Improvements to Satisfiability-Based Boolean Function Bi-Decomposition

Huan Chen and Joao Marques-Silva

Complex & Adaptive Systems Laboratory (CASL)
School of Computer Science and Informatics (CSI)
University College Dublin
Dublin, Ireland
Email: {*huan.chen*, *jpms*}@ucd.ie

Abstract. Boolean function bi-decomposition is pervasive in logic synthesis. Bi-decomposition entails the decomposition of a Boolean function into two other functions connected by a simple two-input gate. Existing solutions are based on Binary Decision Diagrams (BDDs) and, more recently, on Boolean Satisfiability (SAT). Recent work exploited the identification of Minimally Unsatisfiable Subformulas (MUSes) for computing the sets of variables to use in Boolean function bi-decomposition. This paper develops new techniques for improving the use of MUSes in function bi-decomposition. The first technique exploits structural properties of the function being decomposed, whereas the second technique exploits group-oriented MUSes. Experimental results obtained on representative benchmarks from logic synthesis demonstrate significant improvements both in performance and in the quality of decompositions.

Keywords: bi-decomposition, logic synthesis, satisfiability, MUS, group-oriented MUS

1 Introduction

Boolean function decomposition [1, 2] is ubiquitous in logic synthesis, being a fundamental technique in multi-level logic synthesis. The goal of functional decomposition is to represent a complex Boolean function $f(X)$ as $f(X) = h(g_1(X), \ldots, g_m(X))$, such that $h, g_1, \ldots, g_m$ are simpler subfunctions. Functional decomposition plays an important role in Electronic Design Automation (EDA) for VLSI, including multi-level logic synthesis and FPGA synthesis [3–5].

Bi-decomposition [6–12] is a special form (with m=2) of Boolean function decomposition, and it is arguably the most widely used form of Boolean function decomposition. It consists of decomposing Boolean function $f(X)$ into the form of $f(X) = h(f_A(X_A, X_C), f_B(X_B, X_C))$, under variable partition $X = \{X_A|X_B|X_C\}$. The quality of bi-decomposition is mainly determined by the quality of variable partitions, as an optimal solution results in simpler subfunctions f_A and f_B. Typically, two *relative* quality metrics [11, 13], namely

disjointness and *balancedness*, are used to evaluate the resulting variable partitions, for which smaller values represent preferred bi-decompositions. In practice, disjointness is in general preferred [11], since it represents the reduction of common variables to f_A and f_B, which in turn often simplifies the resulting Boolean function. Similar to recent work on functional decomposition [11,13], this paper addresses these two relative metrics, namely disjointness and balancedness. *Absolute* quality metrics are an alternative to relative quality metrics, and include total variable count (Σ) and maximum partition size (Δ) [12]. Nevertheless, in practice absolute quality metrics scale worse with the number of inputs [12].

The research on decomposition of Boolean functions can be traced back to the 1950s [1,2]. The very first algorithm for bi-decomposition was presented for the AND case in [14]. The first solution for XOR case was given in [15]. The general case of bi-decomposing of Boolean network was proposed in [16]. Traditional algorithms [3,5,8–10,17] use BDDs as the underlying data structure. However, BDDs impose severe constraints on the number of input variables circuits can have. Hence, it is generally accepted that BDDs do not scale for *large* Boolean functions. As a result, recent work [11,13,18] proposed the use of Boolean Satisfiability (SAT) to manipulate large Boolean functions. For example, [11] has a number of key features, including: (1) good performance on some large circuits; and (2) capability to automatically identify variable partitions. Nevertheless, detailed experimental evaluation of the work in [11] revealed a few shortcomings: (1) The ever-increasing size of circuits to synthesize requires more efficient techniques for Boolean function bi-decomposition; and (2) The underlying SAT solver affects the efficiency of computing of Minimally Unsatisfiable Subformulas (MUSes), which in turn determine the final quality of variable partitions.

The paper has two main contributions. The first one develops heuristics and adapts modern MUS algorithms, which offer significant performance improvements as well as better quality of computed bi-decompositions. The second contribution exploits the idea of constraint grouping [19] used in group-oriented (or high-level) MUSes [19–23]. The use of group-oriented MUS extraction allows performance improvements that can exceed two orders of magnitude in comparison with the results of [11].

The paper is organized as follows. Section 2 provides the preliminaries. Section 3 reviews the models for Boolean function bi-decomposition. Section 4 proposes new Satisfiability-based models. Section 5 illustrates an example of OR bi-decomposition in detail. Section 6 presents the experimental results. Section 7 concludes the paper and outlines a number of future research directions.

2 Preliminaries

2.1 Notation

Variables are represented by set $X = \{x_1, x_2, \ldots, x_n\}$. The cardinality of X is denoted as $||X||$. A partition of a set X into $X_i \subseteq X$ for $i = 1, \ldots, k$ (with $X_i \bigcap X_j = \emptyset, i \neq j$ and $\bigcup_i X_i = X$) is denoted by $\{X_1|X_2|\ldots|X_k\}$. A Completely Specified Function (CSF) is denoted by $f : \mathcal{B}^n \rightarrow \mathcal{B}$. An Incompletely

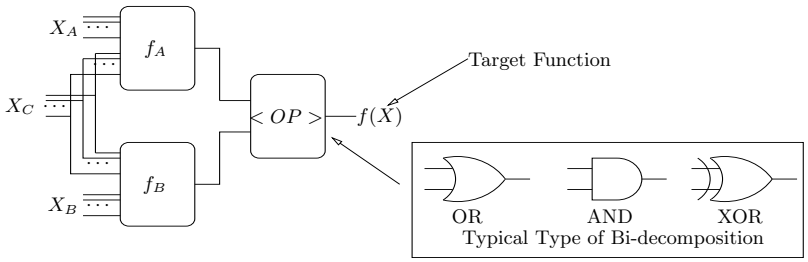

Fig. 1. Bi-decomposition

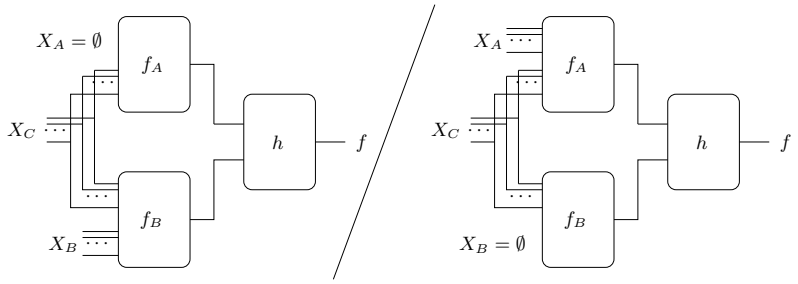

Fig. 2. Trivial variable partition

Specified Function (ISF) $F(X)$ is a 3-tuple (f_q, f_d, f_r), where f_q, f_d and f_r are the onset, don't-care set and offset functions of $F(X)$.

2.2 Boolean Function Bi-Decomposition

Definition 1 (Bi-Decomposition). *[7] Bi-decomposition for a Completely Specified Function (CSF) consists of decomposing a CSF function $f(X)$ under variable partition $X = \{X_A|X_B|X_C\}$, into the form of $f(X) = f_A(X_A, X_C)$ <OP> $f_B(X_B, X_C)$, where <OP> is a binary operator, typically* OR*,* AND *or* XOR*.*

This paper addresses *OR*, *AND* and *XOR* bi-decomposition because these three basic gates form other types of bi-decomposition [11]. Figure 1 illustrates the corresponding concepts. Bi-decomposition is termed *disjoint* if $||X_C|| = 0$. A partition of X is trivial if $X = X_A \bigcup X_C$ or $X = X_B \bigcup X_C$ holds. The concept of trivial partition is illustrated in Figure 2. Similar to earlier work [11, 13], this paper addresses non-trivial bi-decompositions.

Definition 2 (Support Variable). *For a completely specified function $f(X)$ with input variables $X = (x_1, \ldots, x_m)$, variable x_i is a* support variable *of f if*

$$f(x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_m) \neq f(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_m) \qquad (1)$$

2.3 Boolean Satisfiability

Boolean formulas are defined over a finite set X of Boolean variables. Individual variables are represented by letters x, y, z, w and o, and subscripts may be used (e.g. x_1). The Boolean connectives considered will be $\neg$, $\rightarrow$, $\leftrightarrow$, $\wedge$, $\vee$. When necessary, parentheses are used to enforce precedence.

A formula in Conjunctive Normal Form (CNF) $\mathcal{F}$ is defined as a set of sets of literals defined on X, representing a conjunction of disjunctions of literals. A literal is either a variable or its complement. Each set of literals is referred to as a clause c. Moreover, it is assumed that each clause is non-tautological. The Boolean SAT problem is an NP–Complete decision problem [24]. Additional SAT definitions can be found in standard references (e.g. [25]).

Definition 3 (MUS). *[26] $\mathcal{M} \subseteq \mathcal{F}$ is a Minimally Unsatisfiable Subformula (MUS) iff $\mathcal{M}$ is unsatisfiable and $\forall_{c \in \mathcal{M}}$, $\mathcal{M} \setminus \{c\}$ is satisfiable.*

MUSes find a wide range of practical applications, including Boolean function decomposition [11, 13, 18], high-level MUSes [19] for the refinement of datapath abstractions [22] and formal equivalence checking [20, 21]. (See [26] for a recent overview of MUSes.)

Group-oriented MUSes is an alternative name for high-level MUSes [19, 21, 23]. In the *group-oriented* MUS problem, the input is an unsatisfiable set of clauses (a CNF formula) $\mathcal{C} = \mathcal{D} \cup \mathcal{G}_1 \cup \cdots \cup \mathcal{G}_k$ that is *explicitly partitioned* into the groups $\mathcal{D}, \mathcal{G}_1, \ldots, \mathcal{G}_k$ of clauses such that $\mathcal{D} \cap \mathcal{G}_i = \emptyset$ and $\mathcal{G}_i \cap \mathcal{G}_j = \emptyset$ hold for each $i, j \in \{1, \ldots, k\}$ with $i \neq j$.

Definition 4 (Group-oriented MUS). *[23] Given an explicitly partitioned unsatisfiable CNF formula $\mathcal{C} = \mathcal{D} \cup \bigcup_{G \in \mathcal{G}} G$, where $\mathcal{G} = \{\mathcal{G}_1, \ldots, \mathcal{G}_k\}$, and $\mathcal{D}$ and each $\mathcal{G}_i$ are disjoint sets of clauses, a group-oriented MUS of $\mathcal{C}$ is a subset $\mathcal{G}'$ of $\mathcal{G}$ such that $\mathcal{D} \cup \bigcup_{G \in \mathcal{G}'} G$ is unsatisfiable and, for every $\mathcal{G}'' \subset \mathcal{G}'$, we have that $\mathcal{D} \cup \bigcup_{G \in \mathcal{G}''} G$ is satisfiable.*

Notice that $\mathcal{D}$ and the clauses in $\mathcal{D}$ do not contribute to the size of a group-oriented MUS, and can hence be viewed as *don't care* or irrelevant clauses w.r.t. the size of the group-oriented MUSes of $\mathcal{C}$. Many practical applications, e.g. [20,22], require minimizing the number of high-level propositional interesting constraints in the problem formula [21]. The interested constraints are expressed as sets of clauses, where those clauses can be partitioned into groups [23]. For example, clauses which encode one gate of a circuit-level description may form a group. Group-oriented MUS solvers [23], e.g. *MUSer* [27], can identify one group-oriented MUS of $\mathcal{C}$.

2.4 Unsatisfiability proof and Craig Interpolation

This subsection reviews *unsatisfiability proofs* and *Craig Interpolation*, which are used for constructing decomposition functions f_A and f_B in the SAT-based bi-decomposition [11]. Modern SAT solvers learn clauses [28–31]. For unsatisfiable instances, the original and the learned clauses can be used for generating

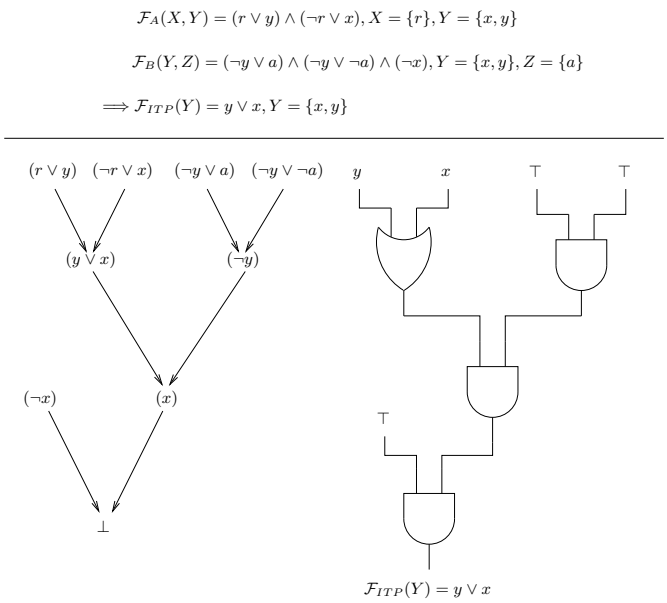

Fig. 3. Resolution graph and interpolation

a resolution-based unsatisfiability proof [32]. Modern SAT solvers can be instructed to generate a *proof trace*, which associates with each learned clause w_L, all the clauses that explain the creation of w_L [32]. Given a proof trace Γ, where the final traced clause is an empty clause $\bot$, it is possible to create a resolution-based unsatisfiability proof in linear time and size w.r.t. proof trace.

Definition 5 (Unsatisfiability Proof e.g. [33]). *A proof of unsatisfiability Π for a set of clauses w is directed acyclic graph (V_Π, E_Π), where V_Π is a set of clauses, such that:*

- *For every $w \in V_\Pi$, either*
  - *$w \in \mathcal{F}$, and w is a root, or*
  - *w has two predecessors, w_1 and w_2, such that w is the resolvent of w_1 and w_2 (the variable p used for resolving w_1 with w_2 is referred to as the* pivot *variable of the resolution step), and*
- *the empty clause $\bot$ is the unique leaf.*

Assume a Boolean formula $\mathcal{F}_A(X,Y)$, defined over the sets of variables X and Y, and a Boolean formula $\mathcal{F}_B(Y,Z)$, defined over the sets of variables Y and Z.

Theorem 1 (Craig Interpolation). *[34] If $\mathcal{F}_A \wedge \mathcal{F}_B$ is unsatisfiable, then there exists a Boolean formula $\mathcal{F}_{ITP}(Y)$, defined over the set of variables Y, such that $\mathcal{F}_A(X,Y) \rightarrow \mathcal{F}_{ITP}(Y)$ is a tautology and $\mathcal{F}_{ITP} \wedge \mathcal{F}_B(Y,W)$ is unsatisfiable. $\mathcal{F}_{ITP}(Y)$ is referred to as an* interpolant *for $\mathcal{F}_A(X,Y)$ and $\mathcal{F}_B(Y,Z)$.*

Interpolant $\mathcal{F}_{ITP}$ can be computed in linear time on the size of a resolution refutation of $\mathcal{F}_A$ and $\mathcal{F}_B$ [33]. Besides, $\mathcal{F}_{ITP}$ has size linear on the size of unsatisfiability proof [33, 35]. In the following, McMillan's interpolant construction [33] is outlined, but Pudlak's construction [35] could also be considered. Assume Boolean formulas $\mathcal{F}_A(X, Y)$ and $\mathcal{F}_B(Y, Z)$. Variables Y is referred to as *global* variables w.r.t. $\mathcal{F}_A(X, Y)$ and $\mathcal{F}_B(Y, Z)$, whereas variables X and Z are local to $\mathcal{F}_A$ and $\mathcal{F}_B$ respectively. Let $g(w)$ denote the literals corresponding to global variables in clause w.

Definition 6 (Interpolatnt). *Let $(\mathcal{F}_A, \mathcal{F}_B)$ be a pair of clause sets and let Π be a proof of unsatisfiability of $\mathcal{F}_A$ and $\mathcal{F}_B$, with leaf vertex $\perp$. For each vertex $w \in V_\Pi$, let $\mathcal{F}_w$ be a Boolean formula, such that:*

- *If w is root then*
  - *if $w \in \mathcal{F}_A$ then $\mathcal{F}_w = g(w)$,*
  - *else $\mathcal{F}_w =$TRUE.*
- *else, let w_1, w_2 be the predecessors of w and let p be their pivot variable*
  - *if p is local to $\mathcal{F}_A$, then $\mathcal{F}_w = \mathcal{F}_{w_1} \vee \mathcal{F}_{w_2}$,*
  - *else $\mathcal{F}_w = \mathcal{F}_{w_1} \wedge \mathcal{F}_{w_2}$.*
- *The Π-interpolation of $(\mathcal{F}_A, \mathcal{F}_B)$, denoted* $\textsc{Itp}(\Pi, \mathcal{F}_A, \mathcal{F}_B)$ *is $\mathcal{F}_\perp$.*

Example 1 (Resolution graph and interpolation). As explained in Figure 3, resolution graph of an UNSAT Boolean formula $\mathcal{F}_A(X, Y) \wedge \mathcal{F}_B(Y, Z)$ leads to an empty clause $\perp$. The interpolation procedure (see above) produces an interpolant $\mathcal{F}_{ITP}(Y)$.

2.5 Quality Metrics

The quality of variable partitions mainly impacts the quality of bi-decomposition [11, 12], and indirectly impacts the decomposed network, e.g. delay, area and power consumption [12]. Similar to [11, 13], this paper measures the quality of variable partitions through two *relative* quality metrics, namely *disjointness* and *balancedness*. Assume a variable partition $\{X_A|X_B|X_C\}$ for $f(X)$, where X_A, X_B and X_C are the sets of the input variables to decomposition functions f_A, f_B and common to f_A and f_B, respectively.

Definition 7 (Disjointness). $\epsilon_D = \frac{||X_C||}{||X||}$ *denotes the ratio of the number of common variables to inputs. A value of ϵ_D close to 0 is preferred, as $\epsilon_D = 0$ represents a disjoint bi-decomposition.*

Definition 8 (Balancedness). $\epsilon_B = \frac{\big|||X_A|| - ||X_B||\big|}{||X||}$ *denotes the absolute size difference between X_A and X_B. $\epsilon_B = 0$ represents a balanced variable partition.*

In practice, disjointness is preferred since a lower value represents a smaller number of shared input variables of the resulting decomposed circuit that typically has smaller area and power footprint. A lower balancedness typically corresponds to smaller delay of the decomposed network.

3 Related Work

Boolean function decomposition approaches are either based on BDDs or SAT. This section briefly overviews earlier work on Boolean function decomposition.

3.1 BDD-Based Bi-Decomposition

Traditional algorithms [3, 5, 8–10, 17] of bi-decomposition are based on BDDs. BDD-based function decomposition approaches implement different forms of bi-decomposition, including OR, AND, XOR, MIN and MAX [7,8], targeting optimization of timing [36] and area of the synthesized circuits [9,10]. Assuming the variable partition $X = \{X_A|X_B|X_C\}$ of $f(X)$ is given, then the bi-decomposition problem can be stated as follows [8]:

Definition 9. *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 BDD quantified formula:*

$$f(X_A, X_B, X_C) \wedge \exists_{X_A}.\neg f(X_A, X_B, X_C) \wedge \exists_{X_B}.\neg f(X_A, X_B, X_C) \qquad (2)$$

is false.

Algorithms based on BDDs have a number of advantages, including flexible Boolean function manipulation [10], the ability to handle don't-care conditions [8] and on-demand selection of best partition of variables [37]. In contrast, the main drawback of BDDs is that they can be used only on functions with a fairly small number of inputs [11].

3.2 SAT-based Bi-Decomposition

Recent work [11] proposed SAT-based solutions. The use of SAT not only makes the computation of bi-decomposition feasible for large circuits, but also serves for automatically selecting and optimizing variable partitions. SAT-based OR, AND and XOR bi-decompositions under known and unknown partition of variables were proposed in [11]. For example, the widely used OR bi-decomposition can be constructed by SAT solving [11]. Given a non-trivial variable partition $X = \{X_A|X_B|X_C\}$, the following result holds:

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 Boolean formula*

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

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

An *instantiated* version x' of Boolean variable x can be viewed as a new Boolean variable x' that replaces x. This approach assumes that a variable partition $X = \{X_A|X_B|X_c\}$ is given. In practice, such variable partitions are generally unknown and must be automatically derived. One approach to consider instead is the following formulation [11]:

$$f(X) \wedge \neg f(X') \wedge \bigwedge_i ((x_i \equiv x'_i) \vee \alpha_{x_i}) \wedge \neg f(X'') \wedge \bigwedge_i ((x_i \equiv x''_i) \vee \beta_{x_i}) \quad (4)$$

where $x' \in X'$ and $x'' \in X''$ are the instantiated version of $x \in X$. α_{x_i} and β_{x_i} are *control* variables for enumerating variable partitions. By assigning different Boolean values to α_{x_i} and β_{x_i}, some of the clauses $((x_i \equiv x'_i) \vee \alpha_{x_i})$, $((x_i \equiv x''_i) \vee \beta_{x_i})$ are relaxed. The resulting clauses $(x_i \equiv x'_i)$ and $(x_i \equiv x''_i)$ impose equivalence relations for each pair of variables in sets X and X', and in X and X'', respectively.

The original work on SAT-based bi-decomposition [11] proposed the use of interpolation for computing the target functions f_A and f_B. Given that our work focuses on improving the identification of MUSes, interpolation can also be used for computing functions f_A and f_B. Similar to OR bi-decomposition, AND and XOR bi-decomposition can be constructed by using Boolean SAT. Due to space limitations, this section omits the explanation of SAT-based AND and XOR bi-decompositions (e.g. see [11]). The approaches proposed [11] are referred to as *LJH* in the remainder of the paper.

4 Improved MUS-Based Bi-Decomposition

Earlier work on SAT-based function bi-decomposition proposed computing MUSes with SAT solvers [11, 13, 18], where partitions are partially enumerated. This section extends this earlier work, and develops two techniques that improve performance significantly and achieve better quality partitions. The first technique exploits structural properties for guiding the computation of MUSes. The second technique exploits recent work on applying group-oriented MUSes in formal verification of large-scale designs [19–22].

4.1 Plain MUS-Based Bi-Decomposition

OR Bi-Decomposition for CSF OR bi-decomposition can be constructed by SAT solving [11]. Given a non-trivial variable partition $X = \{X_A|X_B|X_C\}$, a CSF $f(X)$ 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 *iff* the Boolean formula

$$f(X_A, X_B, X_C) \wedge \neg(X'_A, X_B, X_C) \wedge \neg f(X_A, X'_B, X_C) \quad (5)$$

is *unsatisfiable*. This approach assumes that a variable partition $X = \{X_A|X_B|X_C\}$ is given. In practice, such variable partitions are generally unknown and need to be automatically derived. As a result, the derivation of variable partitions must

be automated. Earlier work [11] proposed the SAT-based model given in (4). This model gives a variable partition if (4) is *unsatisfiable* under a non-trivial partition. α_{x_i} and β_{x_i} are called *control* variables, used for the purpose of relaxing clauses. Assignments $(\alpha_{x_i}, \beta_{x_i}) = (0,0), (0,1), (1,0)$ and $(1,1)$ indicate the partition, to which x_i belongs, $x_i \in X_C$, $x_i \in X_B$, $x_i \in X_A$, and x_i can either be in X_A or X_B, respectively.

Enumerating different values of the control variables will result in different variable partitions. A solution corresponds to an Unsatisfiable Subformula (US) of the original CNF formula. An optimal solution is an MUS. The optimization of variable partitions is the process of enumerating and selecting MUSes. If a disjoint variable partition ($||X_C|| = 0$) is concerned, the solving process corresponds to finding a *minimum* unsatisfiable core [11]. However, it is well-known that computing a minimum-size unsatisfiable core is harder than computing a minimal one. Therefore, a practical solution is to compute an MUS instead.

Equation (4) serves to extract an unsatisfiable subformula that results in a non-trivial partition. This is done by enumerating control variables. However, this enumeration is known *not* to be effective in practice, essentially because the enumeration is exponential in the number of variables. As a result, this SAT-based model is modified such that (1) control variables are removed, (2) structural heuristics are used to guide the search for a partition, and (3) the interface of a modern MUS extractor MUSer [27] is exploited to improve overall performance.

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

$$f(X) \wedge \neg f(X') \wedge \mathcal{F}_{\mathcal{A}} \wedge \neg f(X'') \wedge \mathcal{F}_{\mathcal{B}} \tag{6}$$

is unsatisfiable *under a non-trivial partition, where $\mathcal{F}_{\mathcal{A}} \subset \bigcup_i \{(x_i \equiv x_i')\}$, and $\mathcal{F}_{\mathcal{B}} \subset \bigcup_i \{(x_i \equiv x_i'')\}$, variable set X' and X'' are the instantiated versions of variable set X, $x' \in X'$ and $x'' \in X''$ are the instantiated versions of $x \in X$.*

The identification of a non-trivial variable partition typically starts from identifying a *seed* variable partition [11]. A seed variable partition makes (4) unsatisfiable where partition X_A and X_B each take at least one variable. This scheme also applies to the proposed new model (6). It can be shown that the existence of non-trivial OR bi-decomposition can be checked with at most $C_n^2 = (n-1) + \cdots + 1 = \frac{n(n-1)}{2}$ different seed partitions [11]. Figure 4 shows a AIG (And-Inverter Graph), representing a disjointly decomposable circuit. A normal search may first check $X_A = \{c\}$, $X_B = \{d\}$ and $X_C = \{a, b\}$. Unfortunately, this results in a trivial partition. Afterwards, since the SAT check failed, the algorithm enumerates other combinations of inputs for X_A and X_B until gets a non-trivial partition. Heuristically, the search of seed variables can incorporate circuit structural information. Selecting one *non-common* input variable l ($l \in$ leaf of left-subtree while $l \notin$ leaf of right-subtree, or $l \notin$ leaf of left-subtree while $l \in$ leaf of right-subtree, if possible) from leaves in each subtree of the root node raises the likelihood of getting seed variables in part because the AIGs are

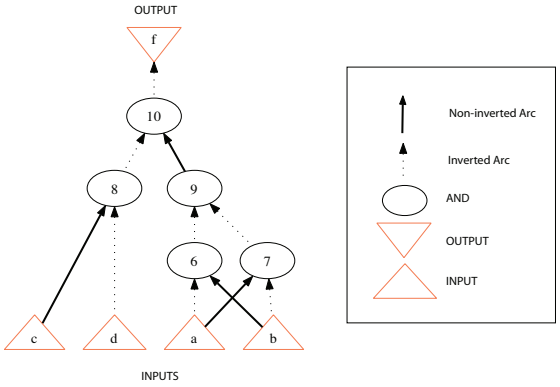

Fig. 4. AIG with disjoint variable partition

structurally hashed. For example in Figure 4, simply select $X_A = \{c\}$ from the left subtree and $X_B = \{a\}$ from the right subtree shapes a seed partition. In practice, this heuristic will help to quickly form the seed variable partitions.

XOR Bi-Decomposition for CSF Similar to the proposed modification to OR bi-decomposition, the XOR bi-decomposition can be constructed by a succinct form of MUS-based model through removing the control variables.

Proposition 3. *A completely specified function $f(X)$ can be decomposed into $f_A(X_A, X_C) \oplus f_B(X_B, X_C)$ for some functions f_A and f_B if and only if the Boolean formula*

$$(f(X) \equiv f(X')) \wedge (f(X'') \not\equiv f(X''')) \wedge \mathcal{F}_{\mathcal{A}} \wedge \mathcal{F}_{\mathcal{B}} \tag{7}$$

is unsatisfiable *under a non-trivial partition, where the sub-formula $\mathcal{F}_{\mathcal{A}} \subset \bigcup_i \{((x_i \equiv x_i'') \wedge (x_i' \equiv x_i'''))\}$, the sub-formula $\mathcal{F}_{\mathcal{B}} \subset \bigcup_i \{((x_i \equiv x_i') \wedge (x_i'' \equiv x_i'''))\}$, variable set X', X'' and X''' are the instantiated versions of variable set X, $x' \in X'$, $x'' \in X''$ and $x''' \in X'''$ are the instantiated versions of $x \in X$.*

4.2 Group-oriented MUS-Based Bi-Decomposition

OR Bi-Decomposition for CSF Essentially, the derivation of variable partitions is the process of switching the input variables between the two partitions. Interestingly, this switching behaviour can be captured by selecting the groups of the input variables. Partition the clauses of formula (6) into $(2i + 1)$ groups:

$$\begin{aligned} \mathcal{D} &= \{f(X) \wedge \neg f(X') \wedge \neg f(X'')\} \\ \mathcal{G}_{i_a} &= \{(x_i \equiv x_i')\} \\ \mathcal{G}_{i_b} &= \{(x_i \equiv x_i'')\} \end{aligned} \tag{8}$$

Proposition 4. *A completely specified function $f(X)$ can be decomposed into $f_A(X_A, X_C) \vee f_B(X_B, X_C)$ for some functions f_A and f_B if and only if the Boolean formula of the set of clauses $\mathcal{C}$, with*

$$\mathcal{C} = \mathcal{D} \cup \mathcal{G}_{\mathcal{A}} \cup \mathcal{G}_{\mathcal{B}} \tag{9}$$

is unsatisfiable *under a non-trivial partition, where the sub-set $\mathcal{G}_{\mathcal{A}} \subset \{\bigcup_i \mathcal{G}_{i_a}\}$, the sub-set $\mathcal{G}_{\mathcal{B}} \subset \{\bigcup_i \mathcal{G}_{i_b}\}$.*

Observe that the resulting subset $\mathcal{C}' = \mathcal{D} \cup \bigcup_i \mathcal{G}'_{i_a} \cup \bigcup_i \mathcal{G}'_{i_b}$ from solving (9) indicates the variable partitions, where $\mathcal{G}'_{i_a}$ and $\mathcal{G}'_{i_b}$ with $((\mathcal{G}'_{i_a} \equiv \mathcal{G}_{i_a}), (\mathcal{G}'_{i_b} \equiv \mathcal{G}_{i_b})) = (1,1)$, $(1,0)$, $(0,1)$, and $(0,0)$ 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. $\mathcal{D}$ consists of $f(X)$, $f(X')$ and $f(X'')$, which is considered as the *don't-care* group. Clauses in this group are irrelevant for MUS extraction; this explains in part the performance improvements observed. As stated earlier, group-oriented MUS extraction must operate on an unsatisfiable formula. Similar to the plain MUS-based approach, a computed seed partition serves as an initial unsatisfied formula of (9).

XOR Bi-Decomposition for CSF The XOR bi-decomposition for CSF can be constructed in a similar way to the group-oriented MUS-based OR bi-decomposition. Partition the clauses of formula (7) into $(2i + 1)$ groups:

$$\begin{aligned} \mathcal{D} &= \{(f(X) \equiv f(X')) \wedge (f(X'') \not\equiv f(X'''))\} \\ \mathcal{G}_{i_a} &= \{((x_i \equiv x''_i) \wedge (x'_i \equiv x'''_i))\} \\ \mathcal{G}_{i_b} &= \{((x_i \equiv x'_i) \wedge (x''_i \equiv x'''_i))\} \end{aligned} \tag{10}$$

Proposition 5. *A completely specified function $f(X)$ can be decomposed into $f_A(X_A, X_C) \oplus f_B(X_B, X_C)$ for some functions f_A and f_B if and only if the Boolean formula of the set of clauses $\mathcal{C}$, with*

$$C = \mathcal{D} \cup \mathcal{G}_{\mathcal{A}} \cup \mathcal{G}_{\mathcal{B}} \tag{11}$$

is unsatisfiable *under a non-trivial partition, where the sub-set $\mathcal{G}_{\mathcal{A}} \subset \{\bigcup_i \mathcal{G}_{i_a}\}$, the sub-set $\mathcal{G}_{\mathcal{B}} \subset \{\bigcup_i \mathcal{G}_{i_b}\}$.*

4.3 AND Bi-Decomposition

AND bi-decomposition is dual to OR bi-decomposition and can be obtained from the construction of OR bi-decomposition [8, 10, 11]. The proposed MUS model (6) is able to decompose $\neg f$ into $f_A \vee f_B$. By negating both sides, f is decomposed into $\neg f_A \wedge \neg f_B$ [11]. Because AIGs (And-Inverter Graphs) are used for manipulating the circuit network, the above conversion could be performed by rewriting only part of the AIG network used in OR bi-decomposition. The following proposition is used to assert the existence and correctness of AND bi-decomposition from the construction of OR bi-decomposition.

Proposition 6. *[11] A function f is AND bi-decomposable if and only if $\neg f$ is OR bi-decomposable.*

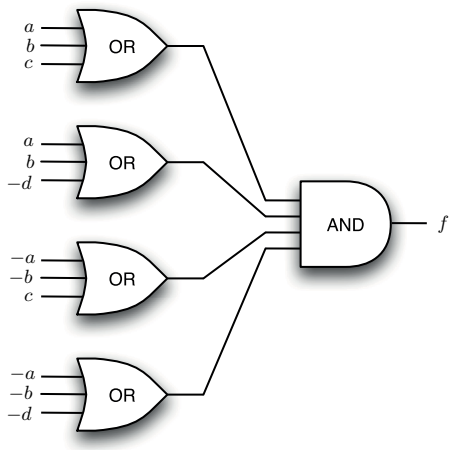

Fig. 5. Circuit network of function $f(a, b, c, d)$

4.4 Incompletely Specified Function (ISF)

This paper assumes Completely Specified Function (CSF). Incompletely Specified Function (ISF) $F(X) = (f_q, f_d, f_r)$ can be decomposed by searching a completely specified function f with $f(X) = f_A(X_A, X_C) \vee f_B(X_B, X_C), f_q(X) \Rightarrow f(X), f(X) \Rightarrow \neg f_r(X)$ if and only if

$$f_q(X_A, X_B, X_C) \wedge f_r(X'_A, X_B, X_C) \wedge f_r(X_A, X'_B, X_C) \tag{12}$$

is unsatisfiable [10, 11].

5 An Example of OR Bi-Decomposition

This section gives an example of plain-MUS-based OR bi-decomposition for a better understanding of the techniques proposed in this paper.

Example 2 (Plain-MUS based OR Bi-decomposition). To OR bi-decompose a Boolean function $f(a, b, c, d)$, shown as Figure 5 in the Product-of-Sums (PoS) form:

$$f(a, b, c, d) = (a \vee b \vee c) \wedge (a \vee b \vee \neg d) \wedge (\neg a \vee \neg b \vee c) \wedge (\neg a \vee \neg b \vee \neg d) \tag{13}$$

where the inputs are $X = \{a, b, c, d\}$. Assume the variable partition is unknown.

Bi-decomposing of a Boolean function starts from encoding the constraints into CNF. Threefold Boolean functions are required to be encoded:

– $f(X)$: the original Boolean function;

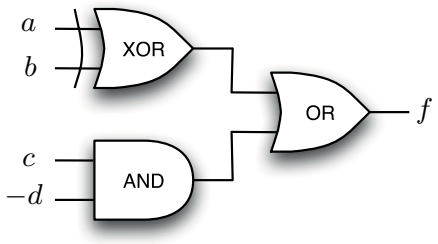

Fig. 6. Bi-decomposed network of function $f(a, b, c, d)$

- $f(X')$: the instantiated version of $f(X)$, where the variables in $f(X)$ are all replaced by fresh new variables;
- $f(X'')$: the instantiated version of $f(X)$, where the variables in $f(X)$ are all replaced by fresh new variables.

In addition, clauses for encoding the equivalent relationship between the input variables X are required:

$$\begin{aligned}(a \vee \neg a') \wedge (\neg a \vee a') \wedge (a \vee \neg a'') \wedge (\neg a \vee a'') \\ \wedge (b \vee \neg b') \wedge (\neg b \vee b') \wedge (b \vee \neg b'') \wedge (\neg b \vee b'') \\ \wedge (c \vee \neg c') \wedge (\neg c \vee c') \wedge (c \vee \neg c'') \wedge (\neg c \vee c'') \\ \wedge (d \vee \neg d') \wedge (\neg d \vee d') \wedge (d \vee \neg d'') \wedge (\neg d \vee d'')\end{aligned} \tag{14}$$

After the encoding of constraints, the next step is to find a seed variable partition with the proposed techniques shown as Figure 4. As a result, an UNSAT formula with non-trivial variable partition is formed:

$$\begin{aligned}f(X) \wedge \neg f(X') \wedge \neg f(X'') \\ \wedge (a \vee \neg a'') \wedge (\neg a \vee a'') \\ \wedge (b \vee \neg b') \wedge (\neg b \vee b') \wedge (b \vee \neg b'') \wedge (\neg b \vee b'') \\ \wedge (c \vee \neg c') \wedge (\neg c \vee c') \\ \wedge (d \vee \neg d') \wedge (\neg d \vee d') \wedge (d \vee \neg d'') \wedge (\neg d \vee d'')\end{aligned} \tag{15}$$

where $a \in X_A, c \in X_B, b \in X_C, d \in X_C$.

The MUS search of (15) helps to refine the variable partitions. In this example, an MUS:

$$\begin{aligned}f(X) \wedge \neg f(X') \wedge \neg f(X'') \\ \wedge (a \vee \neg a'') \wedge (\neg a \vee a'') \wedge (b \vee \neg b'') \wedge (\neg b \vee b'') \\ \wedge (c \vee \neg c') \wedge (\neg c \vee c') \wedge (d \vee \neg d') \wedge (\neg d \vee d')\end{aligned} \tag{16}$$

where $a \in X_A, b, \in X_A, c \in X_B, d \in X_B$ reflects an ideal *disjoint* and *balanced* variable partition, where $disjointness = 0$ and $balancedness = 0$. The search

Table 1. OR Bi-decomposition of Primary Output Functions

Circuit	Circuit Statistics			LJH [11]		Plain-MUS		Group-MUS		CPU Time Ratio		
	#In	#In_Max	#Out	#Dec	Time (s)	#Dec	Time (s)	#Dec	Time (s)	$\frac{LJH}{Plain-MUS}$	$\frac{LJH}{Group-MUS}$	$\frac{Plain-MUS}{Group-MUS}$
s13207	700	212	790	265	171.27	264	57.38	265	3.21	2.98	53.36	17.88
i2	201	201	1	1	0.85	1	0.71	1	0.20	1.20	4.25	3.55
c7552	207	194	108	-	TO	16	97.60	17	14.03	6.15	42.77	6.96
s15850	611	183	684	-	TO	287	429.31	294	21.53	1.40	27.87	19.94
s38584	1464	147	1730	-	TO	1057	62.91	1057	16.67	9.54	35.99	3.77
o64	130	130	1	1	0.23	1	0.19	1	0.11	1.21	2.09	1.73
c2670	233	119	140	40	25.57	40	20.29	40	2.25	1.26	11.36	9.02
i10	257	108	224	-	TO	149	183.91	150	16.83	3.26	35.65	10.93
s3330	173	87	205	59	8.50	74	2.28	74	0.80	3.73	10.63	2.85
s9234	247	83	250	102	159.50	111	20.32	107	11.36	7.85	14.04	1.79
dalu	75	75	16	-	TO	15	20.57	16	3.25	29.17	184.62	6.33
c5315	178	67	123	-	TO	79	41.71	80	12.94	14.39	46.37	3.22
s838	66	66	33	1	4.37	1	2.59	1	2.47	1.69	1.77	1.05
s938	66	66	33	1	2.47	1	2.28	1	1.79	1.08	1.38	1.27
rot	135	63	107	49	47.22	62	2.87	61	1.29	16.45	36.60	2.22
s5378	214	61	228	108	39.54	112	5.67	112	1.46	6.97	27.08	3.88
s1423	91	59	79	26	72.65	41	7.85	34	1.47	9.25	49.42	5.34
pair	173	53	137	117	28.49	114	8.69	114	6.11	3.28	4.66	1.42
c3540	50	50	22	-	TO	10	142.33	13	25.75	4.22	23.30	5.53

of variable partitions requires most of the CPU time in bi-decomposition [12]. In contrast, the computation of decomposition functions f_A and f_B is not time-consuming in SAT-based bi-decompositions.

The Craig Interpolation serves to derive f_A and f_B in OR bi-decomposition [11]. Formula (5) is suggested to replace the use of formula (6).

$$f(a,b,c,d) \wedge \neg f_B(c,d) \wedge \neg f_A(a,b) \tag{17}$$

The remaining work of deriving f_A and f_B follows the procedure proposed [11]. Finally, the interpolation will find the decomposition functions:

$$\begin{aligned} f_A(a,b) &= a \oplus b \\ f_B(c,d) &= c \wedge \neg d \end{aligned} \tag{18}$$

The corresponding bi-decomposed circuit network is shown in Figure 6.

6 Experimental Results

The new techniques described in the previous sections have been implemented in the tool ***STEP*** — ***S****atisfiability-based func****T****ion d****E****com****P****osition* for Boolean

Table 2. AND and XOR Bi-decomposition of Primary Output Functions

Circuit	Circuit Statistics			AND Bi-decomposition				XOR Bi-decomposition				CPU Time Ratio	
				Plain-MUS		Group-MUS		Plain-MUS		Group-MUS		AND	XOR
	#In	#In_Max	#Out	#Dec	Time (s)	#Dec	Time (s)	#Dec	Time (s)	#Dec	Time (s)	$\frac{Plain-MUS}{Group-MUS}$	$\frac{Plain-MUS}{Group-MUS}$
s13207	700	212	790	299	43.57	301	3.58	260	63.52	262	6.69	12.17	9.49
i2	201	201	1	1	1.17	1	0.31	1	2.35	1	0.88	3.77	2.67
c7552	207	194	108	10	104.21	11	26.15	8	457.04	10	52.97	3.99	8.63
s15850	611	183	684	358	417.65	351	19.39	-	TO	237	63.35	21.54	9.47
s38584	1464	147	1730	1099	68.27	1103	22.83	963	173.05	965	34.55	2.99	5.01
o64	130	130	1	1	0.80	1	0.38	0	38.33	0	31.95	2.11	1.20
c2670	233	119	140	37	16.63	37	2.74	33	52.20	35	8.61	6.07	6.06
i10	257	108	224	162	223.99	171	19.77	-	TO	144	56.50	11.33	10.62
s3330	173	87	205	83	2.28	85	0.35	51	4.74	55	1.91	6.51	2.48
s9234	247	83	250	132	21.68	131	10.85	106	45.57	104	9.44	2.00	4.83
dalu	75	75	16	16	19.41	16	2.21	15	40.50	15	3.29	8.78	12.31
c5315	178	67	123	78	16.62	78	3.28	82	59.80	82	10.37	5.07	5.77
s838	66	66	33	1	2.90	1	2.39	32	2.66	32	0.73	1.21	3.64
s938	66	66	33	1	2.28	1	1.76	32	1.86	32	0.63	1.30	2.95
rot	135	63	107	71	2.59	69	0.69	22	14.62	22	2.61	3.75	5.60
s5378	214	61	228	124	6.02	124	1.33	98	27.26	98	9.94	4.53	2.74
s1423	91	59	79	53	10.13	47	1.00	64	15.50	64	2.69	10.13	5.76
pair	173	53	137	121	9.87	121	4.94	98	24.49	98	8.75	2.00	2.80
c3540	50	50	22	12	127.07	14	41.96	6	134.61	9	44.18	3.03	3.05

function bi-decomposition. ***STEP*** is implemented in C++, compiled with G++ 4.4.3, and uses ***ABC*** [38] for circuit manipulation. In addition, ***STEP*** uses ***MUSer*** [27] as the underlying MUS extractor. The tool ***Bi-dec*** implements OR bi-decomposition of LJH model [1] [11].

The experiments compare the performance and quality of Boolean function bi-decompositions between ***Bi-dec*** (with its fastest mode, using command *'bi_dec [circuit.blif] or 0 0'*) and ***STEP***. All results were obtained on the industrial benchmark circuits ISCAS85, ISCAS89, ITC99 and LGSynth. Circuits with *zero* decomposable Primary Output (PO) functions were removed from the tables of results. Due to space restrictions, only representative experimental results (with **#In_Max** ≥ 50) are shown.

The experiments were performed on a Linux machine with an Intel CPU Xeon X3470 2.93 GHz and 6-GB RAM. The *original* circuits were used. Sequential circuits were converted into combinational circuits using ***ABC*** [38]. Similar to [11], for comparison purposes, only experimental results of completely specified functions are shown. For each circuit, the timeout was set to 600 seconds. Each run of the MUS extraction was given a timeout of 10 seconds, that suffices even for the larger circuits.

[1] AND and XOR bi-decompositions using LJH model is unavailable in the tool ***Bi-dec***.

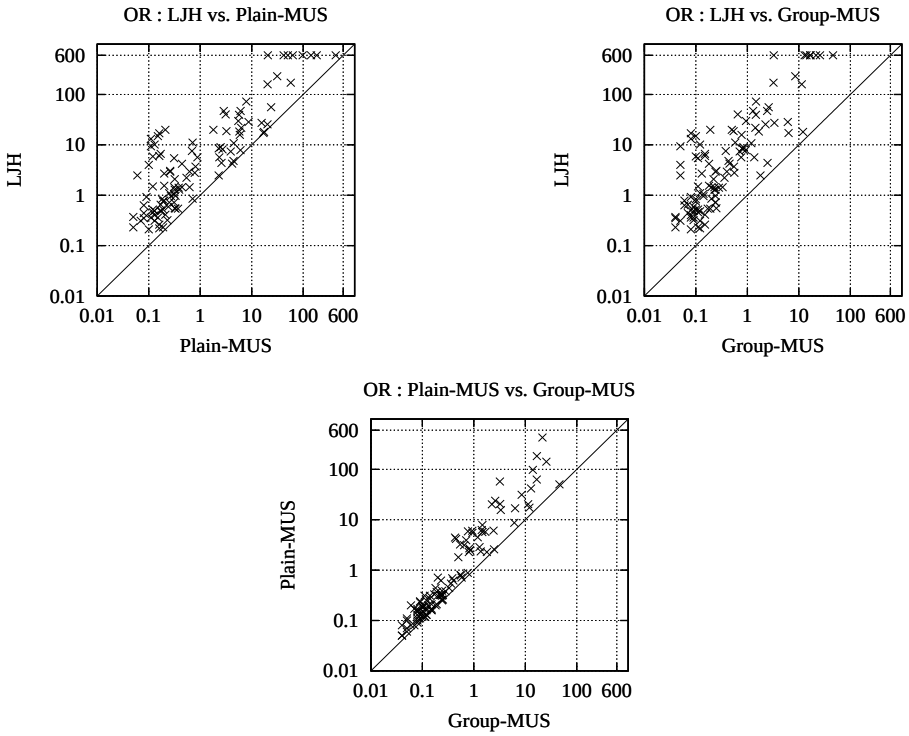

Fig. 7. CPU time comparison between models for OR bi-decomposition

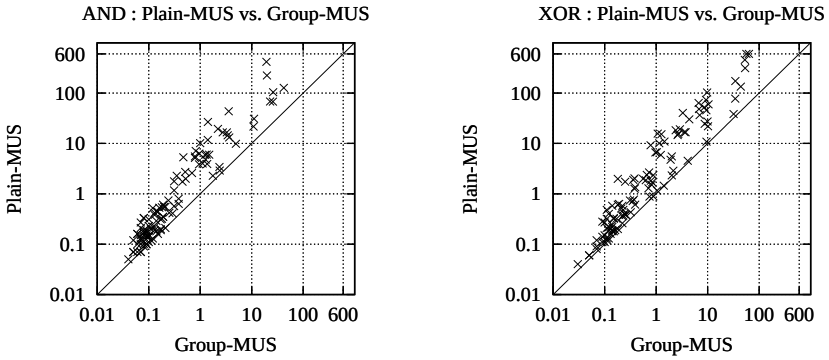

Fig. 8. CPU time comparison between models for AND/XOR bi-decomposition

6.1 Performance of New Techniques

Overall efficiency is crucial in function decomposition as logic synthesis of a circuit involves several iterations of function decomposition. This section evaluates

the performance improvements of the techniques proposed in this paper. Two metrics, CPU time and the number of decomposable functions, were used for assessing performance. Smaller CPU times indicate that decomposing a complete circuit will be faster. A larger number of decomposable functions represents an enhanced decomposability of the tool, indicating the tool is able to decompose more functions in the allowed CPU time, assuming more decomposable functions do exist. Due to space restrictions, only results for circuits with *large* number of support sizes (≥ 50) are presented. Table 1 shows the CPU times and the number of decomposable functions for OR bi-decompositions. Columns **#In**, **#In_Max**, **#Out**, **#Dec** and **Time (s)** denote the number of primary inputs, maximum number of support variables in POs, PO functions (to be decomposed) and decomposable POs and CPU time in seconds, respectively. The experimental data is sorted by decreasing number of maximum support variables (**#In_Max**), to highlight the ability of ***STEP*** at coping with *large* Boolean functions. The results clearly demonstrate that the techniques proposed in this paper significantly outperform the original LJH approach [11], achieving similar decomposability. More importantly, the use of group-oriented MUS extraction yields between one and two orders of magnitude speedup for most benchmarks.

Table 2 shows the CPU times and the number of decomposable functions for AND and XOR bi-decompositions for the approaches based on plain and group-oriented MUS extraction. Figure 7 and 8 show the scatter plots comparing the CPU times (in seconds) for each pair of tools for OR, AND and XOR bi-decomposition on the ISCAS85, ISCAS89, ITC99 and LGSynth benchmark circuits. Each point in each plot represents the CPU time for decomposing a circuit. A more detailed analysis of Figure 7 indicates that the number of aborted circuits for LJH, plain-MUS and group-MUS models are, respectively, 8, 0 and 0, out of 109 circuits. As can be concluded, both the improved plain and the group-oriented MUS approaches achieve significant performance improvements over the LJH approach, often between one and two orders of magnitude. Moreover, between the two approaches proposed in this paper, the group-oriented MUS approach clearly outperforms the improved plain MUS approach.

6.2 Quality of Variable Partitions

The quality of variable partitions mainly determines the quality of bi-decomposition [11]. Similar to [11, 13], the quality of a variable partition is measured by two metrics: *disjointness* and *balancedness*.

Following [11], disjointness is the preferred metric for measuring the quality of decomposition since a better disjointness corresponds to a smaller number of shared input variables of the resulting decomposed circuit hence potentially yields an optimally decomposed circuit during logic synthesis [11]. Similar to [11], ***STEP*** was configured to prefer disjointness over balancedness.

Figure 9 presents the results of quality metrics for LJH OR, Plain-MUS OR/AND/XOR and Group-MUS OR/AND/XOR models. For XOR bi-decompositions, it has been empirically shown that the LJH approach is unable to achieve good quality decompositions in circuits with regular structures [11].

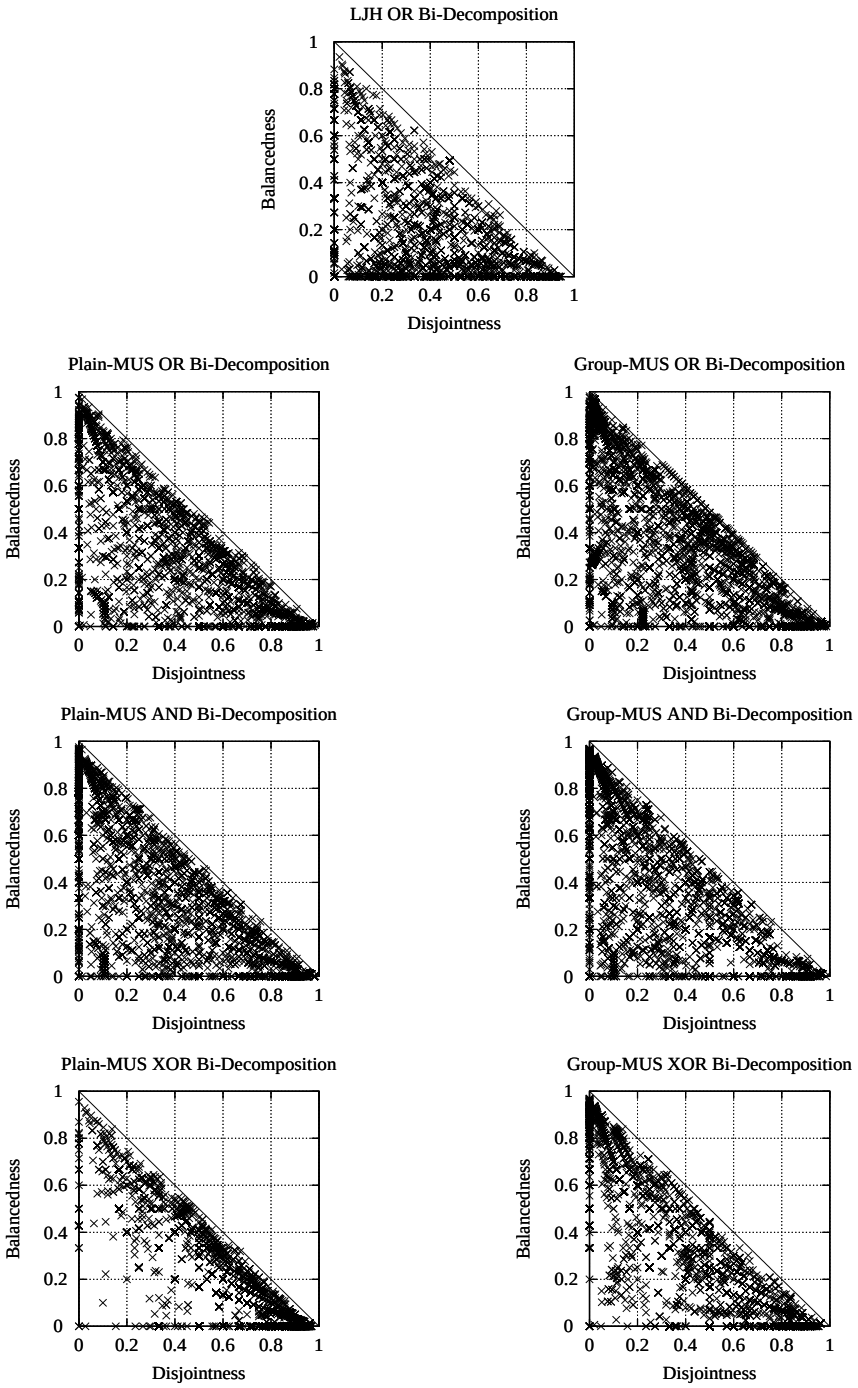

Fig. 9. Quality Metrics for Models

Table 3. Quality metrics comparison between models

Metric	OR LJH vs. PMUS			OR LJH vs. GMUS			OR PMUS vs. GMUS			AND PMUS vs. GMUS			XOR PMUS vs. GMUS		
Disjointness	LJH better	Equal	PMUS better	LJH better	Equal	GMUS better	PMUS better	Equal	GMUS better	PMUS better	Equal	GMUS better	PMUS better	Equal	GMUS better
	24.77%	6.40%	68.83%	36.41%	6.40%	57.19%	28.88%	7.31%	63.81%	2.56%	66.70%	30.74%	0.70%	46.92%	52.38%
Balancedness	LJH better	Equal	PMUS better	LJH better	Equal	GMUS better	PMUS better	Equal	GMUS better	PMUS better	Equal	GMUS better	PMUS better	Equal	GMUS better
	35.08%	48.57%	16.35%	47.71%	45.66%	6.63%	28.32%	63.51%	8.17%	26.02%	61.22%	12.76%	43.95%	47.54%	8.51%

In contrast, the approaches proposed in this paper achieve better disjointness than [11].

Table 3 compares the quality metrics for different approaches, where only the functions that can be decomposed by the both two competitors are computed. The inferior balancedness of new models compared to LJH model results in part from the mutual exclusion nature between low disjointness and low balancedness for some circuits. As can be observed, the techniques proposed in this paper achieve significantly better disjointness than the LJH approach.

7 Conclusion

This paper develops new algorithms for Boolean function bi-decomposition with SAT algorithms. The relative inefficiency of the existing SAT-based models [11] prevent their use on *very large* industrial circuits. This paper proposes new solutions based on group-oriented MUSes, which have found recent application in hardware design and verification [19–22]. The first improvement builds on the existing SAT-based approach [11], by adding heuristics for improving the quality of partitions and by using more effective MUS extraction algorithms [27]. The second improvement consists in formalizing the function bi-decomposition problem in terms of group-oriented MUS extraction [23]. Experimental results obtained on representative circuits, demonstrate that the new MUS-based techniques provide significant performance improvements when compared to the earlier work [11], often by more than one order of magnitude. Moreover, the new approaches yield improved quality of results.

Future work will address the integration of ***STEP*** in a logic design flow [39], targeting area, delay and power reduction. Other research directions involve: (i) exploiting ***STEP*** for optimizing circuit networks in LUT (Looked-Up Table) mapping [38]; (ii) extending the current models to other forms of decomposition, e.g. Ashenhurst Decomposition [1, 13]; (iii) extending the current models for decomposing properties in functional test generation [40]; (iv) exploiting SAT-based ATPG (Automatic Test Pattern Generation) [41] and Minimally Unsatisfiable Circuits [42] for identifying and removing redundancy in decomposition; and (v) exploring the optimum variable partition [12, 39, 43, 44] of function decomposition.

Acknowledgment

The authors would like to thank Prof. Jie-Hong Roland Jiang for his helpful comments and for kindly providing the SAT-based Boolean Function bi-decomposition tool ***Bi-dec***. The authors would like to thank the anonymous referees for helpful comments. This work is partially supported by SFI PI grant BEACON (09/IN.1/I2618).

References

1. Ashenhurst, R.: The decomposition of switching functions. In: Proceedings of an International Symposium on the Theory of Switching. (1957) 74–116
2. Curtis, H.: A new approach to the design of switching circuits. Van Nostrand Princeton, NJ (1962)
3. Lai, Y., Pedram, M., Vrudhula, S.: BDD based decomposition of logic functions with application to FPGA synthesis. In: Design Automation Conference. (1993) 642–647
4. Luba, T., Selvaraj, H.: A general approach to boolean function decomposition and its application in FPGA-based synthesis. VLSI DESIGN **3**(3-4) (1995) 289–300
5. Scholl, C.: Functional decomposition with application to FPGA synthesis. Springer Netherlands (2001)
6. Bochmann, D., Dresig, F., Steinbach, B.: A new decomposition method for multilevel circuit design. In: Proceedings of the European Conference on Design Automation. (1991) 374–377
7. Sasao, T., Butler, J.T.: on bi-decomposition of logic functions. In: International Workshop on Logic and Synthesis. (1997) 1–6
8. Mishchenko, A., Steinbach, B., Perkowski, M.: An algorithm for bi-decomposition of logic functions. In: Design Automation Conference. (2001) 103–108
9. Cortadella, J.: Timing-driven logic bi-decomposition. IEEE Transactions on Computer-Aided Design **22**(6) (Jun. 2003) 675–685
10. Steinbach, B., Lang, C.: Exploiting functional properties of boolean functions for optimal multi-level design by bi-decomposition. Artificial Intelligence Review **20**(3) (2003) 319–360
11. Lee, R.R., Jiang, J.H., Hung, W.L.: Bi-decomposing large boolean functions via interpolation and satisfiability solving. In: Design Automation Conference. (2008) 636–641
12. Choudhury, M., Mohanram, K.: Bi-decomposition of large boolean functions using blocking edge graphs. In: International Conference on Computer-Aided Design. (2010) 586–591
13. Lin, H.P., Jiang, J.H., Lee, R.R.: To sat or not to sat: Ashenhurst decomposition in a large scale. In: International Conference on Computer-Aided Design. (2008) 32–37
14. Malik, A., Harrison, D., Brayton, R.: Three-level decomposition with application to PLDs. In: IEEE International Conference on Computer Design: VLSI in Computers and Processors. (1991) 628–633
15. Sasao, T.: A design method for AND-OR-EXOR three-level networks. In: International Workshop on Logic and Synthesis. (1995) 8:11–8:20

16. Stanion, T., Sechen, C.: Quasi-algebraic decomposition of switching functions. In: Proceedings of the 16th Conference on Advanced Research in VLSI (ARVLSI). (1995) 358–367
17. Chang, S., Marek-Sadowska, M., Hwang, T.: Technology mapping for TLU FPGA's based on decomposition of binary decision diagrams. IEEE Transactions on Computer-Aided Design **15**(10) (1996) 1226–1236
18. Jiang, J.H., Lee, C.C., Mishchenko, A., Huang, C.Y.: To sat or not to sat: Scalable exploration of functional dependency. IEEE Transactions on Computers **59**(4) (Apr. 2010) 457–467
19. Liffiton, M., Sakallah, K.: Algorithms for computing minimal unsatisfiable subsets of constraints. Journal of Automated Reasoning **40**(1) (2008) 1–33
20. Khasidashvili, Z., Kaiss, D., Bustan, D.: A compositional theory for post-reboot observational equivalence checking of hardware. In: Formal Methods in Computer-Aided Design. (2009) 136–143
21. Nadel, A.: Boosting minimal unsatisfiable core extraction. In: Formal Methods in Computer-Aided Design. (2010)
22. Andraus, Z.S., Liffiton, M.H., Sakallah, K.A.: Refinement strategies for verification methods based on datapath abstraction. In: Asia and South Pacific Design Automation Conference. (2006) 19–24
23. International SAT Competitions: http://www.satcompetition.org/. (2002-2011)
24. Cook, S.A.: The complexity of theorem-proving procedures. STOC '71: Proceedings of the third annual ACM symposium on Theory of computing (1971) 151–158
25. Buning, H.K., Lettman, T.: Propositional Logic: Deduction and Algorithms. Cambridge University Press (1999)
26. Marques-Silva, J.: Minimal unsatisfiability: Models, algorithms and applications. In: International Symposium on Multi-Valued Logic. (May 2010) 9–14
27. Marques-Silva, J., Lynce, I.: On improving MUS extraction algorithms. In: International Conference on Theory and Applications of Satisfiability Testing. (2011) 159–173
28. Marques-Silva, J., Sakallah, K.: GRASP-a new search algorithm for satisfiability. In: International Conference on Computer-Aided Design. (Nov. 1996) 220–227
29. Moskewicz, M., Madigan, C., Zhao, Y., Zhang, L., Malik, S.: Chaff: engineering an efficient SAT solver. In: Design Automation Conference. (2001) 530–535
30. Eén, N., Sörensson, N.: An extensible SAT-solver. In: International Conference on Theory and Applications of Satisfiability Testing. (2003) 502–518
31. Biere, A.: PicoSAT essentials. Journal on Satisfiability, Boolean Modeling and Computation (4) (2008) 75–97
32. Zhang, L., Malik, S.: Validating sat solvers using an independent resolution-based checker: Practical implementations and other applications. In: Design, Automation and Test in Europe Conference. (Mar. 2003) 10880–10885
33. McMillan, K.L.: Interpolation and sat-based model checking. In: International Conference on Computer Aided Verification. (2003)
34. Craig, W.: Linear reasoning. a new form of the herbrand-gentzen theorem. Journal of Symbolic Logic **22**(3) (1957) 250–268
35. Pudlak, P.: Lower bounds for resolution and cutting plane proofs and monotone computations. The Journal of Symbolic Logic **62**(3) (1997) 981–998
36. Cheng, L., Chen, D., Wong, M.: DDBDD: Delay-Driven BDD Synthesis for FPGAs. IEEE Transactions on Computer-Aided Design **27**(7) (2008) 1203–1213
37. Kravets, V., Mishchenko, A.: Sequential logic synthesis using symbolic bi-decomposition. In: Design, Automation and Test in Europe Conference. (2009) 1458–1463

38. Berkeley Logic Synthesis and Verification Group. ABC: A System for Sequential Synthesis and Verification, Release 70930: http://www.eecs.berkeley.edu/∼alanmi/abc/
39. Chen, H., Marques-Silva, J.: New and Improved Models for SAT-Based Bi-Decomposition. In: Great Lakes Symposium on VLSI. (2012)
40. Chen, M., Mishra, P.: Decision ordering based property decomposition for functional test generation. In: Design Automation and Test in Europe. (2011) 167–172
41. Chen, H., Marques-Silva, J.: TG-PRO: A SAT-based ATPG System, System Description. In: Journal on Satisfiability, Boolean Modeling and Computation. Volume 8. (Jan. 2012) 83–88
42. Belov, A., Marques-Silva, J.: Minimally Unsatisfiable Boolean Circuits. In: International Conference on Theory and Applications of Satisfiability Testing. (2011) 145–158
43. Chen, H., Janota, M., Marques-Silva, J.: QBF-Based Boolean Function Bi-Decomposition, Computing Research Repository (CoRR), abs/1112.2313 (December 2011)
44. Chen, H., Janota, M., Marques-Silva, J.: QBF-Based Boolean Function Bi-Decomposition. In: Design Automation and Test in Europe. (2012)