}(E))$, $d_{\text{Max}}(E, u) = d_{\min}$.

Using the triangle inequality for the Hamming distance and induction over n, it is easy to prove that $\text{mod}(D^n(\phi))$ contains all models whose distance from ϕ is less or equal to n. Therefore, we can re-write the merging as $\triangle_\mu^{\text{Max}}(E) = \mu \wedge \bigwedge_{i=1}^k D^{d_{\min}}(\phi_i)$. In addition, for all $0 \leq n < d_{\min}$, $\mu \wedge \bigwedge_{i=1}^k D^n(\phi_i)$ must be inconsistent, otherwise this would contradict the assumption about the minimality of $d_{\min}$.

Therefore, by constructing $\mu \wedge \bigwedge_{i=1}^k D^n(\phi_i)$ for increasing values of $0 \leq n$ and testing the result for consistency, we will eventually compute $\triangle_\mu^{\text{Max}}(E)$ as the first consistent conjunction. This completes the proof. □

Algorithm 2 $\triangle_\mu^{\text{Max}}(E)$

$A \leftarrow \langle \phi_1, \ldots, \phi_k \rangle$
$C \leftarrow \mu \wedge \bigwedge_{i=1}^k A_i$
while $C = \bot$ **do**
    for $i = 1, \ldots, k$ **do**
        $A_i \leftarrow D(A_i)$
    $C \leftarrow \mu \wedge \bigwedge_{i=1}^k A_i$
return C

Note that since the distance between any two models can be no greater than

p, it follows that $D^p(\phi) = \top$, for any consistent ϕ. Therefore, in the worst case, the algorithm will perform $\mathcal{O}(p \cdot k)$ dilations and $\mathcal{O}(p \cdot k)$ conjunctions.

3.3 An algorithm for $\triangle_\mu^\Sigma$

The notion of aggregate distance for $\triangle_\mu^\Sigma$ is the sum of the distances to the formulae in the profile. As with $\triangle_\mu^{\text{Max}}$, there exists a minimum profile-to-model distance at which all the models of the merge will be. Therefore, we enumerate the possible aggregate distances in increasing order from 0 to $p \cdot k$ (the maximum distance of a model to a formula times the number of formulae). For each such distance d, we generate all compositions in k parts.[4] Each such composition $\langle c_1, \ldots, c_k \rangle$ represents a vector of dilations of the profile formulae, $\langle D^{c_1}(\phi_1), \ldots, D^{c_k}(\phi_k) \rangle$. The conjunction of these dilations will contain the models at distance less or equal to d, since $\sum_{i=1}^k c_i = d$ by the definition of composition. We produce the disjunction of all these conjunctions of distance d. When the compositions of d are exhausted, if we have found at least one consistent result then we terminate, otherwise we repeat for $d+1$. Algorithm 3 presents this sequence of steps. Note that $A_{i,j}$ is a two-dimensional array, or table and A_0 is the first row. Also, note that for distances d with $p < d$ we ignore the compositions that include elements larger than p since these do not correspond to valid vectors of dilations (due to the fact that $D^n(\phi) = \top$ for all n such that $p \leq n$).

Algorithm 3 $\triangle_\mu^\Sigma(E)$

$A_0 \leftarrow \langle \phi_1, \ldots, \phi_k \rangle$
$solution \leftarrow \mu \wedge \bigwedge_{i=1}^k A_{0,i}$
$d \leftarrow 1$
while $solution = \bot$ **do**
    if $d \leq p$ **then**
        for $i = 1, \ldots, k$ **do**
            $A_{d,i} \leftarrow D(A_{d-1,i})$
    for all $V \in compositions(d, k)$ **do**
        if $\max\{ V_i \mid 1 \leq i \leq k \} \leq p$ **then**
            $solution \leftarrow solution \vee \left(\mu \wedge \bigwedge_{i=1}^k A_{V_i,i} \right)$
    $d \leftarrow d + 1$
return *solution*

Lemma 2. *Algorithm 3 computes* $\triangle_\mu^\Sigma(E)$.

Proof. As with $\triangle_\mu^{\text{Max}}$, there exists a minimum distance $d_{\min}$ such that for any model $u \in \text{mod}(\triangle_\mu^{\text{Max}}(E))$, $d(E, u) = \sum_{i=1}^k d(\phi_i, u) = d_{\min}$. Therefore, we can

[4] This is a standard combinatorial problem and many algorithms exist; we use the library [4] which provides an algorithm that computes the next composition in the enumeration in time linear to k.

express the result of the merging as:

$$\mathrm{mod}(\triangle_\mu^\Sigma(E)) = \bigcup_{\substack{\langle c_1,\ldots,c_k\rangle \in \\ compositions(d_{\min},k)}} \mathrm{mod}(\mu) \cap \bigcap_{i=1}^{k} \mathrm{mod}(D^{c_i}(\phi_i))$$

In addition, for all d such that $0 \leq d < d_{\min}$, any expression of the form $\mu \wedge \bigwedge_{i=1}^{k} D^{c_i}(\phi_i)$ where $\langle c_1, \ldots, c_k\rangle \in \mathit{compositions}(d,k)$, will be inconsistent. □

In the worst case, Algorithm 3 will perform $\mathcal{O}(p \cdot k)$ dilations. There are $\binom{n+k-1}{k-1}$ weak compositions of n into k parts and from this it can be shown that the number of compositions checked in the worst case (when all distances up to $k \cdot p$ need to be enumerated) is $\binom{(p+1)\cdot k}{k}$, an upper bound of which is $((p+1)\cdot e)^k = \mathcal{O}(p^k)$.

3.4 An algorithm for $\triangle_\mu^{\mathrm{Gmax}}$

The basic idea behind Algorithm 5 is to enumerate the (sorted in descending order) distance vectors in increasing lexicographic order. For each such vector all its permutations are generated, and each permutation is translated into a conjunction of corresponding dilations (with the integrity constraints). We keep the disjunction of all such conjunctions as a BDD. When the permutations of a vector have been exhausted, if we have a consistent disjunction we stop, otherwise we repeat for the next vector in the enumeration.

Enumerating vectors in such a manner is done by Algorithm 4. Whenever `NextDistanceVector()` is called it returns the next vector in the enumeration. The intuition behind this algorithm is that its state is a one-dimensional array of numbers D, so that D_i indicates how many formulae are dilated i times; this array is used as a counter in base k with the added constraint that the sum of its elements cannot exceed k. Each successive call to the algorithm will attempt to increase D_1, corresponding to adding more 1s in the distance vector, i.e., least elements according to the sorting order. When that is not possible because the total number of formulae dilated would become greater than k then that element is set to zero and D_2 is incremented, and so on. Finally, the actual sorted distance vector is produced by going through the array D and adding elements to the distance vector as required. Note that D is an array of p elements, initially set to zero, which keeps the required state across successive runs of the algorithm, while V is a one-dimensional array of size k.

As an example, if there are 4 atoms in the language ($p = 4$), 3 formulae in the profile ($k = 3$) and the last invocation of `NextDistanceVector` returned the vector $\langle 2,1,0\rangle$, then it will be the case that $D = \langle 1,1,0,0\rangle$, the next call would return $\langle 2,1,1\rangle$ after which it would be the case that $D = \langle 2,1,0,0\rangle$.

Lemma 3. *Algorithm 4 enumerates vectors in increasing lexicographic order, such that each vector is itself sorted in descending order.*

Algorithm 4 `NextDistanceVector()` (enumerates vectors for $\triangle_{\mu}^{\mathrm{Gmax}}$)

$V \leftarrow \langle 0, \ldots, 0 \rangle$
$i \leftarrow 1$
repeat
  if $\sum_{j=1}^{p} D_j < k$ **then**
    $D_i \leftarrow D_i + 1$
  else
    $D_i \leftarrow 0$
  $i \leftarrow i + 1$
until $0 < D_i$
$l \leftarrow 1$
$i \leftarrow p$
while $0 < i$ **do**
  if $0 < D_i$ **then**
    for $j = 1, \ldots, D_i$ **do**
      $V_l \leftarrow i$
      $l \leftarrow l + 1$
  $i \leftarrow i - 1$
while $l \leq k$ **do**
  $V_l \leftarrow 0$
  $l \leftarrow l + 1$
return V

Proof. It is easy to see that by its construction, V will be always sorted in descending order. Therefore, we need to prove that for any two vectors V_1, V_2 produced by successive runs of Algorithm 4, it will be the case that $V_1 <_L V_2$.

Let us denote as d the value of the array D in the first run and as d' in the next successive run. From the first loop we can see that d' will have a prefix of zeros of length $0 \leq i$, followed by an element d'_{i+1} such that $d_{i+1} + 1 = d'_{i+1}$ and, finally, a common suffix with d, i.e., $d_j = d'_j$, for all $i + 1 < j \leq p$.

$$d = \langle d_1, \ldots \ , \quad d_{i+1} \quad , d_{i+2}, \ldots, d_p \rangle$$
$$d' = \langle \overbrace{0, \ldots, 0}^{i}, d_{i+1} + 1, d_{i+2}, \ldots, d_p \rangle$$

By translating d and d' into distance vectors V and V' through the remainder of the algorithm it follows that V and V' will share a prefix of length $l = \sum_{j=i+1}^{p} d_j$, and will differ at element $l + 1$ with $V'_{l+1} = i + 1 > i = V_{l+1}$.

$$V = \langle \overbrace{p, \ldots, p}^{d_p}, \overbrace{p-1, \ldots, p-1}^{d_{p-1}}, \ldots, \overbrace{i+1, \ldots, i+1}^{d_{i+1}}, \ldots, \overbrace{1, \ldots, 1}^{d_1} \rangle$$
$$V' = \langle \overbrace{p, \ldots, p}^{d_p}, \overbrace{p-1, \ldots, p-1}^{d_{p-1}}, \ldots, \overbrace{i+1, \ldots, i+1, i+1}^{d_{i+1}+1}, 0, \ldots, 0 \rangle$$

Thus, $V <_L V'$. Moreover, it is easy to see that there is no valid vector in between V and V' that is sorted. □

Algorithm 5 $\triangle_\mu^{\mathrm{Gmax}}(E)$

$A_0 \leftarrow \langle \phi_1, \ldots, \phi_k \rangle$
$C \leftarrow \mu \wedge \bigwedge_{i=1}^{k} A_{0,i}$
$d \leftarrow 0$
while $C = \bot$ **do**
    $d \leftarrow d + 1$
    for all $i = 1, \ldots, k$ **do**
        $A_{d,i} \leftarrow D(A_{d-1,i})$
    repeat
        $V \leftarrow$ `NextDistanceVector`()
        for all $\langle c_1, \ldots, c_k \rangle \in \mathit{permutations}(V)$ **do**
            $C \leftarrow C \vee (\mu \wedge \bigwedge_{i=1}^{k} A_{c_i,i})$
    until $d < \max\{ V_i \mid 1 \leq i \leq k \}$ or $C \neq \bot$
return C

By definition, if a vector $\langle a_1, \ldots, a_k \rangle$ is followed by a vector $\langle b_1, \ldots, b_k \rangle$ in the enumeration then $\max\{ a_i \mid 1 \leq i \leq k \} \leq \max\{ b_i \mid 1 \leq i \leq k \}$. This allows for an optimisation in the way we produce the consecutive dilations in Algorithm 5: at any given point, if $m = \max\{ V_i \mid 1 \leq i \leq k \}$ where V is the current sorted distance vector, we only need the dilations $D^n(\phi_j)$ with $1 \leq j \leq k$ and $1 \leq n \leq m$ for checking the corresponding conjunctions. This is also why the **repeat**/**until** loop uses $d < \max\{ V_i \mid 1 \leq i \leq k \}$ as part of its terminating condition; if the enumeration produces a vector whose maximum element refers to a dilation that has not yet been computed, then execution must return to the loop that dilates the profile formulae.

Lemma 4. *Algorithm 5 computes* $\triangle_\mu^{\mathrm{Gmax}}(E)$.

Proof. Of all the sorted distance vectors, there will be one, $V^{\min}$, such that,

$$\mathrm{mod}(\triangle_\mu^{\mathrm{Gmax}}(E)) = \bigcup_{\substack{\langle t_1, \ldots, t_k \rangle \in \\ \mathit{permutations}(\{V_1^{\min}, \ldots, V_k^{\min}\})}} \mathrm{mod}(\mu) \cap \bigcap_{i=1}^{k} \mathrm{mod}(D^{t_i}(\phi_i))$$

In addition, for all sorted vectors V such that $\langle 0, \ldots, 0 \rangle \leq_L V <_L V^{\min}$, for all permutations P of V, the conjunction $\mu \wedge \bigwedge_{i=1}^{k} D^{P_i}(\phi_i)$ will be inconsistent. Therefore, all we need to do is to enumerate sorted distance vectors in increasing lexicographic order and examine their permutations. □

In the worst case, this algorithm performs $\mathcal{O}(p \cdot k)$ dilations and $(p+1)^k \cdot k = \mathcal{O}(p^k)$ conjunctions (the number of distinct unsorted distance vectors times their length). An optimisation that we use in our implementation is that whenever we dilate all the formulae we can check if the conjunction of the profile formulae dilated to the largest degree permitted at that iteration of the algorithm is inconsistent, i.e., if $\mu \wedge \bigwedge_{i=1}^{k} D^d(\phi_i) = \bot$. If it is, then any permutation of distance vectors with dilations up to d will be inconsistent, and therefore can

be skipped. This has a beneficial effect on average-case complexity (as will be seen in Section 4) and a minor one in worst-case complexity ($(p+1)^{k-1} \cdot k^2$ conjunctions).

3.5 An algorithm for $\triangle_\mu^{\text{Gmin}}$

We will provide the motivation behind Algorithm 6 using the running example summarised in Table 1, in Section 2.1. As already noted, $\triangle_\mu^{\text{Gmin}}$ is a majority operator and as such it will not dilate majority sets of agreeing formulae while increasingly compromising the remaining formulae, until consistency is achieved.

From Table 1 we can see that $\mu \wedge \phi_1 \vdash \bot$, that $\mu \wedge \phi_2 \nvdash \bot$ and that $\mu \wedge \phi_3 \nvdash \bot$. From this it should be clear that ϕ_1 will need to be dilated before forming a part of the merge. Since $\triangle_\mu^{\text{Gmin}}$ is a majority operator, we should check whether we can extend these two consistent conjunctions as much as possible before considering dilating (i.e., compromising) the remaining formulae. Indeed, we can see that $\mu \wedge \phi_2 \wedge \phi_3 \nvdash \bot$ and that there is no other pair of formulae that has this property (i.e., $\mu \wedge \phi_1 \wedge \phi_2 \vdash \bot$ and $\mu \wedge \phi_1 \wedge \phi_3 \vdash \bot$). Of course, $\mu \wedge \phi_1 \wedge \phi_2 \wedge \phi_3 \vdash \bot$.

The conjunction $\mu \wedge \phi_2 \wedge \phi_3$ has one model, at distance 0 from both ϕ_2 and ϕ_3. As such it corresponds to a distance vector of $\langle 0, 0, x\rangle$ for some $1 \leq x \leq 3$. We can see that this vector will be strictly smaller than any other vector corresponding to a conjunction of fewer than 2 undilated formulae, i.e., one undilated formula ($\langle 0, 0, x\rangle <_L \langle 0, y, z\rangle$ with $1 \leq y \leq z$).

Consequently, we generate the dilations of degree 1 and test whether any dilations of the formulae missing from $\mu \wedge \phi_2 \wedge \phi_3$ (i.e., ϕ_1) can consistently extend this conjunction. Even though $\mu \wedge D^1(\phi_1) \nvdash \bot$, it is still the case that $\mu \wedge \phi_2 \wedge \phi_3 \wedge D^1(\phi_1) \vdash \bot$. Therefore we repeat, and dilate all formulae up to degree 2. Now it is obvious that $\mu \wedge \phi_2 \wedge \phi_3 \wedge D^2(\phi_1) \nvdash \bot$ corresponding to the distance vector $V = \langle 0, 0, 2\rangle$. It should also be clear that there is no consistent conjunction that corresponds to a distance vector strictly smaller than V. Indeed, $\triangle_\mu^{\text{Gmin}}(E) = \mu \wedge \phi_2 \wedge \phi_3 \wedge D^2(\phi_1) \nvdash \bot$. We formalise this process in Algorithm 6.

First, we define a structure that will help us keep track of combinations of dilations of formulae. Let n be a number such that $0 \leq n \leq k$ and $V = \langle (a_1, b_1), \ldots, (a_n, b_n)\rangle$ be a vector of pairs of numbers with $1 \leq a_i \leq k$ and $0 \leq b_i \leq p$ for all i with $1 \leq i \leq n$. We use $|V| = n$ to denote the length of V, and use $f(V_i) = a_i$ and $d(V_i) = b_i$ as shorthands for referring to the individual elements of V. Each pair of numbers in such a vector corresponds to a profile formula ($f(V_i) = j$ represents ϕ_j) and its dilation degree ($d(V_i)$). Also, we abbreviate the conjunction of the dilations represented by such a vector (with the integrity constraints) by $C(V)$:

$$C(V) = \mu \wedge \bigwedge_{i=1}^{|V|} D^{d(V_i)}\left(\phi_{f(V_i)}\right)$$

We will only be interested in vectors of this form that satisfy the following

requirements. We will call vectors that do so, prefix vectors.

- $C(V) \nvdash \bot$
- All $f(V_i)$ are distinct: for all i, j such that $1 \leq i, j \leq |V|$ and $i \neq j$, it is the case that $f(V_i) \neq f(V_j)$.
- The sequence $d(V_i)$ is non-decreasing: for all i, j such that $1 \leq i \leq j \leq |V|$, $d(V_i) \leq d(V_j)$ holds.

We denote the extension of a prefix vector by $\oplus$:

$$V = \langle (a_1, b_1), \ldots, (a_n, b_n) \rangle \oplus (a_{n+1}, b_{n+1})$$

A vector V' will be called an extension of V if V' can be built by repeated application of $\oplus$ and appropriate pairs of integers. If V, V' are prefix vectors, we will say that V *agrees with* V' if for all i such that $1 \leq i \leq \min\{|V|, |V'|\}$, it is the case that $d(V_i) = d(V'_i)$.

Next we define over n two sets of prefix vectors, W_n and X_n, that will keep track of the combinations of dilations of formulae that we generate during the execution of the algorithm. Informally, the set X_n will contain the maximal consistent prefix vectors with formulae of dilation degree up to n. The set W_n will contain the longest prefix vectors in X_n.

More formally, we define the two sequences of sets using joint induction as follows. The set X_n, for $0 \leq n$, is the largest set of vectors generated by maximally and consistently extending all vectors in W_{n-1} using dilations of degree n. In other words a prefix vector V is in X_n if there is a prefix vector V' in W_{n-1} such that V is an extension of V', every new dilation appearing in V is of degree n and V cannot be extended with any more dilations of degree n consistently. In symbols, $V \in X_n$ if the following conditions hold.

- There exists $V' \in W_{n-1}$ such that V is an extension of V'.
- For all i with $1 \leq i \leq k$ such that there is no j with $f(V_j) = i$, $C(V) \wedge D^n(\phi_i) \vdash \bot$.

We set $W_{-1} = \{\langle\rangle\}$, a set containing an empty prefix vector. The set W_n, for n with $0 \leq n \leq p$ is defined to be a subset of X_n such that the length of vectors in W_n is maximal, i.e., $V \in W_n$ if $V \in X_n$ and for all $V' \in X_n$, $|V'| \leq |V|$. A corollary of these definitions is that for any two vectors $V, V' \in W_n$, $|V| = |V'|$ and $d(V_i) = d(V'_i)$ for all i.

Algorithm 6 makes use of `maxcons`(V, A_d), a sub-routine that computes the maximal consistent extensions of V with dilations of degree d taken from the array A_d. Formally, `maxcons` returns a set of prefix vectors that are maximal consistent extensions of V, and the formulae used to extend V are taken from $A_{d,1}, \ldots, A_{d,k}$.

Lemma 5. *Algorithm 6 computes* $\triangle_\mu^{\mathrm{Gmin}}(E)$.

Algorithm 6 $\triangle_{\mu}^{\mathrm{Gmin}}(E)$

$A_0 \leftarrow \langle \phi_1, \ldots, \phi_k \rangle$
$S \leftarrow \{\langle\rangle\}$
$d \leftarrow 0$
while $\max\{\,|V| \mid V \in S\,\} < k$ **do**
    if $1 \leq d$ **then**
        for all $i = 1, \ldots, k$ **do**
            $A_{d,i} \leftarrow D(A_{d-1,i})$
    $S' \leftarrow \{\langle\rangle\}$
    for all $V \in S$ **do**
        for all $V' \in \texttt{maxcons}(V, A_d)$ **do**
            if $\max\{\,|V''| \mid V'' \in S'\,\} < |V'|$ **then**
                $S' \leftarrow \{V'\}$
            else if $|V'| = \max\{\,|V''| \mid V'' \in S'\,\}$ **then**
                $S' \leftarrow S' \cup \{V'\}$
    if $S' \neq \{\langle\rangle\}$ **then**
        $S \leftarrow S'$
    $d \leftarrow d + 1$
solution $\leftarrow \bot$
for all $V \in S$ **do**
    solution $\leftarrow$ *solution* $\vee \left(\mu \wedge \bigwedge_{i=1}^{k} A_{d(V_i),f(V_i)}\right)$
return *solution*

Proof. Let S be the set of prefix vectors with the following property.

$$\mathrm{mod}(\triangle_{\mu}^{\mathrm{Gmin}}(E)) = \bigcup_{V \in S} \mathrm{mod}(\mu) \cap \bigcap_{i=1}^{k} \mathrm{mod}\left(D^{d(V_i)}\left(\phi_{f(V_i)}\right)\right)$$

It should be clear that for any two $V, V' \in S$, V and V' agree with each other. We prove that for any n with $0 \leq n \leq p$, if $V \in W_n$ and $V' \in S$ then V and V' agree with each other, i.e., $d(V_i) = d(V'_i)$ for all $1 \leq i \leq \min\{|V|, |V'|\}$.

Let V' be some prefix vector in S. If $W_0 \neq \emptyset$ then by construction, if $V \in W_0$ then V and V' agree. To see why this is true, consider that any member of W_0 can be extended to one of length k by adding all the missing formulae dilated at the maximum degree p (ensuring consistency). In other words, V can be extended to:

$$\langle (f(V_1), 0), \ldots, (f(V_{|V|}), 0), (i_1, p), \ldots, (i_{k-|V|}, p) \rangle$$

for an appropriate sequence of numbers $i_1, \ldots, i_{k-|V|}$. So, if V' has a smaller number of leading zeros as dilation degrees then we reach a contradiction as the distance vector corresponding to V' cannot be lexicographically minimal. That V' cannot have more leading zero dilation degrees follows from the requirement that W_0 contains the largest possible consistent conjunctions of profile formulae. Now assume that for some n with $0 \leq n \leq p$, all $V \in W_n$ agree with V'. Then

again, by construction, any $V \in W_{n+1}$ will agree with V' by the same logic used for the case of W_0.

Therefore, since any $V \in W_n$ agrees with any $V' \in S$, because of the fact that we always maximally extend prefix vectors with dilations of profile formulae, then for some j with $0 \leq j \leq p$ the length of the prefix vectors in W_j is k and W_j is equal to S. □

In the worst case, Algorithm 6 will perform $\mathcal{O}(k \cdot p)$ dilations. Computing `maxcons`(V, A_d) is potentially expensive, as it may produce a number of vectors that is exponential in the number of profile formulae missing from V. If we assume that for each call up to $\mathcal{O}(2^k)$ vectors may be produced, and considering that Algorithm 6 will terminate after at most $p+1$ iterations, we obtain a naïve upper bound for the number of conjunctions of $\mathcal{O}\left(2^{k\cdot(p+1)}\right)$.

4 Experimental evaluation

As described in Section 2.1.5, all the merging operators we examined have a query-answer decision problem whose worst-case complexity is intractable. Our algorithms use BDDs to represent formulae, and therefore, can be more efficient but still suffer from intractable worst-case complexity. Given the difficulty of theoretically assessing the average-case complexity, we present here an experimental evaluation.

In order to implement the algorithms presented, we used a freely-available software library for the creation and manipulation of BDDs called `Buddy` [27]. There are language bindings for several languages including Java, C++, Ruby and others. Our implementation was written in C++ and can be downloaded from `http://www.cs.ucl.ac.uk/n.gkorogiannis/source/dilations.tgz`.

The general setup of an experimental valuation of our implementation involves generating random profiles and merging them, while keeping track of how much time and memory the algorithms use. Note that we do not measure the number of nodes in the resulting BDDs because this can, in general, be smaller than the actual number of BDD nodes produced and then freed during the running of the algorithms. Instead we record the total number of BDD nodes ever produced.

Of course, it is very important to choose the profile generation procedure appropriately so that results are not skewed by assumptions implicit in that procedure. We describe and motivate this procedure in the next section.

4.1 Test-case generation

There is, generally, no accepted method for experimentally evaluating knowledge merging algorithms, neither is there a library of standard knowledge merging problems. As such, we need to propose a framework for empirically evaluating the proposed algorithms. It is not easy to circumscribe in an absolute manner a class of profiles that are generally "interesting". To overcome this, we borrow

intuitions and methods from the field of research concerned with the analysis of the complexity of the decision problem of propositional satisfiability (SAT). There is a wealth of experimental results concerning SAT and a long-standing line of experimental research around them [14, 30, 1]. Evaluation studies in SAT rely on a library of standard problems, as well as on methods for generating random problem instances. Since, to the authors' knowledge, the former does not exist in the field of knowledge merging, we will attempt the latter. The only other experimental assessment of algorithms for merging operators we know of ([20]) also uses random knowledge bases as a way of evaluating performance.

In order to keep the volume of the generated data presentable, we opted for the most commonly used type of formulae in SAT. This means that for any $\psi \in \{\phi_1, \ldots, \phi_k, \mu\}$, ψ will be a formula in CNF, consisting of 3-literal clauses (3CNF),[5] or in symbols, $\psi = \bigwedge_{i=1}^{j} l_{a_i} \vee l_{b_i} \vee l_{c_i}$, for sequences a_i, b_i and c_i. A random number generator is used for deciding which atoms will appear in the formula (uniformly distributed) and whether they will be negated or not (with equal probability). We only consider profiles with formulae of the same length (i.e., same number of clauses), so as to simplify the empirical methodology.

One benchmarking approach is to vary the number of variables or clauses and run each merging operator, plotting a curve of the time required against the number of clauses or variables. However, results on the complexity of SAT indicate that a better choice for the x-axis is the ratio of clauses to variables, a measure called *density*, which we will denote by r. Indeed, using this ratio as the x-axis, the time maxima of the plotted curves for SAT appear at the same ratio regardless of the number of variables. This choice turns out to be useful in the case of merging, as the results below show. Another consideration is the consistency of the profiles and of their formulae. An obvious first decision is to only consider profiles with consistent formulae, i.e., for all i, $\phi_i \nvdash \bot$ and $\mu \nvdash \bot$. In addition, we restrict our attention to inconsistent profiles.

The choice of boundaries of the tested density range requires explanation. A well-known result from SAT indicates that there is an almost fixed density where a phase transition occurs with respect to the satisfiability of the random formulae produced: most formulae produced with densities under this threshold will be satisfiable and most formulae over this threshold will be unsatisfiable. This threshold r_{SAT} seems to be around 4.5, where the ratio of satisfiable/unsatisfiable formulae is approximately 1 [30]. As the only way to produce consistent formulae is to generate them and then test for consistency, working with densities close to or over this threshold is intractable, as a large number of generated formulae will have to be discarded. Therefore the densities we examine are lower than that threshold, with regards to the individual profile formulae.

In addition to this upper limit in the density range, there is a lower one related to the requirement that profiles are inconsistent. Since we require that $\mu \wedge \bigwedge_{i=1}^{k} \phi_i \vdash \bot$ (and, thus, there is a point in merging the profile) and this conjunction is a formula in 3CNF, its total number of clauses divided by the number

[5] Conversion of an arbitrary formula in CNF to 3CNF can be done in polynomial time.

of variables will determine the probability of it being inconsistent. Again, so as to avoid prohibitive profile generation times, we restrict our tests to densities above the threshold; this means that for p variables, if each formula has C clauses then we want $r_{\mathrm{SAT}} < \frac{(k+1)\cdot C}{p}$, which means that $r = \frac{C}{p} > \frac{r_{\mathrm{SAT}}}{k+1}$. We took this as the lower end of the density ranges we examined.

4.2 Time and space versus density

The first set of experiments we conducted was to fix the number of profile formulae to 7, vary the number of atoms in the range of 10, 15 and 20 and the density between the two thresholds. For each point, i.e., each choice of p and r, we generate 500 random profiles that fit the criteria outlined earlier. Each profile is then merged using the four presented algorithms while recording how much memory (in terms of BDD nodes produced) and how much processing time the merge requires. We then plot the mean time and mean number of BDD nodes produced against density. The densities we examine range between 0.9 and 3.9, limits calculated according to the considerations laid out in the previous section (including a margin of about 10%, as generating appropriate profiles in the vicinity of the thresholds is intractable). The results can be seen in Figure 4. A number of observations can be made.

The behaviour of $\triangle_{\mu}^{\mathrm{Max}}$ is interesting in that its time and space complexity exhibits a maximum at a density around $2.2 \leq r \leq 2.6$. We believe there are two opposite factors at work here: the mean number of dilations required before a consistent conjunction is formed is inversely proportional to the number of models of each profile formula, which itself is inversely proportional to the number of clauses. Therefore, increased density means more dilations. At the same time, the cost of computing a dilation seems to drop at densities over 3. We conjecture that this is related to the form of the BDDs representing the profile: at the limit, each formula will have a single model, and the BDD for such a formula will be linearly arranged and with size linear to p, therefore making dilation significantly easier. Indeed, the mean size of random profiles in BDD nodes (not shown here) peaks at around density 1.4 and falls exponentially for greater densities.

The algorithm for $\triangle_{\mu}^{\mathrm{Max}}$ seems to be the fastest and least space-consuming of all four algorithms and this may have to do with the simplicity of the merging operator. Indeed, in the worst case, at $p = 20$, $2.2 \leq r \leq 2.6$, the mean execution time is around 1 second and the mean number of nodes is around 6×10^5 (each profile consists of 7 formulae plus the integrity constraints, of around 52 clauses each and with 20 atoms in the language). Also, a nearly linear relation between time and space complexity can be observed. This is mainly related to the fact that most of the time spent by $\triangle_{\mu}^{\mathrm{Max}}$ is spent in BDD manipulations. BDD algorithms are known to have a space complexity which is almost linear to the time complexity, when caching effects are relatively minor (e.g., when arguments to successive calls to `apply` do not repeat).

In contrast, $\triangle_{\mu}^{\Sigma}$ seems to be the worst algorithm in terms of both time and space complexity. We believe that this is due to two factors. First, if we assume

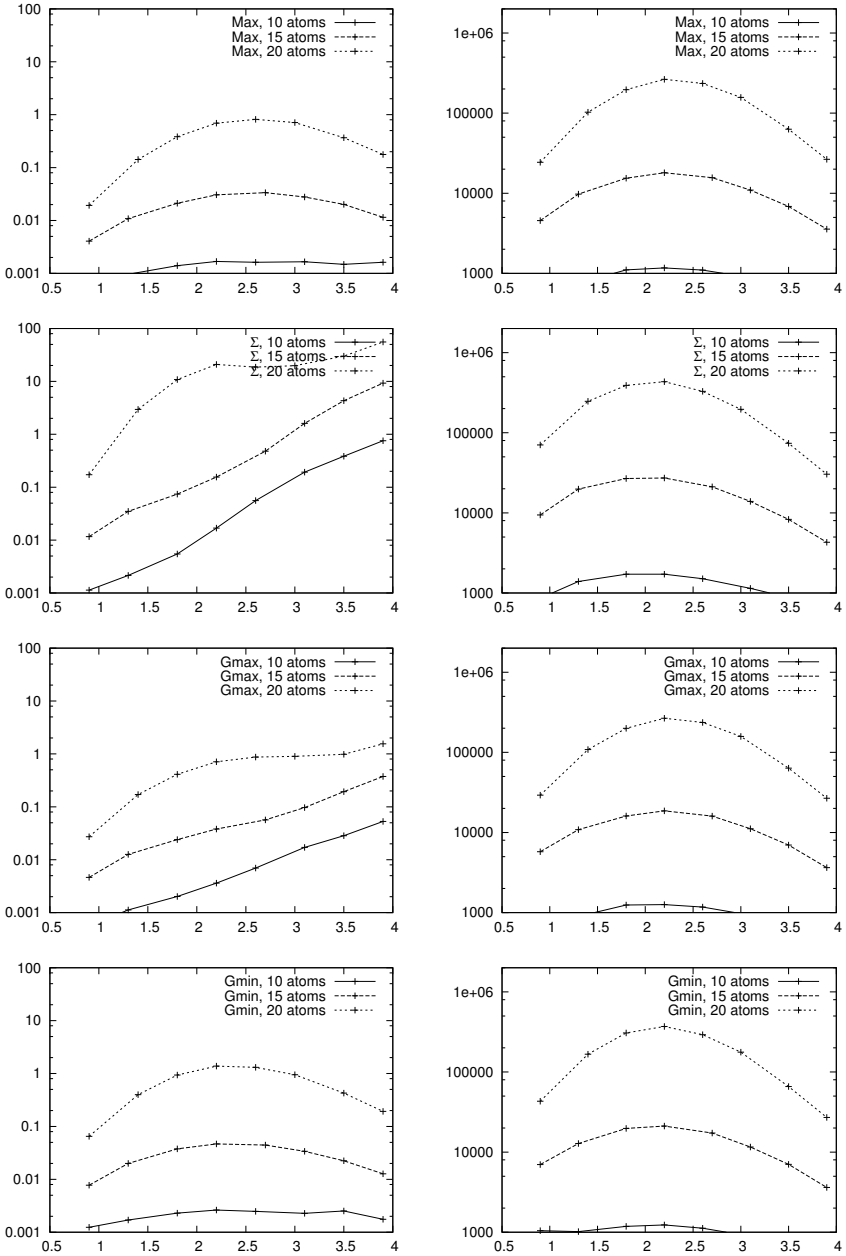

Figure 4: Left, mean time (in seconds) for merging against density. Right, mean number of BDD nodes for merging against density.

that the distribution of aggregated distances enumerated by Algorithm 3 peaks for some d with $p \leq d$, then it follows that many runs of the algorithm will have to generate $k \cdot p$ dilations before continuing to check the distances between $p+1$ and $k \cdot p$. This makes our algorithm for $\triangle_\mu^\Sigma$ at least as time and space consuming as $\triangle_\mu^{\mathrm{Max}}$.

In addition, the nearly linear relation between time and space required is lost. As density increases, so does the mean number of dilations as explained earlier, and similarly so does the mean maximum aggregate distance. Therefore, the number of compositions of the maximum distance increases as well ($\binom{d+k-1}{k-1}$ for distance d), explaining the exponential increase in time required. At the same time, if the number of models in each formula is sufficiently small then most conjunctions will be inconsistent which means that the extra space required will be relatively small (especially due to caching effects) and, therefore, the space complexity plot will no longer be similar to that of the time complexity.

The $\triangle_\mu^{\mathrm{Gmax}}$ algorithm exhibits space complexity similar to $\triangle_\mu^{\mathrm{Max}}$. The enumeration of distance vectors has the property that all vectors with maximum element d can be produced and checked before the ones with maximum element $d+1$. This means that we can interleave the checking of conjunctions and the production of dilations in a better way than for $\triangle_\mu^\Sigma$, as explained in the previous paragraphs. On the other hand, the time complexity of $\triangle_\mu^{\mathrm{Gmax}}$ lies between those of $\triangle_\mu^{\mathrm{Max}}$ and $\triangle_\mu^\Sigma$, and we believe this is again due to the fact that the number of vectors to be checked increases significantly according to the maximum element. In fact, the growth function is the same as with $\triangle_\mu^\Sigma$ since the number of ordered lists of size k with maximum element d is $\binom{d+k-1}{k-1}$. Still, $\triangle_\mu^{\mathrm{Gmax}}$ has a better time complexity than $\triangle_\mu^\Sigma$ and we feel that this is due to the optimisation we mentioned in Section 3, which is that if the conjunction of dilations of degree d is inconsistent then all distance vectors with maximum element d can be skipped as they will by necessity be inconsistent.

The algorithm $\triangle_\mu^{\mathrm{Gmin}}$ performs well, with time and space complexity close to those of $\triangle_\mu^{\mathrm{Max}}$. It is easy to see that the number of dilations required for $\triangle_\mu^{\mathrm{Gmin}}$ will be lower or equal to that of $\triangle_\mu^{\mathrm{Max}}$, owing to the fact that if $\mu \wedge \bigwedge_{i=1}^k D^n(\phi)$ is consistent for some n then Algorithm 6 will need to do at most $n+1$ iterations before terminating. Indeed, the average number of profile dilations (not shown here) peaks at around 10 for this experiment, a value close to that of $\triangle_\mu^{\mathrm{Max}}$.

In addition, we have reasons to believe that the actual number of conjunctions tested is exponential in k but not in the density r. In each iteration of the algorithm, the number of vectors produced is proportional to the maximum number of vectors produced in the next iteration, since each vector will need to be tested as to whether it can be consistently extended. In addition, the algorithm for `maxcons` only has to examine conjunctions of dilations for formulae that do not already appear in the vector provided as argument. These two facts combined mean that if in each iteration the maximum number of maximal consistent subsets is produced (of length $\lfloor \frac{n}{2} \rfloor$, and of number $\binom{n}{\lfloor \frac{n}{2} \rfloor} = \mathcal{O}(2^n)$ for a set of size n) then the total maximum number of vectors created by the

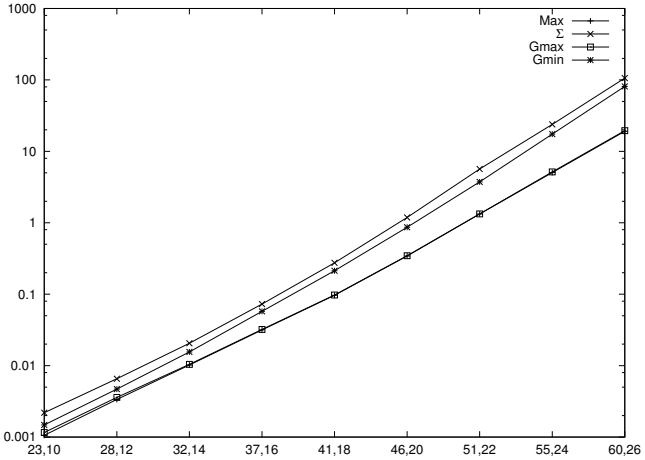

Figure 5: Mean time for merging (in seconds) against (clauses,atoms), for constant density $r = 2.3$ and $k = 5$.

algorithm will be $2^k \cdot 2^{\frac{k}{2}} \cdot 2^{\frac{k}{4}} \cdot \ldots \cdot 2^0 = \mathcal{O}(2^{2k})$, and this number will also be an upper bound for the number of conjunctions. While this bound is of exponential order, it is exponential in k and but not in r.

4.3 Time versus number of atoms

As noted in the previous section, $\triangle_\mu^{\mathrm{Max}}$ and $\triangle_\mu^{\mathrm{Gmin}}$ exhibit their worst performance for densities around 2.3. The algorithms $\triangle_\mu^{\Sigma}$ and $\triangle_\mu^{\mathrm{Gmax}}$ also present the same maximum in their space complexity. We believe that low densities represent a much more interesting area for applications; the typical examples and uses of knowledge merging are concerned with formulae that have more than a few models. Therefore, in order to shed more light on how the algorithms behave for increasing numbers of variables we conducted a second set of experiments, comprising of a sequence of points for a fixed number of knowledge bases (5) and density (2.3), for increasing number of atoms (from 10 to 26 in steps of 2). At each point, 1000 random profiles were generated and merged, recording the memory and processing time consumed.

The resulting graph can be seen in Figure 5. It is easy to see that the dependence of time complexity to the number of atoms is exponential (the space complexity graph is very similar). We conjecture that this dependence is primarily rooted on the complexity characteristics of BDDs representing the profiles to be merged. We have already indicated that the worst-case space complexity for a BDD is exponential. Moreover, there is the probabilistic result stating that the mean size of the BDD for a random boolean function will be equal to that of the worst case up to terms of lower order (i.e., sub-exponential) [16]. Indeed,

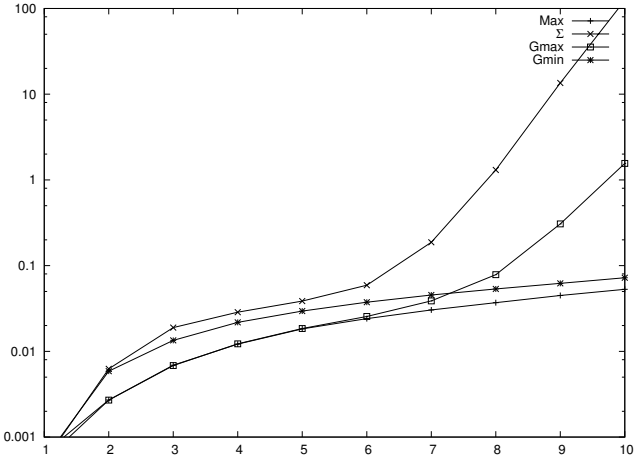

Figure 6: Mean time for merging (in seconds) against number of knowledge bases for 15 atoms and 35 clauses.

this is a special case of a more general characteristic of compilation languages: there is commonly a trade-off phenomenon between the representational succinctness of a language and the tractability of the algorithms for manipulating structures in that language.

4.4 Time versus number of formulae

Finally, we explored the behaviour of these algorithms while varying the number of knowledge bases. Again, we chose the density 2.3, where our algorithms seem to exhibit their worst-case space complexity, for 15 atoms and 35 clauses. For each number from 1 to 10, we tested 1000 profiles of that size. The results can be seen in Figure 6. The algorithms for $\triangle_\mu^{\mathrm{Max}}$ and $\triangle_\mu^{\mathrm{Gmin}}$ perform well, in that the observed behaviour does not seems to be exponential. The performance of $\triangle_\mu^{\mathrm{Max}}$ is due to the fact that the increase in the number of formulae does not exponentially increase the number of conjunctions to be tested. The clearly exponential behaviour of $\triangle_\mu^{\Sigma}$ and $\triangle_\mu^{\mathrm{Gmax}}$ can be directly attributed to the exponential growth of the number of conjunctions tested. The algorithm for $\triangle_\mu^{\mathrm{Gmin}}$ performs well despite the fact that the number of conjunctions tested is exponential to profile size, but only because the sizes tested are relatively small. Indeed, plotting the number of conjunctions tested versus profile size for $\triangle_\mu^{\mathrm{Gmin}}$ yields an exponential graph (not shown here) that does not, however, reach prohibitive values for the range of profile sizes tested.

5 Conclusions

In this paper we have presented one way of implementing semantic knowledge merging operators from the literature using the notion of dilations and the data structure known as binary decision diagram. We have experimentally assessed the viability of these algorithms when run on randomly generated profiles. Merging operators are intractable in terms of abstract complexity and our algorithms show definite signs of exponential complexity in the number of propositional atoms in the language. However, with a computer of modest specifications (Core 2 Duo, 1.8 GHz) we were able to merge random profiles of 5 profile formulae (plus integrity constraints), 46 clauses per formula with 20 propositional atoms, while requiring no more than 10 seconds and 20Mb of memory. These results therefore show that the theoretically appealling semantic merging operators by Konieczny et al. can be used for some practical AI problems.

Hue et al [20], describe an implementation for belief fusion which covers two operators, one of which, $\triangle_\mu^\Sigma$, we also examine in this paper. They cite preliminary experimental results based on a scheme of generating profiles which is similar to ours. Their implementation, however, does not cover $\triangle_\mu^{\text{Max}}$, $\triangle_\mu^{\text{Gmax}}$ or $\triangle_\mu^{\text{Gmin}}$, and does not support the use of integrity constraints. Moreover, the numbers of clauses, variables and knowledge bases they experimentally test are very different to those we examined in the previous section, as they reside in areas where the complexity for SAT is very low (density equal to 0.3). Hence, our work complements existing results by studying more operators and in test-cases that are substantially difficult to compute.

There exist several avenues for further research. First, several improvements on our algorithms can be attempted and studied. For example, the way BDDs for conjunctions are computed can be changed so that the arguments are ordered by their number of models; this, on average will cause inconsistent conjunctions to be computed faster and therefore consume less memory. Another possibility is to explore the effect of using BDD variable re-ordering heuristics. Although such methods are standard in BDD packages, our preliminary attempts to use them proved unrewarding, possibly because of the randomness of our profiles; on the whole, re-ordering heuristics took a lot more time than they saved, when applied naïvely. Also, the structure of our algorithms for $\triangle_\mu^\Sigma$ and $\triangle_\mu^{\text{Gmax}}$ can, perhaps, be improved so that generation of dilations is delayed as much as possible, and a more intelligent strategy is employed for checking permutations or compositions. Finally, a more involved line of research could look into query-answering algorithms using our formulations based on dilations but with Quantified Boolean Formulae satisfiability solvers instead of BDDs.

References

[1] A. San Miguel Aguirre and M. Y. Vardi. Random 3-SAT and BDDs: The plot thickens further. In *Proceedings of the 7th International Conference*

on Principles and Practice of Constraint Programming, CP 2001, volume 2239 of *Lecture Notes in Computer Science*, pages 121–136, 2001.

[2] C. E. Alchourrón, P. Gärdenfors, and D. Makinson. On the logic of theory change: Partial meet contraction and revision functions. *Journal of Symbolic Logic*, 50(2):510–530, 1985.

[3] H. R. Andersen. An introduction to binary decision diagrams. `http://www.itu.dk/~hra/notes-index.html`, 1997. Lecture notes.

[4] J. Arndt. FXT, a library of algorithms. Published online at `http://www.jjj.de/fxt/#fxtbook`.

[5] O. Bailleux and P. Marquis. Some computational aspects of DISTANCE-SAT. *Journal of Automated Reasononing*, 37(4):231–260, 2006.

[6] C. Baral, S. Kraus, and J. Minker. Combining multiple knowledge bases. *IEEE Transactions on Knowledge and Data Engineering*, 3(2):208–220, 1991.

[7] C. Baral, S. Kraus, J. Minker, and V. S. Subrahmanian. Combining knowledge bases consisting of first-order theories. *Computational Intelligence*, 8:45–71, 1992.

[8] I. Bloch and J. Lang. Towards Mathematical Morpho-Logics. In *Technologies for Constructing Intelligent Systems*, volume 2, pages 367–380. Springer-Verlag, 2002.

[9] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 35(8):677–691, 1986.

[10] M. Dalal. Investigations into a theory of knowledge base revision. In *Proceedings of the Seventh National Conference on Artificial Intelligence, AAAI'88*, volume 2, pages 475–479, 1988.

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

[12] T. Eiter and G. Gottlob. On the complexity of propositional knowledge base revision, updates, and counterfactuals. *Artificial Intelligence*, 57(2-3):227–270, 1992.

[13] P. Everaere, S. Konieczny, and P. Marquis. Quota and Gmin merging operators. In *Nineteenth International Joint Conference on Artificial Intelligence, IJCAI'05*, pages 424–429, 2005.

[14] I. P. Gent and T. Walsh. Easy problems are sometimes hard. *Artificial Intelligence*, 70(1-2):335–345, 1994.

[15] N. Gorogiannis and M. Ryan. Implementation of belief change operators using BDDs. *Studia Logica*, 70(1):131–156, 2002.

[16] C. Gröpl, H. J. Prömel, and A. Srivastav. Ordered binary decision diagrams and the Shannon effect. *Discrete Applied Mathematics*, 142(1-3):67–85, 2004.

[17] A. Grove. Two modellings for theory change. *The Journal of Philosophical Logic*, 17:157–170, 1988.

[18] R.W. Hamming. Error detecting and error correcting codes. *Bell System Technical Journal*, 29:147–160, April 1950.

[19] T. Horiyama and T. Ibaraki. Reasoning with ordered binary decision diagrams. *Discrete Applied Mathematics*, 142(1-3):151–163, 2004.

[20] J. Hue, O. Papini, and E. Würbel. Syntactic propositional belief bases fusion with removed sets. In *Symbolic and Quantitative Approaches to Reasoning with Uncertainty, 9th European Conference, ECSQARU 2007*, volume 4724 of *Lecture Notes in Computer Science*, pages 66–77. Springer, 2007.

[21] S. Konieczny, J. Lang, and P. Marquis. DA^2 merging operators. *Artificial Intelligence*, 157(1-2):49–79, 2004.

[22] S. Konieczny and R. Pino Pérez. On the logic of merging. In *Sixth International Conference on Principles of Knowledge Representation and Reasoning, KR'98*, pages 488–498, 1998.

[23] S. Konieczny and R. Pino Pérez. Merging with integrity constraints. In *Proceedings of the European Conference on Symbolic and Quantitative Approaches to Reasoning and Uncertainty, ECSQARU'99*, volume 1638 of *Lecture Notes in Computer Science*, pages 233–244. Springer-Verlag, 1999.

[24] S. Konieczny and R. Pino Pérez. Merging information under constraints: A logical framework. *Journal of Logic and Computation*, 12(5):773–808, 2002.

[25] P. Liberatore and M. Schaerf. Arbitration (or how to merge knowledge bases). *IEEE Transactions on Knowledge and Data Engineering*, 10(1):76–90, 1998.

[26] J. Lin and A. O. Mendelzon. Merging databases under constraints. *International Journal of Cooperative Information Systems*, 7(1):55–76, 1998.

[27] J. Lind-Nielsen. BuDDy: a binary decision diagram library. `http://sourceforge.net/projects/buddy`.

[28] J. C. Madre and O. Coudert. A logically complete reasoning maintenance system based on a logical constraint solver. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence, IJCAI'91*, volume 1, pages 294–299. Morgan Kaufmann, 1991.

[29] P. Z. Revesz. On the semantics of arbitration. *Journal of Algebra and Computation*, 7(2):133–160, 1997.

[30] B. Selman, D. G. Mitchell, and H. J. Levesque. Generating hard satisfiability problems. *Artificial Intelligence*, 81(1-2):17–29, 1996.

[31] E. Würbel, R. Jeansoulin, and O. Papini. Spatial information revision: A comparison between 3 approaches. In *Proceedings of the 6th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty, ECSQARU 2001*, volume 2143 of *Lecture Notes in Computer Science*, pages 454–465. Springer, 2001.