Bi-Decomposing Large Boolean Functions via Interpolation and Satisfiability Solving*

Ruei-Rung Lee, Jie-Hong R. Jiang, and Wei-Lun Hung

Department of Electrical Engineering/Graduate Institute of Electronics Engineering
National Taiwan University, Taipei 10617, Taiwan

ABSTRACT

Boolean function bi-decomposition is a fundamental operation in logic synthesis. A function $f(X)$ is bi-decomposable under a variable partition X_A, X_B, X_C on X if it can be written as $h(f_A(X_A, X_C), f_B(X_B, X_C))$ for some functions h, f_A, and f_B. The quality of a bi-decomposition is mainly determined by its variable partition. A preferred decomposition is disjoint, i.e. $X_C = \emptyset$, and balanced, i.e. $|X_A| \approx |X_B|$. Finding such a good decomposition reduces communication and circuit complexity, and yields simple physical design solutions. Prior BDD-based methods may not be scalable to decompose large functions due to the memory explosion problem. Also as decomposability is checked under a fixed variable partition, searching a good or feasible partition may run through costly enumeration that requires separate and independent decomposability checkings. This paper proposes a solution to these difficulties using interpolation and incremental SAT solving. Preliminary experimental results show that the capacity of bi-decomposition can be scaled up substantially to handle large designs.

Categories and Subject Descriptors

B.6.3 [**Logic Design**]: Design Aids—*automatic synthesis*

General Terms

logic synthesis, algorithms

Keywords

bi-decomposition, variable partition, satisfiability, Craig interpolation

1. INTRODUCTION

Functional decomposition [1, 6] is a fundamental operation on Boolean functions that decomposes a large function f on variables X into a set of small subfunctions $h, g_1, \ldots, g_m$ with $f(X) = h(g_1(X), \ldots, g_m(X))$, often $m < |X|$. It plays a pivotal role in the study of circuit and communication complexity, and has important applications on multi-level and FPGA logic synthesis. Extensive research has been published on this subject, see e.g. [15] for an introduction.

When $m = 2$, the decomposition is known as *bi-decomposition* [3, 14, 16, 11, 4], the simplest nontrivial case, yet the most widely applied since a logic netlist is often expressed as a network of two-fanin gates. A primary issue of bi-decomposition is *variable partition*. For $f(X) = h(f_A(X_A, X_C), f_B(X_B, X_C))$, the variable partition $\{X_A, X_B, X_C\}$ on X (i.e. X_A, X_B, X_C are pairwise disjoint and $X_A \cup X_B \cup X_C = X$) mainly determines the decomposition quality. A particularly desirable bi-decomposition is *disjoint*, i.e., $|X_C| = 0$, and *balanced*, i.e., $|X_A| \approx |X_B|$. An ideal bi-decomposition reduces circuit and communication complexity, and in turn simplifies physical design. Effective approaches to bi-decomposition can be important not only in large-scale circuit minimization, but also in early design closure if combined well with physical design partitioning.

Modern approaches to bi-decomposition, such as [11], were based on BDD data structure for its powerful capability supporting various Boolean manipulations. They are however not scalable to handle large Boolean functions due to the common memory explosion problem. Furthermore, the variable partition problem can not be solved effectively. Because decomposability is checked under a fixed variable partition, searching a good or even feasible partition may run through costly enumeration that requires separate and independent decomposability checkings.

To overcome these limitations, this paper proposes a solution based on satisfiability (SAT) solving. The formulation is motivated by the recent work [9], where a SAT-based formulation made the computation of functional dependency scalable to large designs. Our main results include 1) a pure SAT-based solution to bi-decomposition, 2) subfunction derivation by interpolation and cofactoring, and 3) automatic variable partitioning by incremental solving under unit assumptions. Thereby the scalability of bi-decomposition and the optimality of variable partition can be substantially improved. Experiments show promising results on the scalability of bi-decomposition and the optimality of variable partitioning.

In comparison with the closest work [11], aside from the scalability and variable partitioning issues, this paper focuses on *strong decomposition* (namely, X_A and X_B cannot be empty), whereas [11] gave a more general approach allow-

*This work was supported in part by NSC grants 95-2218-E-002-064-MY3 and 96-2221-E-002-278-MY3.

DAC 2008, June 8–13, 2008, Anaheim, California, USA.

ing *weak decomposition* (namely, X_A or X_B can be empty). Moreover, as don't cares are better handled in BDD than in SAT, they were exploited in [11] for logic sharing.

This paper is organized as follows. Section 2 gives the preliminaries. Our main theoretical results are presented in Section 3, and implementation issues are discussed in Section 4. The proposed methods are evaluated with experimental results in Section 5. Finally Section 6 concludes the paper and outlines future work.

2. PRELIMINARIES

As conventional notation, sets are denoted in upper-case letters, e.g. S; set elements are in lower-case letters, e.g. $e \in S$. The cardinality of S is denoted as $|S|$. A partition of a set S into $S_i \subseteq S$ for $i = 1, \ldots, k$ (with $S_i \cap S_j = \emptyset, i \neq j$, and $\bigcup_i S_i = S$) is denoted as $\{S_1|S_2|\ldots|S_k\}$. For a set X of Boolean variables, its set of valuations (or truth assignments) is denoted as $[\![X]\!]$, e.g., $[\![X]\!] = \{(0,0),(0,1),(1,0),(1,1)\}$ for $X = \{x_1, x_2\}$.

2.1 Bi-Decomposition

DEFINITION 1. *Given a completely specified Boolean function f, variable x is a* support variable *of f if $f_x \neq f_{\neg x}$, where f_x and $f_{\neg x}$ are the positive and negative cofactors of f on x, respectively.*

DEFINITION 2. *An* incompletely specified function *F is a 3-tuple (q, d, r), where the completely specified functions q, d, and r represent the onset, don't-care set and offset functions, respectively.*

DEFINITION 3. *A completely specified function $f(X)$ is* $\langle$OP$\rangle$ bi-decomposable*, or simply* $\langle$OP$\rangle$-decomposable*, under variable partition $X = \{X_A|X_B|X_C\}$ if it can be written as $f(X) = f_A(X_A, X_C)$ $\langle$OP$\rangle$ $f_B(X_B, X_C)$, where $\langle$OP$\rangle$ is some binary operator. The decomposition is called* disjoint *if $X_C = \emptyset$, and* non-disjoint *otherwise.*

Note that bi-decomposition trivially holds if $X_A \cup X_C$ or $X_B \cup X_C$ equals X. The corresponding variable partition is called *trivial*. We are concerned about non-trivial bi-decomposition. In the sequel, a binary operator $\langle$OP$\rangle$ can be OR2, AND2, and XOR. Essentially OR2-, AND2-, and XOR-decompositions form the basis of all types of bi-decompositions because any bi-decomposition is simply one of the three cases with some proper complementation on f, f_A and/or f_B.

2.2 Propositional Satisfiability

Let $V = \{v_1, \ldots, v_k\}$ be a finite set of Boolean variables. A *literal* l is either a Boolean variable v_i or its negation $\neg v_i$. A *clause* c is a disjunction of literals. Without loss of generality, we shall assume there is no repeated or complementary literals appearing in the same clause. A *SAT instance* is a conjunction of clauses, i.e., in the so-called *conjunctive normal form* (CNF). In the sequel, a clause set $C = \{c_1, \ldots, c_k\}$ shall mean to be the CNF formula $c_1 \wedge \cdots \wedge c_k$. An *assignment* over V gives every variable v_i a Boolean value either 0 or 1. A SAT instance is *satisfiable* if there exists a satisfying assignment such that the CNF formula evaluates to 1. Otherwise it is *unsatisfiable*.

2.2.1 Refutation Proof and Craig's Interpolation

DEFINITION 4. *Assume literal v is in clause c_1 and $\neg v$ in c_2. A* resolution *of clauses c_1 and c_2 on variable v yields a new clause c containing all literals in c_1 and c_2 except for v and $\neg v$. The clause c is called the* resolvent *of c_1 and c_2, and variable v the* pivot variable*.*

THEOREM 1. *[13] For an unsatisfiable SAT instance, there exists a sequence of resolution steps leading to an empty clause.*

Often only a subset of the clauses of a SAT instance participates in the resolution steps leading to an empty clause.

DEFINITION 5. *A* refutation proof Π *of an unsatisfiable SAT instance C is a directed acyclic graph (DAG) $\Gamma = (N, A)$, where every node in N represents a clause which is either a root clause in C or a resolvent clause having exactly two predecessor nodes, and every arc in A connects a node to its ancestor node. The unique leaf of Π corresponds the empty clause.*

Modern SAT solvers, (e.g., MiniSat [7]) are capable of producing a refutation proof from an unsatisfiable SAT instance.

THEOREM 2 (CRAIG INTERPOLATION THEOREM). *[5] For any two Boolean formulas ϕ_A and ϕ_B with $\phi_A \wedge \phi_B$ unsatisfiable, then there exists a Boolean formula $\phi_{A'}$ referring only to the common input variables of ϕ_A and ϕ_B such that $\phi_A \Rightarrow \phi_{A'}$ and $\phi_{A'} \wedge \phi_B$ is unsatisfiable.*

The Boolean formula $\phi_{A'}$ is referred to as the *interpolant* of ϕ_A and ϕ_B. We shall assume that ϕ_A and ϕ_B are in CNF. So a refutation proof of $\phi_A \wedge \phi_B$ is available from a SAT solver. How to construct an interpolant circuit from a refutation proof in linear time can be found in, e.g., [10].

2.3 Circuit to CNF Conversion

Given a circuit netlist, it can be converted to a CNF formula in such a way that the satisfiability is preserved [17]. The conversion is achievable in linear time by introducing extra intermediate variables. In the sequel, we shall assume that the clause set of a Boolean formula ϕ (similarly $\neg\phi$) is available from such conversion.

3. MAIN CONSTRUCTS

3.1 OR Bi-decomposition

We show that OR2-decomposition can be achieved using SAT solving. Whenever a non-trivial OR2-decomposition exists, we obtain a feasible variable partition and the corresponding subfunctions f_A and f_B.

3.1.1 Decomposition of Completely Specified Functions

Decomposition with known variable partition

Given a function $f(X)$ and a non-trivial variable partition $X = \{X_A|X_B|X_C\}$, we study if f can be expressed as $f_A(X_A, X_C) \vee f_B(X_B, X_C)$ for some functions f_A and f_B. The following proposition lays the foundation of OR2-decomposition.

PROPOSITION 1. *[11] A completely specified function $f(X)$ can be written as $f_A(X_A, X_C) \vee f_B(X_B, X_C)$ for some functions f_A and f_B if and only if the quantified Boolean formula*

$$f(X) \wedge \exists X_A.\neg f(X) \wedge \exists X_B.\neg f(X) \quad (1)$$

is unsatisfiable.

It can be restated as follows.

PROPOSITION 2. *A completely specified function $f(X)$ can be written as $f_A(X_A, X_C) \vee f_B(X_B, X_C)$ for some functions f_A and f_B if and only if the Boolean formula*

$$f(X_A, X_B, X_C) \wedge \neg f(X'_A, X_B, X_C) \wedge \neg f(X_A, X'_B, X_C) \quad (2)$$

is unsatisfiable, where variable set Y' is an instantiated version of variable set Y.

By renaming quantified variables, the quantifiers of Formula (1) can be removed. That is, Formula (1) can be rewritten as the quantifier-free formula of (2) because existential quantification is implicit in satisfiability checking. Note that the complementations in Formulas (1) and (2) need not be computed. Rather, the complementations can be achieved by adding inverters in the corresponding circuit before circuit-to-CNF conversion, or alternatively by asserting the corresponding variables to be false in SAT solving.

A remaining problem to be resolved is how to derive f_A and f_B. We show that they can be obtained through interpolation from a refutation proof of Formula (2). Consider partitioning the clause set of Formula (2) into two subsets C_A and C_B with C_A the clause set of

$$f(X_A, X_B, X_C) \wedge \neg f(X'_A, X_B, X_C) \quad (3)$$

and C_B the clause set of

$$\neg f(X_A, X'_B, X_C). \quad (4)$$

Then the corresponding interpolant corresponds to an implementation of f_A. On the other hand, to derive f_B we perform a similar computation, but now with C_A the clause set of

$$f(X_A, X_B, X_C) \wedge \neg f_A(X_A, X_C) \quad (5)$$

and C_B the clause set of

$$\neg f(X'_A, X_B, X_C). \quad (6)$$

Then the corresponding interpolant corresponds to an implementation of f_B.

The following theorem asserts the correctness of the above construction.

THEOREM 3. *For any* OR2*-decomposable function f under variable partition $X = \{X_A|X_B|X_C\}$, we have $f(X) = f_A(X_A, X_C) \vee f_B(X_B, X_C)$ for f_A and f_B derived from the above construction.*

REMARK 1. *An interpolant itself is in fact a netlist composed of* OR2 *and* AND2 *gates [10]. The "bi-decomposed" netlist however may contain some amount of redundancy; moreover variable partitioning is not used in its derivation.*

Decomposition with unknown variable partition

The previous construction assumes that a variable partition $X = \{X_A|X_B|X_C\}$ is given. We further automate variable partition in the derivation of f_A and f_B as follows. For each variable $x_i \in X$, we introduce two control variables α_{x_i} and β_{x_i}. In addition we instantiate variables X into X' and X''. Let C_A be the clause set of

$$f(X) \wedge \neg f(X') \wedge \bigwedge_i ((x_i \equiv x'_i) \vee \alpha_{x_i}) \quad (7)$$

and C_B be the clause set of

$$\neg f(X'') \wedge \bigwedge_i ((x_i \equiv x''_i) \vee \beta_{x_i}), \quad (8)$$

where $x' \in X'$ and $x'' \in X''$ are the instantiated versions of $x \in X$. Observe that $(\alpha_{x_i}, \beta_{x_i}) = (0,0), (0,1), (1,0)$, and $(1,1)$ indicate $x_i \in X_C, x_i \in X_B, x_i \in X_A$, and x_i can be in either of X_A and X_B, respectively.

In SAT solving the conjunction of Formulas (7) and (8), we make *unit assumptions* [7] on the control variables. Under an unsatisfiable unit assumption, the SAT solver will return a final conflict clause consisting of only the control variables. Notice that every literal in the conflict clause is of positive phase because the conflict arises from a subset of the control variables set to 0. It reveals that setting to 0 the control variables present in the conflict clause is sufficient making the whole formula unsatisfiable. Hence setting to 1 the control variables absent from the conflict clause can not affect the unsatisfiability. The more the control variables can be set to 1, the better the bi-decomposition is because $|X_C|$ is smaller. In essence, this final conflict clause indicates a variable partition X_A, X_B, X_C on X. For example, the conflict clause $(\alpha_{x_1} + \beta_{x_1} + \alpha_{x_2} + \beta_{x_3})$ indicates that the unit assumption $\alpha_{x_1} = 0$, $\beta_{x_1} = 0$, $\alpha_{x_2} = 0$, and $\beta_{x_3} = 0$ results in the unsatisfiability. It in turn suggests that $x_1 \in X_C$, $x_2 \in X_B$, and $x_3 \in X_A$.

To see how the new construction works, imagine setting all the control variables to 0. As SAT solvers tend to refer to a small subset of the clauses relevant to a refutation proof, it may return a conflict clause with just a few literals. It in effect conducts a desirable variable partition. This perception, unfortunately, is flawed in that SAT solvers are very likely to return a conflict clause that consists of all the control variables reflecting the trivial variable partition $X_C = X$. In order to avoid trivial variable partitions, we initially specify two distinct variables x_a and x_b to be in X_A and X_B, respectively, and all other variables in X_C, that is, having $(\alpha_{x_a}, \beta_{x_a}) = (1,0)$, $(\alpha_{x_b}, \beta_{x_b}) = (0,1)$, and $(\alpha_{x_i}, \beta_{x_i}) = (0,0)$ for $i \neq a, b$ in the unit assumption. We call such an initial variable partition as a *seed variable partition*. If the conjunction of Formulas (7) and (8) is unsatisfiable under a seed partition, then the corresponding bi-decomposition is successful. As SAT solvers often refer to a small unsatisfiable core, the returned variable partition is desirable because $|X_C|$ tends to be small. Otherwise, if the seed partition fails, we should try another one. For a given function $f(X)$ with $|X| = n$, the existence of non-trivial OR2-decomposition can be checked with at most $(n-1) + \cdots + 1 = n(n-1)/2$ different seed partitions. On the other hand, we may enumerate different variable partitions using different seed partitions to find one that is more balanced and closer to disjoint. Even from a successful seed partition, we may further refine the returned variable partition by reducing the corresponding unsatisfiable core. The process can be iterated until the unsatisfiable core is minimal.

LEMMA 1. *For an unsatisfiable conjunction of Formulas (7) and (8) under a seed variable partition, the final conflict clause contains only the control variables, which indicates a valid non-trivial variable partition.*

Theorem 4 asserts the correctness of the construction.

THEOREM 4. *For any* OR2*-decomposable function f, we have $f(X) = f_A(X_A, X_C) \vee f_B(X_B, X_C)$ for f_A, f_B, and a non-trivial variable partition $X = \{X_A|X_B|X_C\}$ derived from the above construction.*

One might speculate about whether $(\alpha_x, \beta_x) = (1, 1)$ is possible as it tends to suggest that x can be in either of X_A and X_B. To answer this question, we study the condition that x_i can be in either of X_A and X_B.

LEMMA 2. *[14] If f is bi-decomposable under some variable partition, then the cofactors f_x and $f_{\neg x}$ for any variable x are both bi-decomposable under the same variable partition.*

The converse however is not true. The following theorem gives the condition that x can be in either of X_A and X_B.

THEOREM 5. *Let $X = \{X_a|X_b|X_C|\{x\}\}$ for some $x \in X$. A function $f = f_A(X_A, X_C) \vee f_B(X_B, X_C)$ can be bi-decomposed under variable partition $\{X_a \cup \{x\}|X_b|X_C\}$ as well as under variable partition $\{X_a|X_b \cup \{x\}|X_C\}$ if and only if both f_x and $f_{\neg x}$ are themselves* OR2*-decomposable under variable partition $\{X_a|X_b|X_C\}$, and also $(f_x \not\equiv 1 \wedge f_{\neg x} \not\equiv 1) \Rightarrow (f_x \equiv f_{\neg x})$ under every $c \in [\![X_C]\!]$.*

That is, under every $c \in [\![X_C]\!]$ either x is not a support variable of f, or f_x or $f_{\neg x}$ equals constant one. It is interesting to note that only the former can make $(\alpha_x, \beta_x) = (1, 1)$. Whenever the latter happens, (α_x, β_x) equals $(0, 1)$, $(1, 0)$, or $(0, 0)$ if the solver is unable to identify a *minimal* unsatisfiable core. To see it, consider $f_x \equiv 1$ (similar for $f_{\neg x} \equiv 1$) and $f_{\neg x} \not\equiv f_x$ under some $c \in [\![X_C]\!]$. If $(\alpha_x, \beta_x) = (1, 1)$, Formula (2) reduces to

$$\begin{aligned}&(\exists x.f(X_a, X_b, c, x)) \wedge \neg(\forall x.f(X'_a, X_b, c, x)) \wedge \\ &\neg(\forall x.f(X_a, X'_b, c, x)) \\ &= 1 \wedge \neg f_{\neg x}(X'_a, X_b, c) \wedge \neg f_{\neg x}(X_a, X'_b, c),\end{aligned}$$

which is satisfiable because $f_{\neg x} \not\equiv 1$ under c. Hence $(\alpha_x, \beta_x) = (1, 1)$ only if x is not a support variable of f.

3.1.2 Decomposition of Incompletely Specified Functions

Proposition 2 can be generalized for incompletely specified functions as follows.

PROPOSITION 3. *Given an incompletely specified function $F = (q, d, r)$, there exists a completely specified function f with $f(X) = f_A(X_A, X_C) \vee f_B(X_B, X_C)$, $q(X) \Rightarrow f(X)$, and $f(X) \Rightarrow \neg r(X)$ if and only if the Boolean formula*

$$q(X_A, X_B, X_C) \wedge r(X'_A, X_B, X_C) \wedge r(X_A, X'_B, X_C) \qquad (9)$$

is unsatisfiable.

The derivations of f_A and f_B can be computed in a way similar to the aforementioned construction. We omit the detailed exposition to save space.

3.2 AND Bi-decomposition

PROPOSITION 4. *[14] A function f is* AND2*-decomposable if and only if $\neg f$ is* OR2*-decomposable.*

By decomposing $\neg f$ as $f_A \vee f_B$, we obtain $f = \neg f_A \wedge \neg f_B$. Hence our results on OR2-decomposition are convertible to AND2-decomposition.

3.3 XOR Bi-decomposition

3.3.1 Decomposition of Completely Specified Functions

Decomposition with known variable partition

We formulate the XOR-decomposability in the following proposition, which differs from prior work [16, 8] and is more suitable for SAT solving.

PROPOSITION 5. *A function f can be written as $f(X) = f_A(X_A, X_C) \oplus f_B(X_B, X_C)$ for some functions f_A and f_B under variable partition $X = \{X_A|X_B|X_C\}$ if and only if*

$$\begin{aligned}&(f(X_A, X_B, X_C) \equiv f(X_A, X'_B, X_C)) \wedge \\ &(f(X'_A, X_B, X_C) \not\equiv f(X'_A, X'_B, X_C))\end{aligned} \qquad (10)$$

is unsatisfiable. Furthermore, $f_A = f(X_A, \vec{0}, X_C)$ and $f_B = f(\vec{0}, X_B, X_C) \oplus f(\vec{0}, \vec{0}, X_C)$, or alternatively $f_A = \neg f(X_A, \vec{0}, X_C)$ and $f_B = f(\vec{0}, X_B, X_C) \oplus \neg f(\vec{0}, \vec{0}, X_C)$.

Accordingly interpolation is not needed in computing f_A and f_B in XOR-decomposition.

Decomposition with unknown variable partition

The XOR-decomposition of Proposition 5 assumes a variable partition is given. We further automate variable partition as follows. For each variable $x_i \in X$, we introduce two control variables α_{x_i} and β_{x_i}. In addition we instantiate variables X into X', X'', and X'''. We modify Formula (10) as

$$\begin{aligned}&(f(X) \equiv f(X')) \wedge (f(X'') \not\equiv f(X''')) \wedge \\ &\bigwedge_i (((x_i \equiv x''_i) \wedge (x'_i \equiv x'''_i)) \vee \alpha_{x_i}) \wedge \\ &\bigwedge_i (((x_i \equiv x'_i) \wedge (x''_i \equiv x'''_i)) \vee \beta_{x_i}).\end{aligned} \qquad (11)$$

By Formula (11), an automatic variable partition can be obtained from a seed partition, similar to what we have in OR2-decomposition.

The correctness of the construction is asserted as follows.

THEOREM 6. *For any* XOR*-decomposable function f, we have $f(X) = f_A(X_A, X_C) \oplus f_B(X_B, X_C)$ for f_A, f_B, along with a non-trivial variable partition $X = \{X_A|X_B|X_C\}$ derived from the above construction.*

To see whether $(\alpha_x, \beta_x) = (1, 1)$ is possible or not for some variable x, we study the condition that x can be in either of X_A and X_B.

THEOREM 7. *Let $X = \{X_a|X_b|X_C|\{x\}\}$ for some $x \in X$. A function $f = f_A(X_A, X_C) \oplus f_B(X_B, X_C)$ can be bi-decomposed under variable partition $\{X_a \cup \{x\}|X_b|X_C\}$ as well as under variable partition $\{X_a|X_b \cup \{x\}|X_C\}$ if and only if both f_x and $f_{\neg x}$ are themselves* XOR*-decomposable under variable partition $\{X_a|X_b|X_C\}$, and also $(f_x \equiv f_{\neg x}) \vee (f_x \equiv \neg f_{\neg x})$ under every $c \in [\![X_C]\!]$.*

Under the flexible partition for variable x, Formula (10) reduces to

$$(\exists x.f(X_a, X_b, X_C, x) \equiv \exists x.f(X_a, X'_b, X_C, x)) \wedge (\exists x.f(X'_a, X_b, X_C, x) \not\equiv \exists x.f(X'_a, X'_b, X_C, x)). \quad (12)$$

If $f_x \equiv f_{\neg x}$, the unsatisfiability of Formula (10) implies the unsatisfiability of Formula (12). On the other hand, if $f_x \equiv \neg f_{\neg x}$, $\exists x.f$ is a constant-1 function, and thus Formula (12) is unsatisfiable. Hence $(\alpha_x, \beta_x) = (1, 1)$ is possible even if x is a support variable of f. In this case, we can first decompose f as $f = x \oplus f_{\neg x}$ or equivalently $\neg x \oplus f_x$. Moreover, it can be generalized as follows.

COROLLARY 1. *For an* XOR*-decomposable function f, suppose x_i, for $i = 1, \ldots, k$, are the* support *variables of f with $(\alpha_{x_i}, \beta_{x_i}) = (1, 1)$ after variable partition. Then f can be decomposed as*

$$f = x_1 \oplus \cdots \oplus x_k \oplus f_{\neg x_1 \cdots \neg x_k}. \quad (13)$$

Further, for $X_a = \{x \mid (\alpha_x, \beta_x) = (1, 0)\}$, $X_b = \{x \mid (\alpha_x, \beta_x) = (0, 1)\}$, and $X_C = \{x \mid (\alpha_x, \beta_x) = (0, 0)\}$, then

$$f = x_1 \oplus \cdots \oplus x_k \oplus f_A(X_a, X_C) \oplus f_B(X_b, X_C) \quad (14)$$

with f_A and f_B derived from the previous construction.

4. IMPLEMENTATION ISSUES

When disjoint variable partitioning is concerned, it corresponds to computing a *minimum* unsatisfiable core. Incremental SAT solving is useful in finding a good minimal unsatisfiable core, see e.g. [12]. In our implementation, a variable of X_C is greedily moved to either of X_A and X_B favoring the small one. The process iterates until no more reduction can be made on X_C.

When balanced variable partitioning is concerned, SAT solvers usually tend to make decisions in a descending priority order based on variable IDs. From empirical experience, this bias makes variable partition unbalanced. To overcome, we interleave the variable IDs of X' and those of X'' of Formulas (7) and (8) for OR2- and AND2-decomposition, and interleave those of Formula (11) for XOR-decomposition. For example, assume that variables x'_i, x''_i, x'_{i+1}, and x''_{i+1} are originally of IDs 100, 200, 101, and 201, respectively. We rename them to 100, 200, 201, and 101, respectively. This shuffling makes variable partitioning more balanced.

5. EXPERIMENTAL RESULTS

The algorithms were implemented in C++ in ABC [2] with MiniSAT [7] as the underlying solver. All experiments were conducted on a Linux machine with Xeon 3.4GHz CPU and 6Gb RAM.

Two sets of experiments were designed to demonstrate the scalability of bi-decomposition and the optimality of variable partitioning. Only circuits containing output functions with large support sizes (≥ 30) were chosen from the ISCAS, ITC, and LGSYNTH benchmark suites.[1] To show the efficiency of decomposing large functions, Table 1 shows the results of OR2- and XOR-decompositions on the output functions of the listed circuits. As can be seen, functions with many input variables, such as `i2` and `o64`, can be decomposed effectively.

[1]Sequential circuits are converted to combinational ones by replacing register inputs and outputs with primary outputs and inputs, respectively.

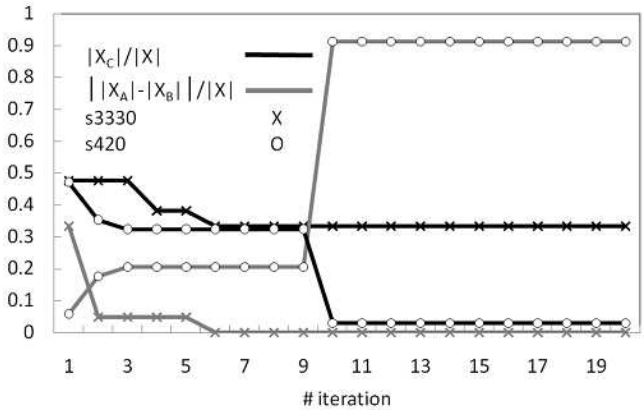

Figure 1: $|X_C|/|X|$ and $||X_A| - |X_B||/|X|$ in the enumeration of variable partitions in OR2-decomposition.

To measure the quality of a variable partition, we use two metrics: $|X_C|/|X|$ for disjointness, and $||X_A| - |X_B||/|X|$ for balancedness. The smaller they are, the better a partition is. In particular, we prefer disjointness to balancedness since the former yields better variable reduction. Experience suggests that $|X_C|/|X|$ very often can be maximally reduced within the first few enumerations while keeping $||X_A| - |X_B||/|X|$ as low as possible. Figure 1 shows how these two values may change in enumerating different variable partitions under OR2-decomposition of two sample functions from circuits `s3330` and `s420`. In the figure, every variable partition corresponds to two markers (one in black and the other in gray) with the same symbol at the same iteration.

It is interesting to note that OR2- and XOR-decompositions exhibit very different characteristics in variable partitioning. Figures 2 and 3 show the difference. In these two plots, a marker corresponds to a first found valid variable partition in decomposing some function. As can be seen, the decomposition quality is generally good in OR2-decomposition, but not in XOR-decomposition. This phenomenon is because XOR-decomposable circuits, e.g. arithmetic circuits, possess some regular structures in their functionality. This regularity makes disjointness and balancedness mutually exclusive in variable partitioning.

6. CONCLUSIONS

We showed that the bi-decomposition of a Boolean function can be achieved through SAT solving. Interpolation (respectively cofactoring) turned out playing an essential role in the computation of OR2- and AND2-decomposition (respectively XOR-decomposition). Moreover variable partitioning was automated as an integrated part of the decomposition process. Thereby the capacity of bi-decomposition can be much extended for large functions. Experiments showed promising results on the scalability of bi-decomposition and the optimality of variable partitioning.

Although our method has its strengths in dealing with large functions and in automating variable partitioning, it is weak in handling don't cares when compared with BDD-based approaches. Future work on hybrid approaches com-

Table 1: Bi-decomposition of PO functions

				OR2-decomposition				XOR-decomposition			
circuit	#in	#max	#out	#dec	#slv	time (sec)	mem (Mb)	#dec	# slv	time (sec)	mem (Mb)
b04	76	38	74	49	3878	12.26	19.35	49	2714	28.82	20.02
b07	49	42	57	14	12985	27.59	22.3	39	601	5.43	18.72
b12	125	37	127	80	12526	25.14	23.32	84	4862	19.22	26.93
C1355	41	41	32	0	26240	354	20.32	–	–	TO	–
C432	36	36	7	7	102	13.15	18.54	0	3654	197.81	17.46
C880	60	45	26	16	222	8.36	20.72	11	4192	83.08	18.72
comp	32	32	3	0	1488	2.61	15.86	1	1014	13.69	16.9
dalu	75	75	16	1	26848	352.87	24.14	16	210	26.59	19.68
e64	65	65	65	0	45760	17.98	22.91	0	45760	388.18	24.37
i2	201	201	1	1	1	1.07	18.6	1	34	2.16	18.59
i4	192	47	6	4	6	0.58	16.08	0	4326	60.04	16.54
k2	45	45	45	33	1071	17.51	22.33	33	612	5.29	20.71
my_adder	33	33	17	0	3656	2.61	18.05	16	577	4.92	17.32
o64	130	130	1	1	1	0.36	16.17	0	8385	623.43	16.12
rot	135	63	107	49	19927	65.97	23.21	46	4975	59.23	21.96
s3330	172	87	205	60	2941	9.42	23.09	71	3135	16.45	21.87
s420	34	34	17	1	817	0.88	17.83	17	114	0.7	16.58
s6669	322	49	294	101	24423	198.14	29.13	176	3120	279.03	22.87
s938	66	66	33	1	5985	2.81	19.86	33	426	4.49	16.28

#in: number of PIs; #max: maximum number of support vars in POs; #out: number of POs;
#dec: number of decomposable POs; #slv: number of SAT solving runs; TO: time out at 1500 sec

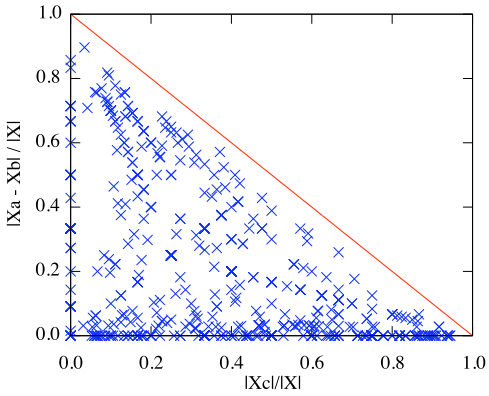

Figure 2: Variable partition in OR2-decomposition.

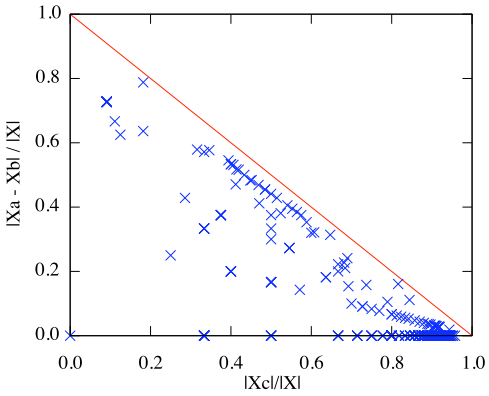

Figure 3: Variable partition in XOR-decomposition.

bining SAT and BDD may exploit more don't cares for better decomposition of large functions. Also, an outstanding open problem remains to be solved is the XOR-decomposition of incompletely specified functions using SAT solving.

7. REFERENCES

[1] R. L. Ashenhurst. The decomposition of switching functions. In *Proc. Int'l Symp. on the Theory of Switching Functions*, pages 74–116, 1959.

[2] Berkeley Logic Synthesis and Verification Group. *ABC: A system for sequential synthesis and verification.* `http://www.eecs.berkeley.edu/~alanmi/abc/`.

[3] D. Bochmann, F. Dresig, and B. Steinbach. A new decomposition method for multilevel circuit design. In *Proc. Euro-DAC*, pages 374–377, 1991.

[4] J. Cortadella. Timing-driven logic bi-decomposition. *IEEE Trans. on CAD*, 22(6):675–685, 2003.

[5] W. Craig. Linear reasoning: A new form of the Herbrand-Gentzen theorem. *J. Symbolic Logic*, 22(3):250–268, 1957.

[6] A. Curtis. *New approach to the design of switching circuits.* Van Nostrand, Princeton, NJ, 1962.

[7] N. Een and N. Soensson. An extensible SAT-solver. In *Proc. SAT*, pages 502–518, 2003.

[8] V. Kravets. Private communication about an IBM internal disclosure. 2008.

[9] C.-C. Lee, J.-H. R. Jiang, C.-Y. Huang, and A. Mishchenko. Scalable exploration of functional dependency by interpolation and incremental SAT solving. In *Proc. ICCAD*, pages 227–233, 2007.

[10] K. L. McMillan. Interpolation and SAT-based model checking. In *Proc. CAV*, pages 1–13, 2003.

[11] A. Mishchenko, B. Steinbach, and M. A. Perkowski. An algorithm for bi-decomposition of logic functions. In *Proc. DAC*, pages 103–108, 2001.

[12] Y. Oh, M. Mneimneh, Z. Andraus, K. Sakallah, and I. Markov. Amuse: A minimally-unsatisfiable subformula extractor. In *Proc. DAC*, pages 518–523, 2004.

[13] J. A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–41, 1965.

[14] T. Sasao and J. Butler. On bi-decomposition of logic functions. In *Proc. IWLS*, 1997.

[15] C. Scholl. *Functional Decomposition with Applications to FPGA Synthesis.* Kluwer Academic Publishers, 2001.

[16] B. Steinbach and A. Wereszczynski. Synthesis of multi-level circuits using EXOR-gates. In *Proc. IFIP Workshop on Applications of the Reed-Muller Expansion in Circuit Design*, pages 161–168, 1995.

[17] G. Tseitin. On the complexity of derivation in propositional calculus. *Studies in Constructive Mathematics and Mathematical Logic*, pages 466–483, 1970.