Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence

A Knowledge Compilation Map for Ordered Real-Valued Decision Diagrams

Hélène Fargier[1] and **Pierre Marquis**[2] and **Alexandre Niveau**[3] and **Nicolas Schmidt**[1,2]
[1] IRIT-CNRS, Univ. Paul Sabatier, Toulouse, France
[2] CRIL-CNRS, Univ. Artois, Lens, France
[3] GREYC-CNRS, Univ. Caen, France

Abstract

Valued decision diagrams (VDDs) are data structures that represent functions mapping variable-value assignments to non-negative real numbers. They prove useful to compile cost functions, utility functions, or probability distributions. While the complexity of some queries (notably optimization) and transformations (notably conditioning) on VDD languages has been known for some time, there remain many significant queries and transformations, such as the various kinds of cuts, marginalizations, and combinations, the complexity of which has not been identified so far. This paper contributes to filling this gap and completing previous results about the time and space efficiency of VDD languages, thus leading to a knowledge compilation map for real-valued functions. Our results show that many tasks that are hard on valued CSPs are actually tractable on VDDs.

1 Introduction

Valued decision diagrams (VDDs) are data structures that represent multivariate functions f having a set $\mathcal{V}$ of valuations (often a subset of $\mathbb{R}^+$) as codomain; such functions are typically cost functions, utility functions, or probability distributions, and as such, are considered in a number of AI applications. Among the various tasks of interest when dealing with such functions are *optimization* queries: find an assignment of the variables leading to an optimal valuation; find a value of a given variable that can be extended to an optimal assignment; etc. Optimization queries are particularly valuable when combined with the *conditioning* transformation, which derives a representation of a restriction of f obtained by assigning some of its variables. For instance, the following task can be handled by combining optimization with conditioning: in configuration problems, when f represents a cost function mapping each assignment (say, a car) to its price, output the cheapest car with seven seats; or, when f represents a probability distribution linking diseases to symptoms, return the most probable explanation (the most probable disease given a set of symptoms).

Many other data structures have been defined for representing such multivariate functions f, valued CSPs (Schiex, Fargier, and Verfaillie 1995), GAI nets (Bacchus and Grove 1995), and Bayes nets (Pearl 1989) being among the best-known. However, they are not adapted to the aforementioned requests when guaranteed response times are required (as is the case in Web-based applications): optimization is indeed NP-hard on CSPs, GAI nets, and Bayes nets.

Contrastingly, optimization is a tractable query on VDDs; and conditioning is tractable as well. This explains why several families of VDDs have been defined and studied in the past twenty years. These include the language ADD of algebraic decision diagrams (Bahar et al. 1993), the language AADD of affine algebraic decision diagrams (Tafertshofer and Pedram 1997; Sanner and McAllester 2005), and the language SLDD of semiring-labeled decision diagrams (Wilson 2005). Actually, SLDD is itself a family of languages $\texttt{SLDD}_{\otimes}$, parameterized by a binary operator $\otimes$, yielding in particular $\texttt{SLDD}_{+}$, or equivalently EVBDD (Lai and Sastry 1992; Lai, Pedram, and Vrudhula 1996; Amilhastre, Fargier, and Marquis 2002), when $\otimes$ is $+$, and $\texttt{SLDD}_{\times}$ when $\otimes$ is $\times$.

There nevertheless exist many queries and transformations of interest that do not amount to a combination of conditioning and optimization. Consider for instance the following request: "tell me whether among the 'cheap' cars (say, with a price lower than 10,000 euros) of this given type, there is one that has seven seats". It requires one to focus on the set of 'cheap' cars, which is neither optimization nor conditioning. Similarly, some transformations, such as projection on variables of interest, or its dual, variable elimination (e.g., forgetting or marginalization), also being of tremendous value (e.g., to solve the "posterior marginal" problem (PM) in Bayes nets), are not reducible to optimization and conditioning. This is also the case of combination transformations (a.k.a. 'apply' operations), which given the representations of two functions f and g, compute a representation of $f \odot g$, $\odot$ being an associative and commutative operator on $\mathcal{V}$ (e.g., addition or product when $\mathcal{V} = \mathbb{R}^+$). Such transformations are very important, if only for incrementally generating representations in a bottom-up way.

The knowledge compilation (KC) map (Darwiche and Marquis 2002) identifies the complexity of requests (i.e., queries and transformations) over many propositional languages, as well as their relative succinctness. However, it has been drawn for the specific case of Boolean functions; although it has been extended in several directions, as of yet no such map exists for the VDD languages. It has been

shown that AADD is strictly more succinct than ADD (Sanner and McAllester 2005), and the succinctness picture has later been completed (Fargier, Marquis, and Schmidt 2013): AADD is strictly more succinct than both $\texttt{SLDD}_+$ and $\texttt{SLDD}_\times$, and both $\texttt{SLDD}_+$ and $\texttt{SLDD}_\times$ are in turn strictly more succinct than ADD. To complete the map, one now needs to provide the set of requests that each language satisfies; in order words, for each request, and for each language among ADD, $\texttt{SLDD}_+$, $\texttt{SLDD}_\times$, and AADD, to either find a polynomial-time algorithm computing the request, or prove that no such algorithm exists unless $\mathsf{P} = \mathsf{NP}$.

This is the main goal of this paper, which is organized as follows. The next section presents the family of VDD languages, and the following one formally defines the queries and transformations used. Then complexity results are presented,[1] that establish whether each VDD language satisfies each query or transformation. This paves the way for a KC map for functions that are not essentially Boolean.

2 Valued Decision Diagrams

Preliminaries. Given a finite set $\mathcal{X} = \{x_1, \ldots, x_n\}$ of variables, each x_i ranging over a finite domain D_{x_i}, and any set $X \subseteq \mathcal{X}$, $\vec{x} = \{ \langle x_i, d_i \rangle \mid x_i \in X, d_i \in D_{x_i} \}$ denotes an assignment of the variables from X; D_X is the set of all such assignments (the Cartesian product of the domains of the variables in X). The concatenation of two assignments $\vec{x}$ and $\vec{y}$ of two disjoint subsets X and Y is an assignment of $X \cup Y$ denoted $\vec{x} \cdot \vec{y}$.

We consider functions f of variables from a subset $\mathrm{Scope}(f) \subseteq \mathcal{X}$ to some set $\mathcal{V}$ (this paper often assumes $\mathcal{V} = \mathbb{R}^+$). The domain of f is denoted as $D_f = D_{\mathrm{Scope}(f)}$. For any $Z \subseteq \mathrm{Scope}(f)$, $f_{\vec{z}}$ denotes the *restriction* (or *semantic conditioning*) of f by $\vec{z}$, that is, the function on $\mathrm{Scope}(f) \setminus Z$ such that for any $\vec{x} \in D_{\mathrm{Scope}(f)\setminus Z}$, $f_{\vec{z}}(\vec{x}) = f(\vec{z} \cdot \vec{x})$.

Given a binary operator $\odot$ over $\mathcal{V}$ and two functions f and g sharing the same scope, $f \odot g$ is the function defined by $f \odot g\ (\vec{x}) = f(\vec{x}) \odot g(\vec{x})$. The $\odot$-projection of f on $Z \subseteq \mathrm{Scope}(f)$ is defined by $f^{\odot,Z}(\vec{x}) = \bigodot_{\vec{y} \in D_{\mathrm{Scope}(f)\setminus Z}} f_{\vec{y}}(\vec{x})$.

Slightly abusing notations, if X and Y are two disjoint sets of variables, $\mathrm{Scope}(f) = X$, and $\vec{x} \cdot \vec{y}$ is an assignment of $X \cup Y$, then we assume that $f(\vec{x} \cdot \vec{y}) = f(\vec{x})$; in practice, we often consider complete assignments, i.e., $\vec{x} \in D_{\mathcal{X}}$.

Given $\succeq$ a reflexive and transitive relation on $\mathcal{V}$ (i.e., a preorder), of which we denote $\sim$ the symmetric part and $\succ$ the asymmetric part, we can define for any $\gamma \in \mathcal{V}$ the following sets, referred to as "cuts":

- $CUT^{\max}(f) = \{ \vec{x}^* \mid \forall \vec{x}, \neg(f(\vec{x}) \succ f(\vec{x}^*)) \}$;
- $CUT^{\min}(f) = \{ \vec{x}^* \mid \forall \vec{x}, \neg(f(\vec{x}) \prec f(\vec{x}^*)) \}$;
- $CUT^{\succeq\gamma}(f) = \{ \vec{x} \mid f(\vec{x}) \succeq \gamma \}$, and using similar definitions, $CUT^{\preceq\gamma}(f)$ and $CUT^{\sim\gamma}(f)$.

[1] For space reasons, proofs are omitted; a full version of the paper, completed with proofs, can be found at ⟨url: https://niveau.users.greyc.fr/pub/AAAI14_FMNS.pdf⟩.

For instance, when optimizing is minimizing, $CUT^{\min}$ is the set of optimal assignments (e.g., the cheapest cars); $CUT^{\preceq\gamma}$ is the set of assignments satisfying the cut condition (e.g., the 'cheap' cars – those with a price $\preceq \gamma$).

A representation language over $\mathcal{X}$ w.r.t. a set $\mathcal{V}$ is a set of data structures equipped with an interpretation function that associates with each of them a mapping $f : D_{\mathcal{X}} \to \mathcal{V}$. This mapping is called the *semantics* of the data structure, and the data structure is a *representation* of the mapping.

Definition 2.1 (representation language; inspired from Gogic et al. (1995)). Given a valuation set $\mathcal{V}$, a *representation language* L over $\mathcal{X}$ w.r.t. $\mathcal{V}$, is a 4-tuple $\langle C_{\mathtt{L}}, \mathrm{Var}_{\mathtt{L}}, f^{\mathtt{L}}, s_{\mathtt{L}} \rangle$, where:

- $C_{\mathtt{L}}$ is a set of data structures α (also referred to as L representations or "formulæ"),
- $\mathrm{Var}_{\mathtt{L}} : C_{\mathtt{L}} \to 2^{\mathcal{X}}$ is a scope function associating with each L representation the subset of $\mathcal{X}$ it depends on,
- $f^{\mathtt{L}}$ is an interpretation function associating with each L representation α a mapping $f^{\mathtt{L}}_{\alpha}$ from the set of all assignments over $\mathrm{Var}_{\mathtt{L}}(\alpha)$ to $\mathcal{V}$,
- $s_{\mathtt{L}}$ is a size function from $C_{\mathtt{L}}$ to $\mathbb{N}$ that provides the size of any L representation.

Two formulæ (possibly from different languages) are equivalent iff they have the same scope and semantics.

Valued decision diagrams. In the following, we consider representation languages based on data structures called *valued decision diagrams*: such diagrams target the representation of $\mathcal{V}$-valued functions by allowing their arcs and nodes to be labeled with values in some set $\mathcal{E}$ (generally, $\mathcal{E} = \mathcal{V}$, but as we will see, it is not always the case).

Definition 2.2 (valued decision diagram). A *valued decision diagram* (VDD) over $\mathcal{X}$ w.r.t. $\mathcal{E}$ is a finite rooted DAG α, of which every internal node N is labeled with a variable $x \in \mathcal{X}$ and has a set $\mathrm{Out}(N)$ of $|D_x|$ outgoing arcs, each arc $a \in \mathrm{Out}(N)$ being labeled with a distinct value $v(a) \in D_x$. Arcs and leaves can be labeled with elements of $\mathcal{E}$: $\varphi(a)$ (resp. $\varphi(L)$) denotes the label of arc a (resp. of leaf L). The size of a decision diagram α, denoted $|\alpha|$, is the size of the graph (its number of nodes and arcs) plus the sizes of all labels in it. VDD is the set of all valued decision diagrams.

In the following, we assume that the decision diagrams are *ordered*, i.e., a total, strict ordering $\rhd$ over $\mathcal{X}$ is chosen, and for each path from the root to a leaf in a VDD α, the associated sequence of internal node labels is required to be compatible w.r.t. this variable ordering (such diagrams are thus read-once). A path from the root to a leaf of α represents a (possibly partial) assignment of $\mathcal{X}$. Note that the structure is deterministic: an assignment $\vec{x}$ of $\mathcal{X}$ corresponds to at most one path $p_\alpha(\vec{x})$ in α.

A VDD α is *reduced* iff it does not contain any (distinct) isomorphic nodes.[2] A cache mechanism can be used

[2] Nodes N and M are isomorphic if they are labeled with the same variable x, and there exists a bijection B from $\mathrm{Out}(N)$ onto $\mathrm{Out}(M)$ such that $\forall a \in \mathrm{Out}(N)$, a and $B(a)$ have the same end node, share the same value of D_x, and $\varphi(a) = \varphi(B(a))$.

to merge isomorphic nodes on the fly; this is why we assume in the following that the diagrams are reduced.

VDD languages. ADD, SLDD, and AADD are VDD languages, i.e., subsets of VDD. Each of them restricts the type of diagrams used (e.g., ADD allows $\mathcal{E}$-labels on leaves only), and defines how the diagram is to be interpreted. ADD, SLDD, and AADD thus differ syntactically (in the way diagrams are labeled) and semantically (in the way they are interpreted).

Definition 2.3 (ADD). ADD is the 4-tuple $\langle C_{\texttt{ADD}}, \mathrm{Var}_{\texttt{ADD}}, f^{\texttt{ADD}}, s_{\texttt{ADD}}\rangle$ where $C_{\texttt{ADD}}$ is the set of ordered VDDs α over $\mathcal{X}$ such that leaves are labeled with elements of $\mathcal{E} = \mathcal{V}$ (in general, $\mathcal{E} = \mathcal{V} = \mathbb{R}^+$), the arcs are not labeled, and $f^{\texttt{ADD}}$ is defined inductively as follows, for every assignment $\vec{x}$:

- if α is a leaf node L, then $f^{\texttt{ADD}}_\alpha(\vec{x}) = \varphi(L)$,
- otherwise, denoting N the root of α, $x \in \mathcal{X}$ its label, $d \in D_x$ such that $\langle x, d\rangle \in \vec{x}$, $a = \langle N, M\rangle$ the arc such that $v(a) = d$, and β the ADD formula rooted at node M in α, then $f^{\texttt{ADD}}_\alpha(\vec{x}) = f^{\texttt{ADD}}_\beta(\vec{x})$.

In the AADD framework of Sanner and McAllester (2005), the co-domain of the represented functions is $\mathcal{V} = \mathbb{R}^+$, but only one (unlabeled) leaf is allowed and the arcs are labeled with pairs of values from $\mathbb{R}^+$ (i.e., $\mathcal{E} = \mathbb{R}^+ \times \mathbb{R}^+ \neq \mathcal{V}$).

Definition 2.4 (AADD). AADD is the 4-tuple $\langle C_{\texttt{AADD}}, \mathrm{Var}_{\texttt{AADD}}, f^{\texttt{AADD}}, s_{\texttt{AADD}}\rangle$ where $C_{\texttt{AADD}}$ is the set of ordered VDDs α over $\mathcal{X}$ with a unique leaf L, and the arcs of which are labeled with pairs $\langle q, f\rangle$ in $\mathbb{R}^+ \times \mathbb{R}^+$. For normalization purposes, the root of α is also equipped with a pair $\langle q_0, f_0\rangle$ from $\mathbb{R}^+ \times \mathbb{R}^+$ (the "offset"). The semantics of the resulting VDD is given by, for every assignment $\vec{x}$, $f^{\texttt{AADD}}_\alpha(\vec{x}) = q_0 + (f_0 \times g^{\texttt{AADD}}_\alpha(\vec{x}))$, where $g^{\texttt{AADD}}_\alpha$ is defined inductively as follows:

- if α is the leaf node L, then $g^{\texttt{AADD}}_\alpha(\vec{x}) = 0$,
- otherwise, denoting N the root of α, $x \in \mathcal{X}$ its label, $d \in D_x$ such that $\langle x, d\rangle \in \vec{x}$, $a = \langle N, M\rangle$ the arc such that $v(a) = d$, $\varphi(a) = \langle q_a, f_a\rangle$, and β the formula rooted at node M in α, then $g^{\texttt{AADD}}_\alpha(\vec{x}) = q_a + (f_a \times g^{\texttt{AADD}}_\beta(\vec{x}))$.

The AADD framework is equipped with a normalization condition that makes (reduced) ordered AADD formulæ canonical. Canonicity is important because it ensures a unique representation for each subformula, which is a key for efficiently recognizing and caching them.

In the $\texttt{SLDD}_\otimes$ framework (Wilson 2005),[3] arcs (but not leaves) are labeled with elements in $\mathcal{E} = \mathcal{V}$, and $\langle \mathcal{E}, \otimes, 1_\otimes\rangle$ is assumed to be a commutative monoid: $f^{\texttt{SLDD}_\otimes}_\alpha(\vec{x})$ is the aggregation by $\otimes$ of the labels of the arcs along $p_\alpha(\vec{x})$.

Definition 2.5 ($\texttt{SLDD}_\otimes$). Given a commutative monoid $\langle \mathcal{E}, \otimes, 1_\otimes\rangle$, $\texttt{SLDD}_\otimes$ is the 4-tuple $\langle C_{\texttt{SLDD}_\otimes}, \mathrm{Var}_{\texttt{SLDD}_\otimes}, f^{\texttt{SLDD}_\otimes}, s_{\texttt{SLDD}_\otimes}\rangle$ where $C_{\texttt{SLDD}\otimes}$ is the set of ordered VDDs α over $\mathcal{X}$ with a unique leaf L, such that $\varphi(L) = 1_\otimes$, and the arcs of which are labeled with elements of $\mathcal{E} = \mathcal{V}$, and $f^{\texttt{SLDD}\otimes}$ is defined inductively as follows: for every assignment $\vec{x}$,

- if α is the leaf node L, then $f^{\texttt{SLDD}\otimes}_\alpha(\vec{x}) = 1_\otimes$,
- otherwise, denoting N the root of α, $x \in \mathcal{X}$ its label, $d \in D_x$ such that $\langle x, d\rangle \in \vec{x}$, $a = \langle N, M\rangle$ the arc such that $v(a) = d$, and β the $\texttt{SLDD}_\otimes$ formula rooted at node M in α, $f^{\texttt{SLDD}\otimes}_\alpha(\vec{x}) = \varphi(a) \otimes f^{\texttt{SLDD}\otimes}_\beta(\vec{x})$.

We add to the root of α a value $\varphi_0 \in \mathcal{E}$ (the "offset"). The augmented interpretation function of α is $f_{\alpha,\varphi_0} = \varphi_0 \otimes f_\alpha$.

Two (ordered) monoids are particularly interesting, namely $\langle \mathbb{R}^+, +, 0\rangle$ (we call the corresponding language $\texttt{SLDD}_+$) and $\langle \mathbb{R}^+, \times, 1\rangle$ (we call the corresponding language $\texttt{SLDD}_\times$). Both $\texttt{SLDD}_+$ and $\texttt{SLDD}_\times$ formulæ have a normalization condition that provides these languages with the canonicity property; moreover, any $\texttt{SLDD}_+$ or $\texttt{SLDD}_\times$ formula can be transformed in linear time into an equivalent AADD formula. This also holds for the ADD language, targeting any of the $\texttt{SLDD}_+$, $\texttt{SLDD}_\times$, or AADD languages. Last, note that when variables are Boolean and $\mathcal{V} = \{0, 1\}$, any ADD, $\texttt{SLDD}_+$, $\texttt{SLDD}_\times$, or AADD formula can be transformed in linear time into an equivalent OBDD, and (obviously) reciprocally.

3 Queries and Transformations

We now define many queries and transformations of interest in the general framework of representation languages w.r.t. $\mathcal{V}$, that is, they are meaningful even when $\mathcal{V} \neq \mathbb{R}^+$.

Definition 3.1 (queries). Let L denote a representation language over $\mathcal{X}$ w.r.t. a set $\mathcal{V}$ totally ordered by some $\succeq$.

- L satisfies optimization $\mathbf{OPT}_{\max}$ (resp. $\mathbf{OPT}_{\min}$) iff there exists a polynomial-time algorithm that maps every L formula α to $\max_{\vec{x}\in D_f} f^{\texttt{L}}_\alpha(\vec{x})$ (resp. $\min_{\vec{x}\in D_f} f^{\texttt{L}}_\alpha(\vec{x})$).
- L satisfies equivalence $\mathbf{EQ}$ iff there exists a polynomial-time algorithm that maps every pair of L formulæ α and β to 1 if $f^{\texttt{L}}_\alpha = f^{\texttt{L}}_\beta$, and to 0 otherwise.
- L satisfies sentential entailment $\mathbf{SE}$ iff there exists a polynomial-time algorithm that maps every pair of L formulæ α and β to 1 if $\forall \vec{x}, f^{\texttt{L}}_\alpha(\vec{x}) \succeq f^{\texttt{L}}_\beta(\vec{x})$, and to 0 otherwise.
- L satisfies partial upper (resp. lower, resp. level) consistency $\mathbf{CO}_{\succeq\gamma}$ (resp $\mathbf{CO}_{\preceq\gamma}$, resp. $\mathbf{CO}_{\sim\gamma}$) iff there exists a polynomial-time algorithm that maps every $\gamma \in \mathcal{V}$ and every L formula α to 1 if $\exists \vec{x}, f^{\texttt{L}}_\alpha(\vec{x}) \succeq \gamma$ (resp. $f^{\texttt{L}}_\alpha(\vec{x}) \preceq \gamma$, resp. $f^{\texttt{L}}_\alpha(\vec{x}) \sim \gamma$) and to 0 otherwise.
- L satisfies partial upper (resp. lower, resp. level) validity $\mathbf{VA}_{\succeq\gamma}$ (resp $\mathbf{VA}_{\preceq\gamma}$, resp. $\mathbf{VA}_{\sim\gamma}$) iff there exists a polynomial-time algorithm that maps every $\gamma \in \mathcal{V}$ and every L formula α to 1 if $\forall \vec{x}, f^{\texttt{L}}_\alpha(\vec{x}) \succeq \gamma$ (resp. $f^{\texttt{L}}_\alpha(\vec{x}) \preceq \gamma$, resp. $f^{\texttt{L}}_\alpha(\vec{x}) \sim \gamma$) and to 0 otherwise.
- L satisfies $\max$-model enumeration $\mathbf{ME}_{\max}$ iff there exists a polynomial p and an algorithm that outputs, for every L formula α, all the elements of $CUT^{\max}(f^{\texttt{L}}_\alpha)$ in time $p(|\alpha|, |CUT^{\max}(f^{\texttt{L}}_\alpha)|)$,
- L satisfies $\max$-model counting $\mathbf{CT}_{\max}$ iff there exists a polynomial-time algorithm that outputs, for every L formula α, the number of elements in $CUT^{\max}(f^{\texttt{L}}_\alpha)$.

[3] Note that our definition of SLDD differs from the original one in two ways: (i) we consider only ordered diagrams; and (ii) we use a commutative monoid instead of a commutative semiring, since the second operation of the semiring does not take part in the data structure (Fargier, Marquis, and Schmidt 2013).

- L satisfies $\max$-model extraction $\mathbf{MX}_{\max}$ iff there exists a polynomial-time algorithm that maps every L formula α to an element $\vec{x}$ of $CUT^{\max}(f^{\mathtt{L}}_{\alpha})$.

We define $\mathbf{ME}_{\min}$, $\mathbf{ME}_{\succeq\gamma}$, $\mathbf{ME}_{\preceq\gamma}$, $\mathbf{ME}_{\sim\gamma}$ (resp. $\mathbf{MX}_{\min}$, $\mathbf{MX}_{\succeq\gamma}$, $\mathbf{MX}_{\preceq\gamma}$, $\mathbf{MX}_{\sim\gamma}$; resp. $\mathbf{CT}_{\min}$, $\mathbf{CT}_{\succeq\gamma}$, $\mathbf{CT}_{\preceq\gamma}$, $\mathbf{CT}_{\sim\gamma}$) in the same way as $\mathbf{ME}_{\max}$ (resp. $\mathbf{MX}_{\max}$; resp. $\mathbf{CT}_{\max}$), using the sets $CUT^{\min}(f^{\mathtt{L}}_{\alpha})$, $CUT^{\succeq\gamma}(f^{\mathtt{L}}_{\alpha})$, $CUT^{\preceq\gamma}(f^{\mathtt{L}}_{\alpha})$, $CUT^{\sim\gamma}(f^{\mathtt{L}}_{\alpha})$ instead of $CUT^{\max}(f^{\mathtt{L}}_{\alpha})$.

The $\mathbf{MX}$ and $\mathbf{ME}$ families of queries are crucial in Bayesian reasoning and in interactive configuration. They capture for instance the computation of a (the) most probable explanation(s) ($\mathbf{MX}_{\max}$), or cheapest configuration(s) ($\mathbf{MX}_{\min}$); counting is also useful for such applications, e.g., for characterizing the number of cars that are 'cheap' ($\mathbf{CT}_{\preceq\gamma}$), or the number of diseases that are likely enough to be considered ($\mathbf{CT}_{\succeq\gamma}$). Other queries like consistency and validity, as well as $\mathbf{EQ}$ and $\mathbf{SE}$, are useful for many reasoning problems (e.g., when pieces of information are encoded into weighted knowledge bases); they extend the corresponding queries defined for (Boolean) NNF formulæ.

Definition 3.2 (transformations). Let L denote a representation language over $\mathcal{X}$ w.r.t. $\mathcal{V}$, and $\odot$ an associative and commutative binary operator over $\mathcal{V}$.

- L satisfies conditioning $\mathbf{CD}$ iff there exists a polynomial-time algorithm that maps every L formula α, every $X \subseteq \mathcal{X}$, and every $\vec{x} \in D_X$ to an L representation of $f^{\mathtt{L}}_{\alpha,\vec{x}}$.
- L satisfies bounded $\odot$-combination $\odot\mathbf{BC}$ iff there exists a polynomial-time algorithm that maps every pair of L formulæ α and β to an L representation of $f^{\mathtt{L}}_{\alpha} \odot f^{\mathtt{L}}_{\beta}$.
- L satisfies $\odot$-combination $\odot\mathbf{C}$ iff there exists a polynomial-time algorithm that maps every set of L formulæ $\{\alpha_1, \ldots, \alpha_n\}$ to an L representation of $\bigodot_{i=1}^{n} f^{\mathtt{L}}_{\alpha_i}$.
- L satisfies variable (resp. single variable) $\odot$-elimination $\odot\mathbf{Elim}$ (resp. $\mathbf{S}\odot\mathbf{Elim}$) iff there exists a polynomial-time algorithm that maps every L formula α and every subset $X \subseteq \mathcal{X}$ of variables (resp every singleton $X \subseteq \mathcal{X}$) to an L representation of $\bigodot_{x\in X} \bigodot_{\vec{x}\in D_x} f^{\mathtt{L}}_{\alpha,\vec{x}}$.
- L satisfies single bounded-variable $\odot$-elimination $\mathbf{SB}\odot\mathbf{Elim}$ iff there exists a polynomial p and an algorithm that maps every L formula α and every $x \in \mathcal{X}$ to an L representation of $\bigodot_{\vec{x}\in D_x} f^{\mathtt{L}}_{\alpha,\vec{x}}$ in time $p(|\alpha|^{|D_x|})$.
- L satisfies single variable $\odot$-marginalization ($\odot\mathbf{Marg}$) iff there exists a polynomial-time algorithm that maps every L formula α and every $x \in \mathcal{X}$ to an L representation of $\bigodot_{y\in\mathcal{X}\setminus\{x\}} \bigodot_{\vec{y}\in D_y} f^{\mathtt{L}}_{\alpha,\vec{y}}$.
- L satisfies γ-cut up $\mathbf{CUT}_{\succeq\gamma}$, w.r.t. a preorder $\succeq$ on $\mathcal{V}$, iff there exists a polynomial-time algorithm that maps every L formula α and every $\gamma, a, b \in \mathcal{V}$ such that $a \succ b$, to an L representation of the function g defined by $g(\vec{x}) = a$ if $\vec{x} \in CUT^{\succeq\gamma}(f_\alpha)$, and $g(\vec{x}) = b$ otherwise.

We define $\mathbf{CUT}_{\preceq\gamma}$, $\mathbf{CUT}_{\sim\gamma}$, $\mathbf{CUT}_{\max}$, and $\mathbf{CUT}_{\min}$ in the same way, from the sets $CUT^{\preceq\gamma}(f_\alpha)$, $CUT^{\sim\gamma}(f_\alpha)$, $CUT^{\max}(f_\alpha)$, and $CUT^{\min}(f_\alpha)$, respectively.

The classical forgetting of variables corresponds to their $\max$-elimination. Single variable marginalization is equivalent to the elimination of all variables but one. For instance, sum-marginalization is important in Bayes nets for achieving the posterior marginal request (see Darwiche (2009)); and in configuration problems, the $\min$-marginalization on a variable in a VDD formula representing some price function amounts to computing the minimal price associated with each possible value of this variable (an option, in the car example; see Astesana, Cosserat, and Fargier (2010)). Another example is value iteration in stochastic planning, which can be performed via a sequence of sum-eliminations and $\times$-combinations on ADD formulæ (Hoey et al. 1999).

Finally, the family of cuts captures the restriction of the function to the optimal assignments (e.g., the cheapest cars, the most probable explanations) or to those that are good enough (cars cheaper than γ euros, diseases with a probability greater than γ). For generality, we defined this request as a transformation within the language, but choosing $a = 1$ and $b = 0$ when $\mathcal{V} = \mathbb{R}^+$, cutting a VDD results in an MDD; the MDD language (Srinivasan et al. 1990) is a direct extension of OBDD to non-Boolean variables, and it satisfies roughly the same queries and transformations as OBDD, e.g., $\mathbf{CO}$, $\mathbf{CD}$, $\mathbf{SE}$, etc. (Amilhastre et al. 2012).

4 A KC Map for Ordered $\mathbb{R}^+$-VDDs

We can now draw the map of existing ordered VDD languages representing functions from $\mathcal{X}$ to $\mathbb{R}^+$, namely ADD, $\mathtt{SLDD}_{\times}$, $\mathtt{SLDD}_{+}$, and AADD. That is, we focus in the following on $\mathcal{V} = \mathbb{R}^+$, with $\succeq$ being the usual $\geq$ order on $\mathbb{R}^+$, and $\odot \in \{\max, \min, +, \times\}$. We refrain from explicitly investigating $\mathtt{SLDD}_{\max}$ and $\mathtt{SLDD}_{\min}$, since it has been shown that these languages are equivalent, up to a polynomial transformation, to ADD (Fargier, Marquis, and Schmidt 2013).

Each of the four languages satisfies $\mathbf{CD}$: conditioning a formula α by an assignment $\vec{x}$ can be achieved in linear time using an algorithm similar to the OBDD one. Also, since the representation of any function as a reduced ADD (resp. reduced and normalized AADD, $\mathtt{SLDD}_{\times}$, $\mathtt{SLDD}_{+}$) is unique, and the reduction and normalization procedures of each language are polynomial-time, $\mathbf{EQ}$ is satisfied by each of the four languages. Finally, $\mathbf{SE}$ is satisfied by ADD, $\mathtt{SLDD}_{+}$, and $\mathtt{SLDD}_{\times}$, using a combination of cuts, inversion of labels, and translation to MDD. Whether AADD satisfies $\mathbf{SE}$ remains open.

Proposition 4.1.

- ADD, $\mathtt{SLDD}_{+}$, $\mathtt{SLDD}_{\times}$, *and* AADD *satisfy* $\mathbf{EQ}$ *and* $\mathbf{CD}$.
- ADD, $\mathtt{SLDD}_{+}$, *and* $\mathtt{SLDD}_{\times}$ *satisfy* $\mathbf{SE}$.

Let us now draw the KC map for the requests identified in the previous section. We distinguish three classes of requests: those related to optimization, that are tractable on the VDD languages; those related to cutting with respect to some threshold γ, that may become harder; and the combination and projection transformations.

Requests related to optimization tasks. The results we obtained are summarized in Table 1. As already mentioned, $\mathbf{OPT}_{\max}$ and $\mathbf{OPT}_{\min}$ have been shown to be tractable for

Table 1: Results about basic queries, optimization, and γ-cutting; $\surd$ means "satisfies", $\bullet$ means "does not satisfy", and $\circ$ means "does not satisfy unless $\mathsf{P} = \mathsf{NP}$". Results for additive valued constraint satisfaction problems ($\mathtt{VCSP}_+$) are given here as a baseline.

Query	ADD	$\mathtt{SLDD}_+$	$\mathtt{SLDD}_\times$	AADD	$\mathtt{VCSP}_+$
$\mathbf{CD}$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$
$\mathbf{EQ}$	$\surd$	$\surd$	$\surd$	$\surd$	?
$\mathbf{SE}$	$\surd$	$\surd$	$\surd$	?	$\circ$
$\mathbf{OPT}_{\max}$ / $\mathbf{OPT}_{\min}$	$\surd$	$\surd$	$\surd$	$\surd$	$\circ$
$\mathbf{CT}_{\max}$ / $\mathbf{CT}_{\min}$	$\surd$	$\surd$	$\surd$	$\surd$	$\circ$
$\mathbf{ME}_{\max}$ / $\mathbf{ME}_{\min}$	$\surd$	$\surd$	$\surd$	$\surd$	$\circ$
$\mathbf{MX}_{\max}$ / $\mathbf{MX}_{\min}$	$\surd$	$\surd$	$\surd$	$\surd$	$\circ$
$\mathbf{CUT}_{\max}$ / $\mathbf{CUT}_{\min}$	$\surd$	$\surd$	$\surd$	$\surd$	?
$\mathbf{VA}_{\sim\gamma}$	$\surd$	$\surd$	$\surd$	$\surd$	?
$\mathbf{VA}_{\succeq\gamma}$ / $\mathbf{VA}_{\preceq\gamma}$	$\surd$	$\surd$	$\surd$	$\surd$	$\circ$
$\mathbf{CO}_{\sim\gamma}$	$\surd$	$\circ$	$\circ$	$\circ$	$\circ$
$\mathbf{CO}_{\succeq\gamma}$ / $\mathbf{CO}_{\preceq\gamma}$	$\surd$	$\surd$	$\surd$	$\surd$	$\circ$
$\mathbf{ME}_{\sim\gamma}$	$\surd$	$\circ$	$\circ$	$\circ$	$\circ$
$\mathbf{ME}_{\succeq\gamma}$ / $\mathbf{ME}_{\preceq\gamma}$	$\surd$	$\surd$	$\surd$	$\surd$	$\circ$
$\mathbf{MX}_{\sim\gamma}$	$\surd$	$\circ$	$\circ$	$\circ$	$\circ$
$\mathbf{MX}_{\succeq\gamma}$ / $\mathbf{MX}_{\preceq\gamma}$	$\surd$	$\surd$	$\surd$	$\surd$	$\circ$
$\mathbf{CUT}_{\sim\gamma}$	$\surd$	$\circ$	$\circ$	$\circ$	?
$\mathbf{CUT}_{\succeq\gamma}$ / $\mathbf{CUT}_{\preceq\gamma}$	$\surd$	$\bullet$	$\bullet$	$\bullet$	?
$\mathbf{CT}_{\sim\gamma}$	$\surd$	$\circ$	$\circ$	$\circ$	$\circ$
$\mathbf{CT}_{\succeq\gamma}$ / $\mathbf{CT}_{\preceq\gamma}$	$\surd$	$\circ$	$\circ$	$\circ$	$\circ$

normalized AADD formulæ (Sanner and McAllester 2005), for SLDD formulæ (Wilson 2005), and their satisfaction is obvious for ADD. All tractability results of Table 1 can be related to the fact that (i) VDDs are circuit-free graphs, and (ii) the aggregation of the φ values is monotonic in the classes considered. In such diagrams, minimal (resp. maximal) paths can be obtained in polynomial time, thanks to a shortest (resp. longest) path algorithm; this is the basis for a polynomial-time procedure that builds an MDD formula representing the optimal assignments, and hence implies the satisfaction of $\mathbf{CUT}_{\max}$ and $\mathbf{CUT}_{\min}$.

Proposition 4.2. *ADD, $\mathtt{SLDD}_+$, $\mathtt{SLDD}_\times$, and AADD satisfy $\mathbf{OPT}_{\max}$, $\mathbf{OPT}_{\min}$, $\mathbf{CUT}_{\max}$, $\mathbf{CUT}_{\min}$, $\mathbf{ME}_{\max}$, $\mathbf{ME}_{\min}$, $\mathbf{MX}_{\max}$, $\mathbf{MX}_{\min}$, $\mathbf{CT}_{\max}$, and $\mathbf{CT}_{\min}$.*

Requests related to γ-cuts. Requests related to pure optimization are easy to solve. However, optimization alone is not sufficient for many applications; for instance, a customer looking for a car is not always interested in finding out one of the cheapest cars: a 'cheap' car (i.e., with a cost lower than a given threshold) may be more interesting than the cheapest ones if it fulfills other desiderata of the customer. Hence the importance of requests related to γ-cuts.

Fortunately, comparing the maximal (resp. minimal) value of $f_\alpha^{\mathtt{L}}$ (which can be computed in polynomial time) to some $\gamma \in \mathcal{V}$ is enough to decide whether there exists an $\vec{x}$ such that $f_\alpha^{\mathtt{L}}(\vec{x}) \succeq \gamma$ (resp. $\preceq \gamma$). Similarly, deciding the γ-validity of a formula α only requires the comparison of γ to the maximal (resp. minimal) value of $f_\alpha^{\mathtt{L}}$.

Proposition 4.3. *ADD, $\mathtt{SLDD}_+$, $\mathtt{SLDD}_\times$, and AADD satisfy $\mathbf{CO}_{\succeq\gamma}$, $\mathbf{VA}_{\succeq\gamma}$, $\mathbf{CO}_{\preceq\gamma}$, $\mathbf{VA}_{\preceq\gamma}$, and $\mathbf{VA}_{\sim\gamma}$.*

Although checking that there is an assignment that leads to a valuation greater than or equal to γ is polynomial, deciding whether there exists an assignment leading *exactly* to γ is not tractable for $\mathtt{SLDD}_+$, $\mathtt{SLDD}_\times$, and AADD. The proof is based on a polynomial reduction from the SUBSET SUM decision problem (Garey and Johnson 1979).

Proposition 4.4. *$\mathtt{SLDD}_+$, $\mathtt{SLDD}_\times$, and AADD do not satisfy $\mathbf{CO}_{\sim\gamma}$ unless $\mathsf{P} = \mathsf{NP}$.*

It is quite clear that the satisfaction of $\mathbf{MX}_{\sim\gamma}$ or $\mathbf{CT}_{\sim\gamma}$ is a sufficient condition to that of $\mathbf{CO}_{\sim\gamma}$; and this holds in all generality, not only for our four $\mathbb{R}^+$-valued languages. Similarly, it can be shown that $\mathbf{ME}_{\sim\gamma}$ is a sufficient condition for $\mathbf{CO}_{\sim\gamma}$, and that $\mathbf{CUT}_{\sim\gamma}$ implies $\mathbf{MX}_{\sim\gamma}$ as long as one of the $\mathbf{MX}$ queries is satisfied.

Proposition 4.5. *Let L be a representation language over $\mathcal{X}$ w.r.t. $\mathcal{V}$, where $\mathcal{V}$ is totally ordered by a relation $\succeq$.*

- *If L satisfies both $\mathbf{CUT}_{\sim\gamma}$ and one of the $\mathbf{MX}$ queries, then it satisfies $\mathbf{MX}_{\sim\gamma}$.*
- *If L satisfies $\mathbf{MX}_{\sim\gamma}$, $\mathbf{CT}_{\sim\gamma}$, or $\mathbf{ME}_{\sim\gamma}$, then it satisfies $\mathbf{CO}_{\sim\gamma}$.*

Corollary 4.6. *$\mathtt{SLDD}_+$, $\mathtt{SLDD}_\times$, and AADD do not satisfy $\mathbf{MX}_{\sim\gamma}$, $\mathbf{CUT}_{\sim\gamma}$, $\mathbf{CT}_{\sim\gamma}$, or $\mathbf{ME}_{\sim\gamma}$ unless $\mathsf{P} = \mathsf{NP}$.*

Thus, except for $\mathbf{VA}_{\sim\gamma}$, queries about a precise cut of the function represented by an arc-labeled (in the wide sense) VDD are intractable. The situation is more nuanced when lower and upper cuts are considered instead.

Proposition 4.7. *Let $\mathtt{L} \in \{\mathtt{SLDD}_+, \mathtt{SLDD}_\times, \mathtt{AADD}\}$.*

- *L satisfies $\mathbf{ME}_{\succeq\gamma}$, $\mathbf{ME}_{\preceq\gamma}$, $\mathbf{MX}_{\succeq\gamma}$, and $\mathbf{MX}_{\preceq\gamma}$.*
- *L does not satisfy $\mathbf{CT}_{\succeq\gamma}$ or $\mathbf{CT}_{\preceq\gamma}$ unless $\mathsf{P} = \mathsf{NP}$.*
- *L does not satisfy $\mathbf{CUT}_{\succeq\gamma}$ or $\mathbf{CUT}_{\preceq\gamma}$.*

For proving the first item, the basic idea is that from any node of a normalized AADD formula, there is a path to the leaf that has value 1, and a path to the leaf that has value 0. A path from the root to this node gives it an offset, gathered from its arcs, say $\langle p, q\rangle$. Hence we can enumerate all paths, without exploring nodes for which $p + q < \gamma$ for $\mathbf{ME}_{\succeq\gamma}$ (resp. $p > \gamma$ for $\mathbf{ME}_{\preceq\gamma}$).

The proof of the next item comes from that fact that if $\mathbf{CT}_{\succeq\gamma}$ held, then we could count the assignments $\vec{x}$ such that $f_\alpha(\vec{x}) \geq \gamma$, and also those for which $f_\alpha(\vec{x}) > \gamma$; that is, we could satisfy $\mathbf{CT}_{\sim\gamma}$, which has been shown intractable (and similarly for $\mathbf{CT}_{\preceq\gamma}$).

The intractability of $\mathbf{CUT}_{\succeq\gamma}$ or $\mathbf{CUT}_{\preceq\gamma}$ is unconditional, and the proofs are trickier. For the specific case of $\mathbf{CUT}_{\succeq\gamma}$ on $\mathtt{SLDD}_+$, the proof uses the fact that the function $f(\vec{y}\cdot\vec{z}) = \sum_{i=1}^n y_i \cdot 2^{n-i} + \sum_{i=1}^n z_i \cdot 2^{n-i}$ can be represented as an $\mathtt{SLDD}_+$ formula of $2n+1$ nodes only, using the variable ordering $y_1 \lhd \cdots \lhd y_n \lhd z_1 \lhd \cdots \lhd z_n$, whereas there is no polynomial-size $\mathtt{OBDD}_\lhd$ representation of the function returning 1 when $f(\vec{y}\cdot\vec{z}) \geq 2^n$ and 0 otherwise. The other proofs are similar, using well-chosen functions instead of f.

These results about AADD and SLDD contrast with the case of ADD, which satisfies each $\mathbf{CUT}$ transformation (γ-cuts

are obtained thanks to a simple leaf-merging procedure), and thus satisfies each of the $\mathbf{MX}$, $\mathbf{ME}$, $\mathbf{CT}$, and $\mathbf{CO}$ queries.

Combination, variable elimination, marginalization. None of the VDD languages considered in this paper satisfies the unbounded combination or unbounded variable elimination transformations; this result is not very surprising, observing that (i) unbounded disjunctive combination ($\vee\mathbf{C}$) and forgetting ($\mathbf{FO}$) are not satisfied by OBDD; (ii) any OBDD formula can be viewed as an ADD formula, and thus be translated in polynomial time into an $\texttt{SLDD}_+$ (resp. $\texttt{SLDD}_\times$, AADD) formula; and (iii) the disjunction of several OBDD formulæ amounts to cutting their $+$-combination at the minimal level ($\mathbf{CUT}_{\min}$ is satisfied): by construction, the countermodels of the disjunction are the $\vec{x}$ such that $\varphi(\vec{x}) = 0$, so the resulting formula can be viewed as an OBDD formula the negation of which is equivalent to the disjunction of the original formulæ). The other proofs are based on similar arguments. Note that contrary to the OBDD case, even $\odot$-eliminating a single variable is hard (roughly speaking, the $\odot$-combination of two formulæ can be built by $\odot$-eliminating an additional variable).

Proposition 4.8. *For any* $\texttt{L} \in \{\texttt{ADD}, \texttt{SLDD}_+, \texttt{SLDD}_\times, \texttt{AADD}\}$ *and any* $\odot \in \{\max, \min, +, \times\}$, L *does not satisfy* $\odot\mathbf{C}$, $\odot\mathbf{Elim}$, *or* $\mathbf{S}\odot\mathbf{Elim}$.

More difficult is the question of bounded combination. An extension of "apply" algorithm of Bryant (1986), which is polynomial-time on OBDD structures for the AND and OR operators, has been proposed (Sanner and McAllester 2005) to compute the combination (e.g., by $+$, $\times$, max, min) of two AADD formulæ; it can be easily adapted to the other VDD languages considered in this paper. However, the complexity of this "apply" algorithm had not been formally identified. One of the main results of this paper is that bounded combinations are not tractable for AADD, which implies that the extended "apply" is not a polynomial-time algorithm.

Proposition 4.9.

- $\texttt{SLDD}_+$, $\texttt{SLDD}_\times$, *and* AADD *do not satisfy* $\max\mathbf{BC}$, $\min\mathbf{BC}$, $\mathbf{SB}\max\mathbf{Elim}$, *or* $\mathbf{SB}\min\mathbf{Elim}$.
- $\texttt{SLDD}_\times$ *and* AADD *do not satisfy* $+\mathbf{BC}$ *or* $\mathbf{SB}+\mathbf{Elim}$.
- $\texttt{SLDD}_+$ *and* AADD *do not satisfy* $\times\mathbf{BC}$ *or* $\mathbf{SB}\times\mathbf{Elim}$.

The difficulty of $\max\mathbf{BC}$ or $\min\mathbf{BC}$ comes from that of $\mathbf{CUT}_{\succeq\gamma}$and $\mathbf{CUT}_{\preceq\gamma}$. We show the difficulty of $\times\mathbf{BC}$ by considering the two following functions on Boolean variables: $f(\vec{x}) = \sum_{i=0}^{n-1} x_i \cdot 2^i$ (representation of an integer by a bitvector) and $g(\vec{x}) = 2^{n+1} - f(\vec{x})$; each can be represented as an $\texttt{SLDD}_+$ formula (with ordering $x_0 \lhd x_1 \lhd \cdots \lhd x_{n-1}$) with $n+1$ nodes and $2n$ arcs. Then we can show that the $\texttt{SLDD}_+$ representation of $f \times g$ (using the same variable ordering) contains an exponential number of terminal arcs, even if all nodes at the last level have been normalized; this proves that each ordered $\texttt{SLDD}_+$ representation of $f \times g$ is of exponential size. The other proofs are similar, using well-chosen functions f and g.

There are actually only two cases in which bounded combinations are tractable: when the language considered is ADD, and when the combination operator is also the one that aggregates arc values in the language considered (i.e., addition for $\texttt{SLDD}_+$, and product for $\texttt{SLDD}_\times$). In these cases, it is also polynomial to eliminate a single variable with a bounded domain, by definition of variable elimination.

Table 2: Results about transformations (legend in Table 1).

Transformation	ADD	$\texttt{SLDD}_+$	$\texttt{SLDD}_\times$	AADD
$\max\mathbf{BC}$ / $\min\mathbf{BC}$	√	•	•	•
$+\mathbf{BC}$	√	√	•	•
$\times\mathbf{BC}$	√	•	√	•
$\max\mathbf{C}$ / $\min\mathbf{C}$	•	•	•	•
$+\mathbf{C}$ / $\times\mathbf{C}$	•	•	•	•
$\max\mathbf{Elim}$ / $\min\mathbf{Elim}$	•	•	•	•
$+\mathbf{Elim}$ / $\times\mathbf{Elim}$	•	•	•	•
$\mathbf{S}\max\mathbf{Elim}$ / $\mathbf{S}\min\mathbf{Elim}$	•	•	•	•
$\mathbf{S}+\mathbf{Elim}$ / $\mathbf{S}\times\mathbf{Elim}$	•	•	•	•
$\mathbf{SB}\max\mathbf{Elim}$ / $\mathbf{SB}\min\mathbf{Elim}$	√	•	•	•
$\mathbf{SB}+\mathbf{Elim}$	√	√	•	•
$\mathbf{SB}\times\mathbf{Elim}$	√	•	√	•
$\max\mathbf{Marg}$ / $\min\mathbf{Marg}$	√	√	√	√
$+\mathbf{Marg}$	√	√	√	√
$\times\mathbf{Marg}$	√	?	√	?

Proposition 4.10.

- $\texttt{SLDD}_+$ *satisfies* $+\mathbf{BC}$ *and* $\mathbf{SB}+\mathbf{Elim}$.
- $\texttt{SLDD}_\times$ *satisfies* $\times\mathbf{BC}$ *and* $\mathbf{SB}\times\mathbf{Elim}$.
- ADD *satisfies* $\odot\mathbf{BC}$ *and* $\mathbf{SB}\odot\mathbf{Elim}$, *for any operation* $\odot \in \{\times, +, \min, \max\}$.

Finally, we have proved that $\times$-marginalization is tractable for ADD and $\texttt{SLDD}_\times$, and that $\max$-marginalization, $\min$-marginalization and $+$-marginalization are tractable for all languages considered. Whether AADD and $\texttt{SLDD}_+$ satisfy $\times$-marginalization remains an open question.

5 Conclusion

In this paper, we presented a complexity analysis of VDD languages, based on a set of queries and transformations of interest. Requests related to optimization appear as tractable for all VDD languages, as well as additive marginalization and some of the queries related to cutting. Cutting queries prove computationally difficult when a level γ is has to be reached exactly, and this is also the case for the transformations we considered. One of the main results of the paper is that bounded combinations are not tractable on VDD languages, which implies that no "apply" algorithm can run in polynomial time on AADD formulæ in the general case (this is the case even for simple "Boolean-style" operations such as $\min$ and $\max$). When bounded additive (resp. multiplicative) combination is required to be achieved efficiently, the time/space tradeoff offered by $\texttt{SLDD}_+$ (resp. $\texttt{SLDD}_\times$) is valuable. Finally, it turns out that the complexity of the various queries related to optimization is better for the VDD languages than for other languages dedicated to the representation of non-Boolean functions, such as $\texttt{VCSP}_+$ (and similar results are expected for GAI nets or Bayes nets, for which optimization is also hard).

Acknowledgements

This work was partially supported by the project BR4CP ANR-11-BS02-008 of the French National Agency for Research.

References

Amilhastre, J.; Fargier, H.; Niveau, A.; and Pralet, C. 2012. Compiling CSPs: A complexity map of (non-deterministic) multivalued decision diagrams. In *Proceedings of ICTAI'12*, 1–8.

Amilhastre, J.; Fargier, H.; and Marquis, P. 2002. Consistency restoration and explanations in dynamic CSPs: Application to configuration. *Artificial Intelligence* 135(1–2):199–234.

Astesana, J.-M.; Cosserat, L.; and Fargier, H. 2010. Constraint-based vehicle configuration: A case study. In *Proceedings of ICTAI'10*, 68–75.

Bacchus, F., and Grove, A. J. 1995. Graphical models for preference and utility. In *Proceedings of UAI'95*, 3–10.

Bahar, R. I.; Frohm, E. A.; Gaona, C. M.; Hachtel, G. D.; Macii, E.; Pardo, A.; and Somenzi, F. 1993. Algebraic decision diagrams and their applications. In *Proceedings of ICCAD'93*, 188–191.

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

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

Darwiche, A. 2009. *Modeling and Reasoning with Bayesian Networks*. Cambridge University Press.

Fargier, H.; Marquis, P.; and Schmidt, N. 2013. Semiring labelled decision diagrams, revisited: Canonicity and spatial efficiency issues. In *Proceedings of IJCAI'13*, 884–890.

Garey, M. R., and Johnson, D. S. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman.

Gogic, G.; Kautz, H.; Papadimitriou, C.; and Selman, B. 1995. The comparative linguistics of knowledge representation. In *Proceedings of IJCAI'95*, 862–869.

Hoey, J.; St-Aubin, R.; Hu, A. J.; and Boutilier, C. 1999. SPUDD: Stochastic planning using decision diagrams. In *Proceedings of UAI'99*, 279–288.

Lai, Y.-T., and Sastry, S. 1992. Edge-valued binary decision diagrams for multi-level hierarchical verification. In *Proceedings of DAC'92*, 608–613.

Lai, Y.-T.; Pedram, M.; and Vrudhula, S. B. K. 1996. Formal verification using edge-valued binary decision diagrams. *IEEE Transactions on Computers* 45(2):247–255.

Pearl, J. 1989. *Probabilistic reasoning in intelligent systems – networks of plausible inference*. Morgan Kaufmann series in representation and reasoning. Morgan Kaufmann.

Sanner, S., and McAllester, D. A. 2005. Affine algebraic decision diagrams (AADDs) and their application to structured probabilistic inference. In *Proceedings of IJCAI'05*, 1384–1390.

Schiex, T.; Fargier, H.; and Verfaillie, G. 1995. Valued constraint satisfaction problems: Hard and easy problems. In *Proceedings of IJCAI'95 (1)*, 631–639.

Srinivasan, A.; Kam, T.; Malik, S.; and Brayton, R. K. 1990. Algorithms for discrete function manipulation. In *Proceedings of ICCAD'90*, 92–95.

Tafertshofer, P., and Pedram, M. 1997. Factored edge-valued binary decision diagrams. *Formal Methods in System Design* 10(2/3).

Wilson, N. 2005. Decision diagrams for the computation of semiring valuations. In *Proceedings of IJCAI'05*, 331–336.