On Decomposability and Interaction Functions

Knot Pipatsrisawat and **Adnan Darwiche**[1]

Abstract. A formal notion of a Boolean-function decomposition was introduced recently and used to provide lower bounds on various representations of Boolean functions, which are subsets of decomposable negation normal form (DNNF). This notion has introduced a fundamental optimization problem for DNNF representations, which calls for computing decompositions of minimal size for a given partition of the function variables. We consider the problem of computing optimal decompositions in this paper for general Boolean functions and those represented using CNFs. We introduce the notion of an *interaction function,* which characterizes the relationship between two sets of variables and can form the basis of obtaining such decompositions. We contrast the use of these functions to the current practice of computing decompositions, which is based on heuristic methods that can be viewed as using approximations of interaction functions. We show that current methods can lead to decompositions that are exponentially larger than optimal decompositions, pinpoint the specific reasons for this lack of optimality, and finally present empirical results that illustrate some characteristics of interaction functions in contrast to their approximations.

1 Introduction

Decomposability has been identified as a fundamental property that underlies many tractable languages in propositional logic, such as disjunctive normal form (DNF), ordered binary decision diagrams (OBDD) and the widely encompassing decomposable negation norma form (DNNF). Decomposability is a property of conjunctions, requiring that conjuncts share no variables [4]. Given decomposability, one can devise polynomial time algorithms for many queries that are known to be generally intractable. Satisfiability is one such example, which can be tested for efficiently once we have decomposability [6].

Establishing decomposability lies at the heart of many reasoning systems such as model counters [2, 1, 11] and knowledge compilers [5, 7, 8]. In many of these systems, decomposability is established by instantiating enough variables in order to syntactically disconnect the underlying formula into sub-formulas that no longer share a variable. This process can then be applied recursively until each sub-formula become trivial or sufficiently simple.

Recently, a restricted version of decomposability has been identified, which requires decomposable formulas to adhere to a data structure known as a *vtree* [9]. The vtree is simply a full binary tree whose leaves are in one-to-one correspondence with the variables of interest. Hence, each internal vtree node corresponds to a set of variables and to a particular partition of these variables (defined by the variables in its two children). A decomposable formula adheres to a vtree if every sub-formula is decomposable across the partition defined by the corresponding vtree node over its variables. This type of decomposability is called *structured decomposability* and leads to stronger properties than plain decomposability. A number of languages based on structured decomposability have been identified and studied in [9], including the influential OBDD.

The process of decomposing a formula with respect to a variable partition has been formulated more explicitly recently using the formal notion of a *decomposition* [10]. In particular, different types of decompositions have been defined and then shown to underlie corresponding subsets of decomposable negation normal forms (DNNF). This formalization, which we will review next, has crystalized a fundamental optimization problem in automated reasoning, which is the problem of computing an optimal decomposition of a formula across a given variable partition. None of the existing systems, however, try to optimize the process of constructing decompositions as they rely mostly on heuristic methods for computing such decompositions.

The goal of this paper is to introduce a new notion, which we call an *interaction function,* that can shed light on the limitation of current decomposition techniques, and that can provide a basis for more principled and optimal decomposition techniques. Intuitively, an interaction function is a formula α that captures precisely the knowledge encoded by another formula β on the relationship between two sets of variables $\mathbf{X}$ and $\mathbf{Y}$. Hence, if the goal is to compute a decomposition of formula β across the partition $(\mathbf{X}, \mathbf{Y})$, then it is sufficient to only obtain a decomposition of the corresponding interaction function for these variables.

This paper is based on a number of contributions. First, a formal definition of the interaction function and its properties. Second, a result showing that optimal decompositions of the interaction function can be converted into optimal decompositions of the original formula. Third, a result which shows that current syntactic techniques for generating decompositions of CNF can be viewed as working with an approximation of the interaction function, where we formulate precisely the distinction between what is currently used and the interaction function. Fourth, we show examples where these syntactic decomposition techniques can be exponentially worse than optimal. Fifth, we suggest a method for computing interaction functions, albeit impractical, and use it to provide some empirical results on the nature of interaction functions, in comparison to the original formulas we are trying to decompose, and the syntactic techniques used by some systems to compute decompositions.

We start next by providing some technical preliminaries. We then review the newly formulated notion of a decomposition, followed by the fundamental concept in this paper: the notion of an interaction function. The following sections explicate the various results discussed earlier. We provide some proofs in the appendix and leave the others for the full version of the paper because of space limitations.

[1] Computer Science Department, University of California, Los Angeles, email: {thammakn,darwiche}@cs.ucla.edu

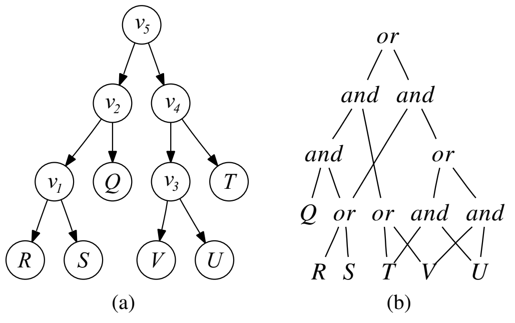

Figure 1. A vtree (a) and a respecting structured DNNF (b).

2 Basic Definitions

In this section, we provide definitions of basic concepts that will be used throughout the paper. A *Boolean function* (or simply function) over a set of variables $\mathbf{Z}$ is a function that maps each complete assignment of variables $\mathbf{Z}$ to either *true* or *false* (most of our definitions will be based on Boolean functions instead of Boolean formulas). The *conditioning* of function f on variable assignment $\mathbf{x}$ (of variables $\mathbf{X}$) is defined as $f|\mathbf{x} = \exists\mathbf{X}(f \wedge \mathbf{x})$. If f is represented by a formula, we can obtain $f|\mathbf{x}$ by replacing each occurrence of variable $X \in \mathbf{X}$ by its value in $\mathbf{x}$. We also refer to $\mathbf{x}$ as an *instantiation* of variables $\mathbf{X}$. A function f *depends only* on variables $\mathbf{Z}$ iff for any variable $X \notin \mathbf{Z}$, we have $f|X = f|\neg X$. We will write $f(\mathbf{Z})$ to mean that f is a function that depends only on variables $\mathbf{Z}$. Note that $f(\mathbf{Z})$ may not necessary depend on every variable in $\mathbf{Z}$.

A conjunction is *decomposable* if each pair of its conjuncts share no variables. A *negation normal form* (NNF) is a DAG whose internal nodes are labelled with disjunctions and conjunctions and whose leaf nodes are labeled with literals or the constants *true* and *false*. An NNF is decomposable (called a DNNF) iff each of its conjunctions is decomposable; see Figure 1(b). We use $vars(N)$ to denote the set of variables mentioned by an NNF node N.

Figure 1(a) depicts an example vtree. Given an internal node v in a vtree for variables $\mathbf{Z}$, we use v^l and v^r to refer to its left and right children, use $vars(v)$ to denote the set of variables at or below v in the tree. A DNNF respects a vtree iff every and-node has exactly two children N^l and N^r, and we have $vars(N^l) \subseteq vars(v^l)$ and $vars(N^r) \subseteq vars(v^r)$ for some vtree node v. The DNNF in Figure 1(b) respects the vtree in Figure 1(a).

We use an upper case letter to denote a variable (e.g., X) and a lower case letter to denote its instantiation (e.g., x). Moreover, we use a bold upper case letter to denote a set of variables (e.g., $\mathbf{X}$) and a bold lower case letter to denote their instantiations (e.g., $\mathbf{x}$).

3 Decompositions of Boolean Functions

We review in this section the key notion of a decomposition, which was formulated recently [10]. This notion provides an abstraction of many of the tractable logical representations that have been proposed and studied in the literature. For example, this notion has been used in [10] to establish lower bounds on the sizes of these representations, highlighting the problem of constructing optimal decompositions as the central problem in optimizing the size of these representations.

In the following definition and the rest of the paper, we will assume that variables $\mathbf{X}$ and $\mathbf{Y}$ form a partition of variables $\mathbf{Z}$.

Definition 1 *An $\underline{\mathbf{X}\text{-decomposition}}$ of function $f(\mathbf{Z})$ is a collection of functions (a.k.a. elements) $f^1(\mathbf{Z}), \ldots, f^m(\mathbf{Z})$ such that (i) $f = f^1 \vee \ldots \vee f^m$ and (ii) each f^i can be expressed as follows:*

$$f^i(\mathbf{Z}) = g^i(\mathbf{X}) \wedge h^i(\mathbf{Y}).$$

The number m is called the $\underline{\text{size}}$ of the decomposition in this case. A decomposition is $\underline{\text{minimal}}$ if no other decomposition has a smaller size.

Note that an $\mathbf{X}$-decomposition for $f(\mathbf{Z})$ is also a $\mathbf{Y}$-decomposition for $f(\mathbf{Z})$. We will typically just say "decomposition" when variables $\mathbf{X}$ and $\mathbf{Y}$ are explicated.

Consider the boolean function $f = (X_1 \wedge Y_1) \vee (X_2 \wedge Y_2) \vee (X_2 \wedge Y_3)$ and the partition $\mathbf{X} = \{X_1, X_2\}, \mathbf{Y} = \{Y_1, Y_2, Y_3\}$. The following are two decompositions of this function:

$g(\mathbf{X})$	$h(\mathbf{Y})$
X_1	Y_1
X_2	$Y_2 \vee Y_3$

$g(\mathbf{X})$	$h(\mathbf{Y})$
X_1	Y_1
$\neg X_1 \wedge X_2$	$Y_2 \vee Y_3$
$X_1 \wedge X_2$	$\neg Y_1 \wedge (Y_2 \vee Y_3)$

Each row corresponds to an element of the decomposition. Moreover, we present each element in terms of its $\mathbf{X}$ and $\mathbf{Y}$ components; the corresponding element can be obtained by simply the conjoining these components together.

The notion of a decomposition was used in [10] to characterize various subsets of decomposable negation normal form, and to establish lower bounds on the sizes of these subsets depending on the sizes of their corresponding optimal decompositions. Algorithm 1 provides pseudocode for constructing a structured DNNF representation for a given Boolean function and a given vtree. The size and type of resulting DNNF is completely determined by the size and type of the decompositions computed by the algorithm on Lines 3-5.[2]

Algorithm 1 does not assume any particular representation of the given Boolean function. Hence, as is, this algorithm is only meant to highlight the central role of decompositions in characterizing the size and type of DNNFs.

4 The Interaction Function

In this section, we introduce a new notion, called *interaction function,* that captures the logical interactions between two sets of variables in a given boolean function.

Definition 2 (Interaction) *Let $(\mathbf{X}, \mathbf{Y})$ be any partition of set $\mathbf{Z}$. The $\underline{(\mathbf{X}, \mathbf{Y})\text{ interaction function}}$ of function $f(\mathbf{Z})$ is a function defined as follows:*

$$f_{\mathbf{XY}} = f \vee \neg(\exists\mathbf{X}f) \vee \neg(\exists\mathbf{Y}f).$$

The notion of interaction function allows us to view any boolean function $f(\mathbf{Z})$ as a conjunction of three components: (i) one that captures constraints on the values of variables $\mathbf{X}$: $\exists\mathbf{Y}f$ (ii) one that captures constraints on the values of variables $\mathbf{Y}$: $\exists\mathbf{X}f$ and (iii) one that captures the constraints between variables $\mathbf{X}$ and $\mathbf{Y}$: $f_{\mathbf{XY}}$. The following proposition formalizes these properties and goes further to state that the interaction function, as defined above, is in fact the most general function for this purpose.

[2] The vtree also determines the type of DNNF generated. For example, a linear vtree (which encodes a total variable order), leads to OBDD representations.

Algorithm 1: $DNNF(v, f)$: keeps a cache $cache(.,.)$ where the first argument is a vtree node and the second argument is a function. The cache is initialized to nil.

```
input:
  $v$:  vtree node
  $f$:  function that depends only on $vars(v)$
output: DNNF for function $f$ respecting vtree rooted at node $v$
main:
 1: If $f = true$ or $f = false$ or $v$ is a leaf node, return $f$
 2: If $cache(v, f) \neq nil$, return $cache(v, f)$
 3: $\mathbf{X} \leftarrow$ variables in the vtree rooted at $v^l$
 4: $\mathbf{Y} \leftarrow$ variables in the vtree rooted at $v^r$
 5: $g^1(\mathbf{X}) \wedge h^1(\mathbf{Y}), \ldots, g^m(\mathbf{X}) \wedge h^m(\mathbf{Y}) \leftarrow$ a decomposition
    of $f$
 6: $\alpha \leftarrow false$
 7: for $i = 1$ to $m$ do
 8:    $\alpha \leftarrow \alpha \vee (DNNF(v^l, g^i(\mathbf{X})) \wedge DNNF(v^r, h^i(\mathbf{Y})))$
 9: end for
10: $cache(v, f) \leftarrow \alpha$
11: return $\alpha$
```

Proposition 1 *The interaction function $f_{\mathbf{XY}}$ satisfies the following properties:*

1. $f = (\exists \mathbf{X} f) \wedge (\exists \mathbf{Y} f) \wedge f_{\mathbf{XY}}$
2. $\exists \mathbf{X} f_{\mathbf{XY}} = true$
3. $\exists \mathbf{Y} f_{\mathbf{XY}} = true$
4. *$f_{\mathbf{XY}}$ is the weakest function satisfying the above properties.*[3]

Moreover, $f_{\mathbf{XY}} \neq false$ and if $f = false$ or $f = true$, then $f_{\mathbf{XY}} = true$.

Note here that every instantiation $\mathbf{x}$ is consistent with $f_{\mathbf{XY}}$ and every instantiation $\mathbf{y}$ is consistent with $f_{\mathbf{XY}}$, because $f_{\mathbf{XY}}$ puts no restriction on variables $\mathbf{X}$ or on variables $\mathbf{Y}$ per se. We illustrate these properties with the following examples.

Consider the CNF $f = (\neg A \vee B) \wedge (\neg B \vee C) \wedge (\neg C \vee D)$ and let $\mathbf{X} = \{A, B\}$ and $\mathbf{Y} = \{C, D\}$. One can verify that

$$\begin{aligned} \exists \mathbf{X} f &= (\neg C \vee D) \\ \exists \mathbf{Y} f &= (\neg A \vee B) \\ f_{\mathbf{XY}} &= (\neg B \vee C) \end{aligned}$$

Consider the CNF $f = (\neg A \vee B) \wedge (A \vee C)$ and let $\mathbf{X} = \{A\}$ and $\mathbf{Y} = \{B, C\}$. One can verify that

$$\begin{aligned} \exists \mathbf{X} f &= (B \vee C) \\ \exists \mathbf{Y} f &= true \\ f_{\mathbf{XY}} &= (\neg A \vee \neg C \vee B) \wedge (A \vee \neg B \vee C) \end{aligned}$$

Notice that in the first example above, the interaction function contains exactly the clause that mention both variables in $\mathbf{X}$ and $\mathbf{Y}$. However, as illustrated by the second example, in general, the interaction function of a CNF may not correspond to any subset of clauses from the original CNF.

We now present a key result that relates decompositions of interaction functions to those of the original functions. Because the interaction function captures the relationship between $\mathbf{X}$ and $\mathbf{Y}$, it should come as no surprise that the size of the minimal $\mathbf{X}$-decomposition of the interaction function is roughly the same as the size of the minimal $\mathbf{X}$-decomposition of the function itself.

[3] Any function with any additional model will fail to satisfy these properties.

Proposition 2 *Let k be the size of a minimal $\mathbf{X}$-decomposition for function $f(\mathbf{Z})$ and let $k_{\mathbf{XY}}$ be the size of a minimal $\mathbf{X}$-decomposition for interaction function $f_{\mathbf{XY}}$. Then $k \leq k_{\mathbf{XY}} \leq k + 2$.*

This is an important result because it shows us that being able to find a compact decomposition of the interaction function is just as good as the ability to find a compact decomposition for the function itself. This result does not make any assumptions on the specific syntax of any of the involved functions. However, the proof of the result is constructive, showing how one can convert a decomposition of function f into a decomposition of interaction function $f_{\mathbf{XY}}$ and vice versa.

5 CNF Decompositions

One is typically interested in decomposing functions that are represented as CNFs. We will next review one of the key methods for decomposing CNFs, which is usually utilized by knowledge compilers (e.g., [5]), and show that it can be viewed as working with an approximation of the interaction function. We will also show that the method could be far from being optimal, while pinpointing the specific reason for this lack of optimality.

Consider a CNF Δ and suppose that we partition its clauses into two sets Δ^l and Δ^r and then consider the variables $\mathbf{V}$ shared between these sets. The CNF is then conditioned on each instantiation $\mathbf{v}$ of these variables, which leads to disconnecting the components Δ^l and Δ^r from each other since $\Delta^l|\mathbf{v}$ and $\Delta^r|\mathbf{v}$ will no longer share variables. The decomposition computed will then be:

$$(\Delta^l|\mathbf{v}^1) \wedge (\Delta^r|\mathbf{v}^1) \wedge \mathbf{v}^1, \ldots, (\Delta^l|\mathbf{v}^m) \wedge (\Delta^r|\mathbf{v}^m) \wedge \mathbf{v}^m, \quad (1)$$

where $\mathbf{v}^1, \ldots, \mathbf{v}^m$ are the instantiations of variables $\mathbf{V}$. This CNF decomposition method has been proposed more than a decade ago [3] and has been used extensively since then. We will later discuss some variations on this method, but our goal next is to show its relation to interaction functions.

Proposition 3 *Consider CNF $\Delta(\mathbf{Z})$ and any clause partition (Δ^l, Δ^r). Let $(\mathbf{X}, \mathbf{Y})$ be any partition of variables $\mathbf{Z}$ such that $\mathbf{X} \subseteq vars(\Delta^l)$ and $\mathbf{Y} \subseteq vars(\Delta^r)$. The decomposition given in (1) is then an $\mathbf{X}$-decomposition of Δ.*

Note that the CNF decomposition method does not explicitly target a particular variable partition $(\mathbf{X}, \mathbf{Y})$. Instead, these variable partitions are determined mainly by the chosen clause partition. Still, the outcome is a decomposition in the sense of Definition 1.

We will next show that conditioning on variables $\mathbf{V}$, as done above, is meant to eliminate the interaction between variables $\mathbf{X}$ and $\mathbf{Y}$ in a very specific way and then relate this to decomposing the interaction function. Let $\Delta(\mathbf{X})$ be the clauses that mention only variables $\mathbf{X}$, $\Delta(\mathbf{Y})$ be the clauses that mention only variables $\mathbf{Y}$, and $\Delta(\mathbf{X}, \mathbf{Y})$ be the remaining clauses. We can then write CNF Δ as follows:

$$\Delta(\mathbf{Z}) = \Delta(\mathbf{X}) \wedge \Delta(\mathbf{X}, \mathbf{Y}) \wedge \Delta(\mathbf{Y}).$$

We will refer to $\Delta(\mathbf{X}, \mathbf{Y})$ as the *cutset clauses* and show later that they are related in a very specific way to the $(\mathbf{X}, \mathbf{Y})$ interaction function. But first the following result.

Proposition 4 *Given a CNF $\Delta(\mathbf{Z})$ and any clause partition (Δ^l, Δ^r), let $\mathbf{V}$ be the set of variables shared between Δ^l, Δ^r and $(\mathbf{X}, \mathbf{Y})$ be any partition of $\mathbf{Z}$ such that $\mathbf{X} \subseteq vars(\Delta^l)$ and $\mathbf{Y} \subseteq vars(\Delta^r)$. For every clause α in $\Delta(\mathbf{X}, \mathbf{Y})$ and any instantiation $\mathbf{v}$ of $\mathbf{V}$, we have that $vars(\alpha|\mathbf{v}) \subseteq \mathbf{X}$ or $vars(\alpha|\mathbf{v}) \subseteq \mathbf{Y}$.*

Hence, the process of conditioning on variables $\mathbf{V}$ can be interpreted as a process of eliminating the interaction between variables $\mathbf{X}$ and variables $\mathbf{Y}$ since this conditioning removes all cutset clauses. Once all these interactions have been eliminated, the resulting formula is guaranteed to be decomposable.

One of the key results we shall present later shows that the dependence on cutset clauses in characterizing the interaction between two sets of variables, and the dependence on these clauses in computing decompositions, can be an overkill as these clauses may not faithfully capture the interaction between the corresponding variables. Before we present this result, however, we discuss another method for decomposing a CNF based on cutset clauses.

Consider cutset clauses $\alpha_1, \ldots, \alpha_k$ for variable partition $(\mathbf{X}, \mathbf{Y})$ and let us express clause α_i as $\alpha_i(\mathbf{X}) \vee \alpha_i(\mathbf{Y})$, where $\alpha_i(\mathbf{X})$ is the sub-clause of α_i mentioning variables $\mathbf{X}$ and $\alpha_i(\mathbf{Y})$ is the sub-clause of α_i mentioning variables in $\mathbf{Y}$. We then have the following decomposition of CNF Δ:

$$\left\{ \left(\Delta(\mathbf{X}) \wedge \bigwedge_{i \in S} \alpha_i(\mathbf{X}) \right) \wedge \left(\Delta(\mathbf{Y}) \wedge \bigwedge_{i \in \overline{S}} \alpha_i(\mathbf{Y}) \right) \middle| S \subseteq \{1, 2, \ldots, k\} \right\}.$$

We can now establish the following upper bounds on the size of CNF decompositions computed using the above two methods.

Proposition 5 *Let $f(\mathbf{Z})$ be a CNF, let $(\mathbf{X}, \mathbf{Y})$ be a partition of variables $\mathbf{Z}$, and let g be the cutset clauses of CNF f. Let k_f and k_g be the sizes of minimal $\mathbf{X}$-decompositions for f and g, respectively, and let k_c and k_v be the number of clauses and variables in g, respectively. Then*

$$k_f \leq k_g \leq 2^{\min(k_v, k_c)}.$$

Consider the following CNF with $k_c = O(n)$ and $k_v = O(n)$:

$$\begin{array}{l} (A_1 \vee B_1), \\ (A_1 \vee A_2 \vee B_2), \\ (A_1 \vee A_2 \vee A_3 \vee B_3), \\ \vdots \\ (A_1 \vee \ldots \vee A_n \vee B_n), \end{array}$$

Let $\mathbf{X} = \{A_1, \ldots, A_n\}$ and $\mathbf{Y} = \{B_1, \ldots, B_n\}$. The above methods (and corresponding bounds) lead to decompositions of exponential size. Yet, we have an $\mathbf{X}$-decomposition of size $O(n)$:

$g(\mathbf{X})$	$h(\mathbf{Y})$
A_1	*true*
$(A_1 \vee A_2)$	B_1
$(A_1 \vee A_2 \vee A_3)$	$B_1 \wedge B_2$
$\vdots$	$\vdots$
$(A_1 \vee \ldots \vee A_n)$	$B_1 \wedge \ldots \wedge B_n$

This example shows that the specific decomposition schemes we discussed, based on cutset clauses, could yield decompositions that are much larger than the optimal one. In fact, this sub-optimality turns out to go well beyond the specific decomposition schemes discussed here. The following example shows that even the smallest $\mathbf{X}$-decomposition of the cutset clauses (obtained by *any* method) could be exponentially larger than the smallest $\mathbf{X}$-decomposition of the entire CNF. Consider the following CNF $\Delta(\mathbf{Z})$:

$$X_1 \wedge \ldots \wedge X_n \wedge$$
$$(\neg X_1 \vee Y_1) \wedge \ldots \wedge (\neg X_n \vee Y_n)$$

If we let $\mathbf{X} = \{X_1, \ldots, X_n\}$ and $\mathbf{Y} = \{Y_1, \ldots, Y_n\}$, then we have

$$\Delta(\mathbf{X}) = X_1 \wedge \ldots \wedge X_n, \; \Delta(\mathbf{Y}) = \textit{true},$$
$$\Delta(\mathbf{X}, \mathbf{Y}) = (\neg X_1 \vee Y_1) \wedge \ldots \wedge (\neg X_n \vee Y_n).$$

Note that this CNF is logically equivalent to $X_1 \wedge \ldots \wedge X_n \wedge Y_1 \wedge \ldots \wedge Y_n$, which is already decomposable according to the partition $(\mathbf{X}, \mathbf{Y})$. Hence, its minimal decomposition has size 1. Yet, we can show that the minimal $\mathbf{X}$-decomposition for the cutset clauses $\Delta(\mathbf{X}, \mathbf{Y})$ has size exponential in n (proof in the full paper). Note also that the interaction function for this CNF is *true*.

6 The Relationship between Cutset Clauses and Interaction Functions

We have thus far presented two methods that try to eliminate the interaction between two sets of variables by effectively eliminating cutset clauses. We have also shown that these methods can lead to decompositions that are exponentially larger in size than optimal decompositions. We have also shown that the optimal decomposition of cutset clauses (using any method) can be exponentially larger than the optimal decomposition of the whole CNF.

Earlier, however, we have shown that this cannot happen when using interaction functions. In particular, we have shown that the size of an optimal decomposition of a function is roughly equal to the size of an optimal decomposition of its interaction function. We will next show the relationship between cutset clauses and the interaction function of a CNF. The goal here is to show what is exactly missing when using cutset clauses, which could lead to sub-optimal decompositions.

Proposition 6 *Consider a CNF $f(\mathbf{Z})$ and let $(\mathbf{X}, \mathbf{Y})$ be a partition of variables $\mathbf{Z}$. Let $g(\mathbf{X})$ be the clauses of the CNF mentioning only variables in $\mathbf{X}$, $h(\mathbf{Y})$ be the clauses of the CNF mentioning only variables in $\mathbf{Y}$, and $e(\mathbf{X}, \mathbf{Y})$ be the cutset clauses. We then have:*

$$f(\mathbf{Z}) = g(\mathbf{X}) \wedge h(\mathbf{Y}) \wedge e(\mathbf{X}, \mathbf{Y}).$$

Moreover, the $(\mathbf{X}, \mathbf{Y})$ interaction function is given by:

$$f_{\mathbf{XY}} \;=\; e \vee \neg(h \wedge \exists \mathbf{X}(g \wedge e)) \vee \neg(g \wedge \exists \mathbf{Y}(h \wedge e)).$$

This proposition spells out precisely the difference between cutset clauses and interaction functions. In particular, while we have shown earlier that an optimal decomposition for cutset clauses e can be exponentially larger than an optimal decomposition for CNF f, the above proposition (together with Proposition 2) show that the size of an optimal decomposition for $e \vee \neg(h \wedge \exists \mathbf{X}(g \wedge e)) \vee \neg(g \wedge \exists \mathbf{Y}(h \wedge e))$ is at most off by 2 from the size of an optimal decomposition for the CNF f.[4] Hence, while current decomposition methods consider only the cutset clauses e, one also needs to account for the components $\neg(h \wedge \exists \mathbf{X}(g \wedge e))$ and $\neg(g \wedge \exists \mathbf{Y}(h \wedge e))$, which may lead to a decomposition whose size is exponentially smaller.

Consider the following CNF for an example:

$$f(\mathbf{Z}) = X_2 \wedge X_3 \wedge \ldots \wedge X_n \wedge (\neg X_1 \vee Y_1) \wedge \ldots \wedge (\neg X_n \vee Y_n).$$

If $\mathbf{X} = \{X_1, \ldots, X_n\}$, $\mathbf{Y} = \{Y_1, \ldots, Y_n\}$, the cutset clauses are

$$e(\mathbf{X}, \mathbf{Y}) = (\neg X_1 \vee Y_1) \wedge \ldots \wedge (\neg X_n \vee Y_n).$$

[4] Note that $\neg(h \wedge \exists \mathbf{X}(g \wedge e))$ depends only on variables $\mathbf{Y}$ and $\neg(g \wedge \exists \mathbf{Y}(h \wedge e))$ depends only on variables $\mathbf{X}$.

Moreover, we have $g(\mathbf{X}) = X_2 \wedge \ldots \wedge X_n$ and $h(\mathbf{Y}) = true$. One can verify that $\neg(h \wedge \exists \mathbf{X}(g \wedge e)) = (\neg Y_2 \vee \ldots \vee \neg Y_n)$ and that $\neg(g \wedge \exists \mathbf{Y}(h \wedge e)) = (\neg X_2 \vee \ldots \vee \neg X_n)$. The interaction function will then simplify to

$$f_{\mathbf{XY}} = (\neg X_1 \vee \ldots \vee \neg X_n) \vee (Y_1 \vee \neg Y_2 \vee \ldots \vee \neg Y_n).$$

Even though the cutset clauses $e(\mathbf{X}, \mathbf{Y})$ admit no polysize $\mathbf{X}$-decomposition (proof in full paper), the interaction function has an $\mathbf{X}$-decomposition of size $2n$.

7 Computing Interaction Functions

Our results thus far show that dependence on cutset clauses in computing decompositions could lead to sub-optiomal decompositions. Our results also show that using interaction functions, which are weaker than cutset clauses, do not suffer from this problem. This leaves the question, however, of how to practically compute interaction functions, which are not readily available (as they do not necessarily correspond to a subset of the CNF clauses). We do not have an answer to this question in this paper and we view this as an important problem to be addressed in future research. In the short term, however, one may not want to completely compute and use interaction functions, but one may only seek to find better approximations of this function than the cutset clauses. In fact, some knowledge compilers and model counters can be viewed as doing just that. For example, the C2D compiler [5], will instantiate variables incrementally, apply unit propagation (to the whole CNF), and then simplify clauses, before continuing to instantiate more variables. In a sense, this can be viewed as integrating (even in a limited way) the other missing components of the interaction function. Our results, however, pinpoint precisely and semantically the CNF contents which are relevant, and which must be brought to bear on the decomposition process if one is to guarantee optimality.

In the following experiments, we set out to shed some light on the distinctions between cutset clauses and interaction functions. In particular, we implemented a method for computing the OBDD representations of interaction functions based on the formula given in Proposition 6. We will compare these OBDDs against the OBDDs for cutset clauses. One reason for choosing OBDD for this comparison is because of its canonicity–the size of the OBDD depends only on the underlying boolean function and the variable order used. Moreover, representing these functions using OBDDs also allow us to easily compare their model counts. Lastly, the OBDDs of these functions also provide us with useful insights on certain types of decompositions that they admit. This is mainly due to a result in a recent work [10]. In that work, it was shown that, given a variable ordering that puts $\mathbf{X}$ before $\mathbf{Y}$, any OBDD of function f respecting that ordering *must* induce an $\mathbf{X}$-decomposition. Moreover, the size of the induced $\mathbf{X}$-decomposition is equal to the number of OBDD nodes labeled with a variable in $\mathbf{Y}$ that are pointed to by some nodes labeled with a variable in $\mathbf{X}$. We refer to these nodes as *decomposition-nodes* in the following discussion.

In this preliminary experiment, we used more than 1,000 randomly-generated, satisfiable 3CNF formulas over 20 variables. The number of clauses in these formulas ranges from 40 to 100 clauses. For each CNF formula, we generated a random variable partition, and computed the OBDDs for the cutset clauses and the interaction function (using CUDD [12]) with respect to that partition. For a partition $(\mathbf{X}, \mathbf{Y})$, we used the variable ordering $\langle \mathbf{X}, \mathbf{Y} \rangle$ for constructing OBDDs.[5] We used the natural order within sets $\mathbf{X}, \mathbf{Y}$. Table 1 reports the sizes and model counts for representative instances.

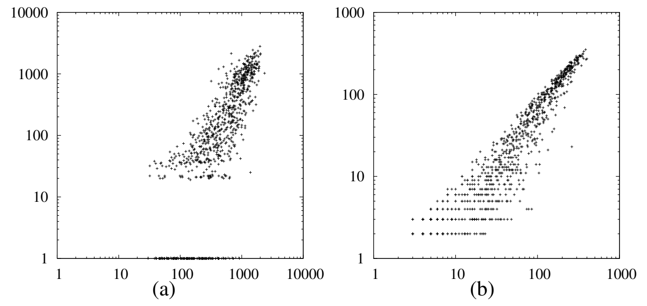

Figure 2. Scatter plots comparing (a) OBDD sizes and (b) number of decomposition-nodes for cutset clauses (x-axis) against those for interaction functions (y-axis).

vars	clauses	partition	cutset clauses		interaction	
		sizes	OBDD nodes	models	OBDD nodes	models
20	40	10|10	1,611	30,890	1,448	968,189
20	40	9|11	901	60,684	908	1,010,684
20	50	8|12	764	20,664	412	1,042,563
20	50	9|11	1,486	43,505	748	1,038,957
20	60	7|13	630	13,834	230	1,048,230
20	60	8|12	1,396	12,434	347	1,045,503
20	70	10|10	646	830	304	1,046,941
20	70	10|10	528	507	349	1,046,740
20	80	8|12	314	1,020	34	1,048,560
20	80	9|11	400	926	75	1,048,542
20	90	9|11	836	1,754	190	1,048,179
20	90	10|10	530	647	184	1,045,774
20	100	10|10	246	246	37	1,048,570
20	100	8|12	207	93	38	1,048,568

Table 1. Properties of cutset clauses and interaction functions for representative instances.

On average, the interaction functions tend to have several orders of magnitude more models than cutset clauses. Hence, these results show that interaction functions tend to be much weaker than cutset clauses. Nevertheless, their OBDDs tend to be smaller. On average, the size of the OBDD of interaction is about 54% smaller than that of the corresponding cutset clauses. Figure 2(a) shows a scatter plot that compares the OBDD sizes of cutset cluases (x-axis) and corresponding interaction functions (y-axis). Each data point in this plot corresponds to one instance. Each data point below the $x = y$ diagonal line indicates that the interaction function for the CNF has a smaller OBDD representation.

In terms of decomposition-nodes, the decompositions induced by the OBDDs of interaction functions are 45% smaller than those of cutset clauses on average. Figure 2(b) compares the number of decomposition-nodes in these OBDDs in the same manner. In fact, the number of decomposition-nodes in the OBDDs of interaction functions is almost always ($> 99\%$) smaller than that of cutset clauses. Even though we only experimented with random formulas, the results are promising as they clearly indicate that interaction functions tend to lead to smaller decompositions than cutset clauses.

[5] Using this ordering allows us to measure the sizes of $\mathbf{X}$-decompositions induced by these OBDDs as explained above.

8 Conclusions

We discussed in this paper the problem of computing optimal decompositions for Boolean functions — those represented using CNFs in particular. We introduced the notion of an interaction function between two sets of variables, which captures the relationship between these variables as encoded by the Boolean function. We showed that optimally decomposing the interaction function is equivalent to optimally decomposing the full Boolean function. We also showed that the current practice of computing decompositions based on cutset clauses can lead to decompositions that are exponentially larger than optimal decompositions. We also provided precise relationships between cutset clauses and interaction functions, showing that cutset clauses can be viewed as approximations of interaction functions. Our results pinpoint why decomposition methods based on cutset clauses are sub-optimal and identify what else needs to be accounted for by these methods to guarantee the optimality of computed decompositions.

Acknowledgment
We thank Jinbo Huang for providing a program for converting CNFs into OBDDs.

A Proofs

Proof of Proposition 1

By the definition of interaction function, we have

$$\begin{aligned}&(\exists \mathbf{X} f) \wedge (\exists \mathbf{Y} f) \wedge f_{\mathbf{XY}}\\ &\quad = \quad (\exists \mathbf{X} f) \wedge (\exists \mathbf{Y} f) \wedge (f \vee \neg(\exists \mathbf{X} f) \vee \neg(\exists \mathbf{Y} f))\\ &\quad = \quad (\exists \mathbf{X} f) \wedge (\exists \mathbf{Y} f) \wedge f = f,\end{aligned}$$

since $f \models \exists \mathbf{X} f$ and $f \models \exists \mathbf{Y} f$. We also have

$$\begin{aligned}\exists \mathbf{X} f_{\mathbf{XY}} &= \exists \mathbf{X}\,(f \vee \neg(\exists \mathbf{X} f) \vee \neg(\exists \mathbf{Y} f))\\ &= (\exists \mathbf{X} f) \vee (\exists \mathbf{X} \neg(\exists \mathbf{X} f)) \vee (\exists \mathbf{X} \neg(\exists \mathbf{Y} f))\\ &= (\exists \mathbf{X} f) \vee \mathit{true} \vee (\exists \mathbf{X} \neg(\exists \mathbf{Y} f)) = \mathit{true}.\end{aligned}$$

Similarly, $\exists \mathbf{Y} f_{\mathbf{XY}} = \mathit{true}$. We will next show that $f_{\mathbf{XY}}$ is the weakest function satisfying $f = (\exists \mathbf{X} f) \wedge (\exists \mathbf{Y} f) \wedge f_{\mathbf{XY}}$. That is, any function with one more model will fail to satisfy this property. We first observe that $f \models (\exists \mathbf{X} f) \vee (\exists \mathbf{Y} f)$. Therefore, the models of $\neg f \wedge ((\exists \mathbf{X} f) \vee (\exists \mathbf{Y} f))\quad (*)$ are precisely the models of $(\exists \mathbf{X} f) \vee (\exists \mathbf{Y} f)$ that are not models of f. For a function h to satisfy

$$f = (\exists \mathbf{X} f) \wedge (\exists \mathbf{Y} f) \wedge h$$

it cannot have any of the models of (*). That is, the following must be inconsistent $h \wedge (\neg f \wedge ((\exists \mathbf{X} f) \vee (\exists \mathbf{Y} f)))$. The weakest function that satisfies this property is:

$$\begin{aligned}h &= \neg\,(\neg f \wedge ((\exists \mathbf{X} f) \vee (\exists \mathbf{Y} f)))\\ &= f \vee \neg(\exists \mathbf{X} f) \vee \neg(\exists \mathbf{Y} f).\end{aligned}$$

The following properties follow immediately from the definition of an interaction: $f_{\mathbf{XY}} \neq \mathit{false}$ and if $f = \mathit{false}$ or $f = \mathit{true}$, then $f_{\mathbf{XY}} = \mathit{true}$. □

Proof of Proposition 2

Suppose that $f_{\mathbf{XY}} = \bigvee_{i=1}^{k_{\mathbf{XY}}} g^i(\mathbf{X}) \wedge h^i(\mathbf{Y})$ is a minimal $\mathbf{X}$-decomposition for $f_{\mathbf{XY}}$. We then have

$$\begin{aligned}f &= (\exists \mathbf{Y} f) \wedge (\exists \mathbf{X} f) \wedge f_{\mathbf{XY}}\\ &= (\exists \mathbf{Y} f) \wedge (\exists \mathbf{X} f) \wedge \left(\bigvee_{i=1}^{k_{\mathbf{XY}}} g^i(\mathbf{X}) \wedge h^i(\mathbf{Y})\right)\\ &= \bigvee_{i=1}^{k_{\mathbf{XY}}} \Big((\exists \mathbf{Y} f) \wedge g^i(\mathbf{X})\Big) \wedge \Big((\exists \mathbf{X} f) \wedge h^i(\mathbf{Y})\Big)\end{aligned}$$

which is an $\mathbf{X}$-decomposition for f of size $k_{\mathbf{XY}}$. Hence, $k \leq k_{\mathbf{XY}}$. Suppose now that $f = \bigvee_{i=1}^{k} g^i(\mathbf{X}) \wedge h^i(\mathbf{Y})$ is a minimal $\mathbf{X}$-decomposition for f. Then $\neg(\exists \mathbf{X} f) \vee \neg(\exists \mathbf{Y} f) \vee \bigvee_{i=1}^{k} g^i(\mathbf{X}) \wedge h^i(\mathbf{Y})$ is an $\mathbf{X}$-decomposition for $f_{\mathbf{XY}}$ of size $k+2$. Hence, $k_{\mathbf{XY}} \leq k+2$. □

Proof of Proposition 6

We want to compute $f_{\mathbf{XY}} = f \vee \neg\exists \mathbf{X} f \vee \neg\exists \mathbf{Y} f$ for $f = g(\mathbf{X}) \wedge h(\mathbf{Y}) \wedge e(\mathbf{X}, \mathbf{Y})$. We have

$$\begin{aligned}\exists \mathbf{X} f &= \exists \mathbf{X}(g \wedge h \wedge e)\\ &= h \wedge \exists \mathbf{X}(g \wedge e) \quad (h \text{ does not depend on } \mathbf{X}).\end{aligned}$$

Similarly, we have $\exists \mathbf{Y} f = g \wedge \exists \mathbf{Y}(h \wedge e)$. We then have

$$\begin{aligned}f_{\mathbf{XY}} &= f \vee \neg\exists \mathbf{X} f \vee \neg\exists \mathbf{Y} f\\ &= (g \wedge h \wedge e) \vee \neg h \vee \neg\exists \mathbf{X}(g \wedge e) \vee \neg g \vee \neg\exists \mathbf{Y}(h \wedge e)\\ &= e \vee \neg h \vee \neg\exists \mathbf{X}(g \wedge e) \vee \neg g \vee \neg\exists \mathbf{Y}(h \wedge e)\\ &= e \vee \neg(h \wedge \exists \mathbf{X}(g \wedge e)) \vee \neg(g \wedge \exists \mathbf{Y}(h \wedge e)).\end{aligned}$$

□

REFERENCES

[1] Roberto J. Bayardo, Jr. and Joseph Daniel Pehoushek, 'Counting models using connected components', in *Proc. of the AAAI-00*, pp. 157–162. AAAI Press / The MIT Press, (2000).
[2] Elazar Birnbaum and Eliezer L. Lozinskii, 'The good old davis-putnam procedure helps counting models', *J. Artif. Int. Res.*, **10**(1), 457–477, (1999).
[3] Adnan Darwiche, 'Compiling knowledge into decomposable negation normal form', in *Proc. of IJCAI-99*, pp. 284–289. Morgan Kaufmann, (1999).
[4] Adnan Darwiche, 'Decomposable negation normal form', *Journal of the ACM*, **48**(4), 608–647, (2001).
[5] Adnan Darwiche, 'New advances in compiling CNF to decomposable negational normal form', in *Proceedings of European Conference on Artificial Intelligence, Valencia, Spain*, pp. 328–332, (2004).
[6] Adnan Darwiche and Pierre Marquis, 'A knowledge compilation map', *JAIR*, **17**, 229–264, (2002).
[7] Jinbo Huang and Adnan Darwiche, 'Using dpll for efficient obdd construction', in *Proc. of SAT-04 (Selected Papers)*, pp. 157–172, (2004).
[8] Robert Mateescu and Rina Dechter, 'Compiling constraint networks into and/or multi-valued decision diagrams (AOMDDs)', in *Proc. of CP-06*, pp. 329–343, (2006).
[9] Knot Pipatsrisawat and Adnan Darwiche, 'New compilation languages based on structured decomposability', in *Proc. of AAAI-08*, pp. 517–522, (2008).
[10] Knot Pipatsrisawat and Adnan Darwiche, 'A lower bound on the size of decomposable negation normal form', in *Proceedings of AAAI-10, to appear*, (July 2010).
[11] Tian Sang, Fahiem Bacchus, Paul Beame, Henry Kautz, and Toniann Pitassi, 'Combining component caching and clause learning for effective model counting', in *In Proc. of SAT-04*, (2004).
[12] F. Somenzi. Cudd: Cu decision diagram package. available from http://vlsi.colorado.edu/~fabio/.