Boolean Matching of Function Vectors with Strengthened Learning

Chih-Fan Lai
GIEE
National Taiwan University
Taipei 10617, Taiwan

Jie-Hong R. Jiang
EE Dept./GIEE
National Taiwan University
Taipei 10617, Taiwan

Kuo-Hua Wang
CSIE Dept.
Fu Jen Catholic University
Hsinchuang 24205, Taiwan

jhjiang@cc.ee.ntu.edu.tw

ABSTRACT

Boolean matching for multiple-output functions determines whether two given (in)completely-specified function vectors can be identical to each other under permutation and/or negation of their inputs and outputs. Despite its importance in design rectification, technology mapping, and other logic synthesis applications, there is no much direct study on this subject due to its generality and consequent computational complexity. This paper extends our prior Boolean matching decision procedure BooM to consider multiple-output functions. Through conflict-driven learning and partial assignment reduction, Boolean matching in the most general setting can still be accomplishable even when all other techniques lose their foundation and become unapplicable. Experiments demonstrate the indispensable power of strengthened learning for practical applications.

1. INTRODUCTION

Given two (in)completely-specified Boolean function vectors, Boolean matching (under NPNP-equivalence) determines if they can be equivalent to each other under certain permutation and/or negation of their inputs and outputs. It is an important subject both in theory, see, e.g., [7, 4], and in practice, e.g., [5, 8, 17, 14]. However, most prior efforts on Boolean matching focused mainly on single-output functions, e.g., [5, 23, 22, 1, 3, 2, 24], and very few on multiple-output functions, e.g., [13]. This bias can be attributed to the fact that single-output Boolean matching is more fundamental in theory, easier in computation, and more pervasive in applications. Nevertheless, there are reasons that multiple-output Boolean matching can be equally important. Firstly, the problem itself subsumes the single-output case and is more general. Secondly, there are more unique (output) signatures to be exploited for computational scalability, despite the fact that the theoretical complexity is higher. Thirdly, it has niche applications, e.g., in design rectification [14].

Matching two n-input and m-output functions under NPNP-equivalence can be complex. A naïve exhaustive computation may require functional equivalence checking of $O(2^{m+n}m!n!)$ configurations. A better implementation may reduce the complexity to $O(m^2 2^n n!)$ by checking, over the $2^n n!$ input configurations, whether each output of one function can be equivalent or complementary to some output of the other function. To be shown, with our proposed learning scheme the time complexity can be further reduced to $O(2^{2n})$ as a result of effective search space reduction.

In our earlier work [15], a decision procedure BooM, similar to satisfiability (SAT) and quantified Boolean formula (QBF) solving [10, 11, 9], was proposed for Boolean matching of single-output functions under NPN-equivalence. Based on the previous result, we go one step further extending conflict-driven learning for multiple-output Boolean matching under NPNP-equivalence. In addition the learning is strengthened by exploiting partial truth assignments essential to satisfiability checking. Practical experience suggests that such strengthening is often capable of reducing learned information by 9 times and making deduction much more powerful. On the other hand, learning for other specialized equivalences, namely, NPP-, PNP-, and PP-equivalences, follows straightforwardly from that for the general NPNP-equivalence. Essentially the special problem structure of Boolean matching allows effective learning for search space reduction, which is beyond the capability of a generic QBF solver.

The main features of our method include: 1) It handles NPNP-equivalence, and both completely and incompletely specified function vectors. 2) It is equipped with a strengthened learning technique for effective search space reduction. 3) It supports the search of one matching solution and of all solutions. 4) It admits easy integration with signature-based techniques for search space reduction, and helps signature-based Boolean matching be complete. 5) It uses memory efficient data structures, specifically, and-inverter graphs (AIGs) [16] and conjunctive normal form (CNF) formulas, for scalable Boolean function representation. Experimental results show the effectiveness of our method in conquering instances hard to solve without strengthened learning.

Among prior efforts, the work closest to ours is [13], where support signature (characterizing I/O dependencies), simulation, and SAT solving techniques were integrated for multiple-output Boolean matching under PP-equivalence. Even though support information is a power invariant legitimate under NPNP-equivalence, the applied simulation and SAT solving are applicable to PP-equivalence only. In contrast, we focus on the most general NPNP-equivalence. When PP-equivalence is considered, our method complements the prior method. Our current implementation uses support, unateness, and symmetry signatures for preprocessing. However they can be tightly integrated into the main computation kernel.

This paper is organized as follows. Section 2 gives the preliminaries. Boolean matching under NPNP-equivalence is presented in Section 3, and other specialized equivalences in Section 4. Implementation issues are discussed in Section 5. The proposed methods are evaluated with experimental results in Section 6. Finally Section 7 concludes this paper and outlines future work.

2. PRELIMINARIES

As conventional notation, a set of Boolean variables is denoted with an upper-case letter, e.g., X; its elements are in lower-case letters, e.g., $x_i \in X$. The ordered version (namely, vector) of a set $X = \{x_1, \ldots, x_n\}$ is denoted as $\vec{x} =$

$(x_1, \ldots, x_n)$. The cardinality of a set X (respectively $\vec{x}$) is denoted as $|X|$ (respectively $|\vec{x}|$). The set of truth valuations of $\vec{x}$ is denoted $[\![\vec{x}]\!]$, e.g., $[\![(x_1, x_2)]\!] = \{(0,0), (0,1), (1,0), (1,1)\}$. Let $\perp$ denote an unknown value. The set of partial valuations of $\vec{x}$ is denoted $[\![\vec{x}]\!]^{\perp}$, e.g., $[\![(x_1, x_2)]\!]^{\perp} = \{(\perp,\perp), (\perp,0), (\perp,1), (0,\perp), (0,0), (0,1), (1,\perp), (1,0), (1,1)\}$. The **weight** of a vector $\vec{u}$ of truth values, denoted $wt(\vec{u})$, is the number of 1's in the vector, e.g., $wt((0,1,1)) = 2$.

2.1 Boolean Matching

A **permutation** π over X is a bijection function $\pi : X \rightarrow X$; a **negation** ν over X is a componentwise mapping with $\nu(x_i) = x_i$ or $\neg x_i$. We let $\pi(\vec{x})$ and $\nu(\vec{x})$ be the shorthand for $(\pi(x_1), \ldots, \pi(x_{|\vec{x}|}))$ and $(\nu(x_1), \ldots, \nu(x_{|\vec{x}|}))$, respectively.

Given two (completely specified) function vectors $\vec{f}(\vec{x})$ and $\vec{g}(\vec{y})$ with $|\vec{x}| = |\vec{y}|$ and $|\vec{f}| = |\vec{g}|$, **Boolean matching** under **NPNP-equivalence** determines if these two function vectors can be equivalent under some input negation ν_I (the first "N" of "NPNP") and permutation π_I (the first "P"), and under some output negation ν_O (the second "N") and permutation π_O (the second "P"), that is, $\vec{f}(\vec{x}) = \nu_O \circ \pi_O(\vec{g}(\nu_I \circ \pi_I(\vec{x})))$. Let $\nu \circ \pi$ denote the combined mapping of $\nu_I \circ \pi_I$ and $\nu_O \circ \pi_O$. We call $\nu \circ \pi$ a **matching solution** for NPNP-equivalence if $\vec{f}(\vec{x}) = \nu_O \circ \pi_O(\vec{g}(\nu_I \circ \pi_I(\vec{x})))$. In the sequel, we shall assume $|\vec{x}| = |\vec{y}| = n$ and $|\vec{f}| = |\vec{g}| = m$.

Boolean matching for two incompletely specified function vectors is similar, except that the functional equivalence is asserted only under the care-conditions of both function vectors.

2.2 Propositional Satisfiability

By assuming the reader's familiarity with circuit-to-CNF conversion [21] and SAT solving, including conflict-based learning [19] and other commonly used techniques in modern SAT solvers, e.g., [18, 12], we omit to provide the background knowledge.

3. MATCHING FOR NPNP-EQUIVALENCE

Given two function vectors $\vec{f}(\vec{x})$ and $\vec{g}(\vec{y})$, optionally with their care-conditions $\vec{f^c}(\vec{x})$ and $\vec{g^c}(\vec{y})$, respectively, we decide whether $\vec{f}$ and $\vec{g}$ can be NPNP-equivalent under the care-conditions. Formally speaking, this task is to decide the validity of the second-order formula

$$\exists \nu_O \circ \pi_O, \exists \nu_I \circ \pi_I, \forall \vec{x}. \bigwedge_{i=1}^{m} ((f_i^c(\vec{x}) \wedge g_i^{c*}(\nu_I \circ \pi_I(\vec{x}))) \Rightarrow (f_i(\vec{x}) \equiv g_i^*(\nu_I \circ \pi_I(\vec{x})))), \quad (1)$$

where $\vec{g^{c*}} = (g_1^{c*}, \ldots, g_m^{c*}) = \nu_O \circ \pi_O(\vec{g^c})$ and $\vec{g^*} = (g_1^*, \ldots, g_m^*) = \nu_O \circ \pi_O(\vec{g})$. It is convertible to a first-order formula as shown below.

To represent $\vec{y} = \nu_I \circ \pi_I(\vec{x})$ and $\vec{g^*} = \nu_O \circ \pi_O(\vec{g})$ (also $\vec{g^{c*}} = \nu_O \circ \pi_O(\vec{g^c})$), we introduce the 0-1 matrices

$$M_I = \begin{array}{c} \\ y_1 \\ y_2 \\ \vdots \\ y_n \end{array} \begin{array}{c} \begin{array}{cccccccc} x_1 & \neg x_1 & x_2 & \neg x_2 & \cdots & x_n & \neg x_n \end{array} \\ \begin{pmatrix} a_{11} & b_{11} & a_{12} & b_{12} & \cdots & a_{1n} & b_{1n} \\ a_{21} & b_{21} & a_{22} & b_{22} & \cdots & a_{2n} & b_{2n} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{n1} & b_{n1} & a_{n2} & b_{n2} & \cdots & a_{nn} & b_{nn} \end{pmatrix} \end{array} \quad (2)$$

and

$$M_O = \begin{array}{c} \\ g_1^* \\ g_2^* \\ \vdots \\ g_m^* \end{array} \begin{array}{c} \begin{array}{cccccccc} g_1 & \neg g_1 & g_2 & \neg g_2 & \cdots & g_m & \neg g_m \end{array} \\ \begin{pmatrix} c_{11} & d_{11} & c_{12} & d_{12} & \cdots & c_{1m} & d_{1m} \\ c_{21} & d_{21} & c_{22} & d_{22} & \cdots & c_{2m} & d_{2m} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ c_{m1} & d_{m1} & c_{m2} & d_{m2} & \cdots & c_{mm} & d_{mm} \end{pmatrix} \end{array} \quad (3)$$

respectively. By asserting

$$\sum_{j=1}^{n} (a_{ij} + b_{ij}) = 1 \qquad \text{for } i = 1, \ldots, n, \quad (4)$$

$$\sum_{i=1}^{n} (a_{ij} + b_{ij}) = 1 \qquad \text{for } j = 1, \ldots, n, \quad (5)$$

$$\sum_{l=1}^{m} (c_{kl} + d_{kl}) = 1 \qquad \text{for } k = 1, \ldots, m, \text{ and} \quad (6)$$

$$\sum_{k=1}^{m} (c_{kl} + d_{kl}) = 1 \qquad \text{for } l = 1, \ldots, m, \quad (7)$$

these matrices represent some legal mapping $\nu \circ \pi$. Valuations $a_{ij} = 1$ and $b_{ij} = 1$ indicate mapping x_j to y_i and mapping $\neg x_j$ to y_i, respectively; valuations $c_{kl} = 1$ and $d_{kl} = 1$ indicate mapping g_l to g_k^* and mapping $\neg g_l$ to g_k^*, respectively. These cardinality constraints (4), (5) (6), and (7) can be expressed by a propositional formula φ_C of $2(n^2 + m^2)$ Boolean variables, a_{ij}, b_{ij}, c_{kl}, and d_{kl}, for $i, j = 1, \ldots, n$ and $k, l = 1, \ldots, m$. By asserting the formula

$$\begin{aligned} \varphi_A \quad = \quad & \bigwedge_{i,j=1}^{n} ((a_{ij} \Rightarrow (y_i \equiv x_j))(b_{ij} \Rightarrow (y_i \equiv \neg x_j))) \wedge \\ & \bigwedge_{k,l=1}^{m} ((c_{kl} \Rightarrow (g_k^* \equiv g_l))(d_{kl} \Rightarrow (g_k^* \equiv \neg g_l))) \wedge \\ & \bigwedge_{k,l=1}^{m} ((c_{kl} \vee d_{kl}) \Rightarrow (g_k^{c*} \equiv g_l^c)), \end{aligned} \quad (8)$$

a solution to φ_C induces some unique $\nu \circ \pi$. Conversely any $\nu \circ \pi$ correspond to some unique solution to φ_C. Hence in the sequel we shall not distinguish mapping $\nu \circ \pi$ from its corresponding solution to φ_C, and vice versa. In practice, φ_C and φ_A are written in the conjunctive normal form.

Clearly solving Formula (1) is equivalent to solving the following (first-order) QBF

$$\exists \vec{a}, \exists \vec{b}, \exists \vec{c}, \exists \vec{d}, \forall \vec{x}, \forall \vec{y}. (\varphi_C \wedge \varphi_A \wedge \bigwedge_{i=1}^{m} ((f_i^c \wedge g_i^{c*}) \Rightarrow (f_i \equiv g_i^*)), \quad (9)$$

where $\vec{a} = (a_{11}, \ldots, a_{nn})$, $\vec{b} = (b_{11}, \ldots, b_{nn})$, $\vec{c} = (c_{11}, \ldots, c_{mm})$, and $\vec{d} = (d_{11}, \ldots, d_{mm})$. That is, we look for a truth assignment to variables $\vec{a}$, $\vec{b}$, $\vec{c}$, and $\vec{d}$ that satisfies φ_C and makes the miter constraint

$$\Psi = \varphi_A \wedge \bigvee_{i=1}^{m} (f_i^c \wedge g_i^{c*} \wedge (f_i \not\equiv g_i^*)) \quad (10)$$

unsatisfiable. (Note that f_i, f_i^c, g_i, g_i^c, g_i^*, g_i^{c*} in Formulas (8), (9), and (10) are meant to be the variables corresponds to functions $f_i(\vec{x})$, $f_i^c(\vec{x})$, $g_i(\vec{y})$, $g_i^c(\vec{y})$ $g_i^*(\vec{y})$, $g_i^{c*}(\vec{y})$, respectively, after the circuit-to-CNF conversion.) For simplicity, unless otherwise said we shall assume that the care conditions $\vec{f_c}$ and $\vec{g_c}$ are tautologies in the sequel. Figure 1 shows the construct of satisfiability formulation for Boolean matching under NPNP-equivalence.

The key to solving Formula (9) is to effectively reduce the search space. By exploiting the functional properties specific

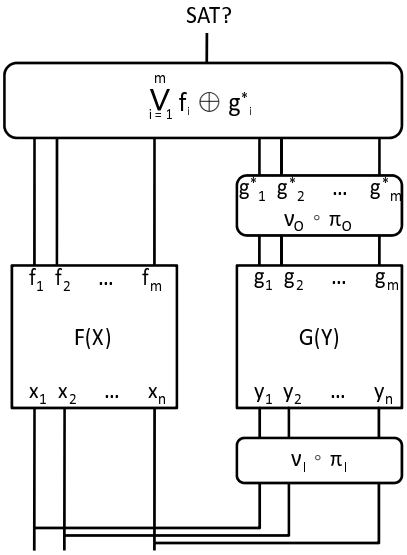

Figure 1: Boolean Matching under NPNP-equivalence

to $\vec{f}$ and $\vec{g}$ (such as variable symmetry, unateness, support sizes, and other functional properties), a preprocessing step is possible to screen out from φ_C a substantial amount of infeasible solutions. Let formula $\Phi^{\langle 0\rangle}$ characterize the remaining solutions of variables $\vec{a}$, $\vec{b}$, $\vec{c}$, and $\vec{d}$ after the preprocessing. We show below how the solution space corresponding to legal $\nu \circ \pi$ can be effectively refined in a sequence $\Phi^{\langle 0\rangle}$, $\Phi^{\langle 1\rangle}$, ..., $\Phi^{\langle k\rangle}$, (for $\Phi^{\langle i+1\rangle} \Rightarrow \Phi^{\langle i\rangle}$), along the solution search process.

3.1 Boolean Matching with Dynamic Learning

We discuss two different Boolean matching goals: to search one matching solution and to search all matching solutions.

3.1.1 Searching one matching solution

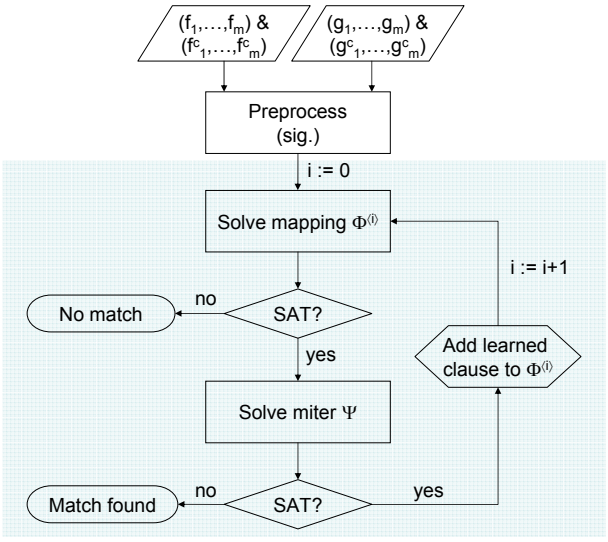

Figure 2: Search one Boolean matching solution

Figure 2 sketches the procedure for finding one solution. It takes on two input function vectors $\vec{f}$ and $\vec{g}$, possibly with their care-conditions $\vec{f_c}$ and $\vec{g_c}$. A preprocessing step is first conducted to strengthen φ_C yielding $\Phi^{\langle 0\rangle}$. It fast prunes infeasible matching solutions based on functional properties. After preprocessing, an iterative procedure is conducted between two interacting SAT solving instances to refine the solution space. The first SAT solving instance solves $\Phi^{\langle i\rangle}$, i.e., the remaining matching solutions after the ith iteration. No solution to $\Phi^{\langle i\rangle}$ indicates that f and g cannot be matched; otherwise, a solution to $\Phi^{\langle i\rangle}$ corresponds to a candidate mapping $\nu \circ \pi$. Further justification by the second SAT solving instance is needed to check whether this solution makes the miter formula Ψ unsatisfiable. If yes, the procedure terminates since $\nu\circ\pi$ is indeed a matching solution. Otherwise, the procedure learns from this (conflict) solution and strengthens $\Phi^{\langle i\rangle}$ to $\Phi^{\langle i+1\rangle}$ accordingly. This action blocks not only this current solution of $\Phi^{\langle i\rangle}$ but also other infeasible solutions from occurrence in later search. (Ordinary learning blocks only the current conflict solution.) The procedure continues with $\Phi^{\langle i+1\rangle}$ in the new iteration.

Below we show how to strengthen $\Phi^{\langle i\rangle}$ through learning.

FACT 1. *Given two function vectors $\vec{f}(\vec{x})$ and $\vec{g}(\vec{y})$ for Boolean matching under NPNP-equivalence, if $f_i(\vec{u}) \neq g_j(\vec{v})$ (respectively $f_i(\vec{u}) = g_j(\vec{v})$) for $\vec{u} \in [\![\vec{x}]\!]$ and $\vec{v} \in [\![\vec{y}]\!]$, then any mapping $\nu\circ\pi$ with $\nu_O \circ \pi_O$ having $g_i^* = g_j$ (respectively $g_i^* = \neg g_j$) and with $\nu_I \circ \pi_I(\vec{u}) = \vec{v}$ is infeasible.*

EXAMPLE 1. *For two function vectors $\vec{f}(x_1, x_2, x_3)$ and $\vec{g}(y_1, y_2, y_3)$ with $f_1(1,0,1) = 0$, $f_2(1,0,1) = 1$, $g_1(0,1,1) = 1$, and $g_2(0,1,1) = 1$, then $\vec{f}$ and $\vec{g}$ cannot be matched under NPNP-equivalence by any of the six input mappings with $\vec{y} = (\neg x_1, \neg x_2, x_3)$, $(\neg x_1, x_3, \neg x_2)$, (x_2, x_1, x_3), (x_2, x_3, x_1), $(\neg x_3, x_1, \neg x_2)$, and $(\neg x_3, \neg x_2, x_1)$ under any of the four output mappings with $g_1^* = g_1$, $g_1^* = g_2$, $g_2^* = \neg g_1$, and $g_2^* = \neg g_2$.*

Fact 1 can be rephrased in the language of formulas $\Phi^{\langle i\rangle}$ and Ψ as follows.

PROPOSITION 1. *Given two function vectors $\vec{f}(\vec{x})$ and $\vec{g}(\vec{y})$ for Boolean matching under NPNP-equivalence, if under $\nu\circ\pi$ satisfying $\Phi^{\langle i\rangle}$ there exists some $f_i(\vec{u}) \neq g_i^*(\vec{v})$ with $\vec{v} = \nu_I \circ \pi_I(\vec{u})$ for $\vec{u} \in [\![\vec{x}]\!]$ and $\vec{v} \in [\![\vec{y}]\!]$, then conjuncting $\Phi^{\langle i\rangle}$ with $\kappa = (\bigwedge_{p,q=1}^{m} l_{pq}^O \vee \bigvee_{i,j=1}^{n} l_{ij}^I) = \bigwedge_{p,q=1}^{m}(l_{pq}^O \vee \bigvee_{i,j=1}^{n} l_{ij}^I)$ for literals*

$$l_{ij}^I = \begin{cases} a_{ij}, & \text{if } v_i \neq u_j; \\ b_{ij}, & \text{otherwise,} \end{cases}$$

$$l_{ij}^O = \begin{cases} \neg c_{pq}, & \text{if } f_p(\vec{u}) \neq g_q(\vec{v}); \\ \neg d_{pq}, & \text{otherwise,} \end{cases}$$

excludes from $\Phi^{\langle i\rangle}$ exactly the input mappings $\{\nu_I' \circ \pi_I' \mid \nu_I' \circ \pi_I'(\vec{u}) = \nu_I \circ \pi_I(\vec{u})\}$ under the output mappings with $g_p^* = g_q$ (respectively $g_p^* = \neg g_q$) for $f_p(\vec{u}) \neq g_q(\vec{v})$ (respectively $f_p(\vec{u}) = g_q(\vec{v})$) for $p, q = 1, \ldots, m$.

In essence a satisfying solution to the miter formula Ψ with respect to a solution of $\Phi^{\langle i\rangle}$ reveals additional infeasible matching solutions. Letting $\Phi^{\langle i+1\rangle} = \Phi^{\langle i\rangle} \wedge \kappa$ prevents the above procedure from searching the learned infeasible solutions in later iterations. As a result, m^2 clauses with $n^2 + 1$ literals each is added in each iteration.

EXAMPLE 2. *Continue Example 1. The learned clause set is $\{(\neg c_{11} \vee a_{11} \vee b_{12} \vee a_{13} \vee b_{21} \vee a_{22} \vee b_{23} \vee b_{31} \vee a_{32} \vee b_{33})$, $(\neg c_{12} \vee a_{11} \vee b_{12} \vee a_{13} \vee b_{21} \vee a_{22} \vee b_{23} \vee b_{31} \vee a_{32} \vee b_{33})$, $(\neg d_{21} \vee a_{11} \vee b_{12} \vee a_{13} \vee b_{21} \vee a_{22} \vee b_{23} \vee b_{31} \vee a_{32} \vee b_{33})$, $(\neg d_{22} \vee a_{11} \vee b_{12} \vee a_{13} \vee b_{21} \vee a_{22} \vee b_{23} \vee b_{31} \vee a_{32} \vee b_{33})\}$. It excludes the previously listed infeasible mappings.*

To be shown in Section 3.2, the number of learned clauses and their literals can be much reduced by exploiting partially assigned satisfying solutions in SAT solving. On the

other hand, the literal count of κ can be alternatively reduced by introducing a new variable representing $\bigwedge_{p,q=1}^{m} l_{pq}^{O}$. More precisely it can be reexpressed with one clause of n^2+1 literals, one clause of m^2+1 literals, and m clauses of two literals each.

EXAMPLE 3. *The learned clause set of Example 2 can be rewritten as* $\{(e \vee a_{11} \vee b_{12} \vee a_{13} \vee b_{21} \vee a_{22} \vee b_{23} \vee b_{31} \vee a_{32} \vee b_{33}), (c_{11} \vee c_{12} \vee d_{21} \vee d_{22} \vee e), (\neg e \vee \neg c_{11}), (\neg e \vee \neg c_{12}), (\neg e \vee \neg c_{21}), (\neg e \vee \neg c_{22})\}$*, where e is a newly introduced variable.*

The pruning power of a learned clause can be characterized as follows.

PROPOSITION 2. *For Boolean matching of NPNP-equivalence, the clause set κ learned from a satisfying solution to $f(\vec{x}) \not\equiv g(\vec{y})$ with $\vec{y} = \nu \circ \pi(\vec{x})$ for some $\nu \circ \pi$ prunes $(2^m - 1)m!n!$ infeasible mappings.*

However the sets of mappings pruned by two different learned clause sets may not be disjoint.

Since there are 2^{2n} distinct truth assignments to variables $\vec{x}$ and $\vec{y}$ and each introduces a unique learned-clause set, the number of different learned-clause sets is upper bounded by 2^{2n}. This fact asserts the termination of the procedure.

PROPOSITION 3. *The Boolean matching procedure of Figure 2 for NPNP-equivalence terminates within $O(2^{2n})$ iterations.*

This upper bound is surprisingly at first glance because of its independence of the output number m. Nevertheless it is understandable because the output values are essentially induced by the input assignments.

3.1.2 Searching all matching solutions

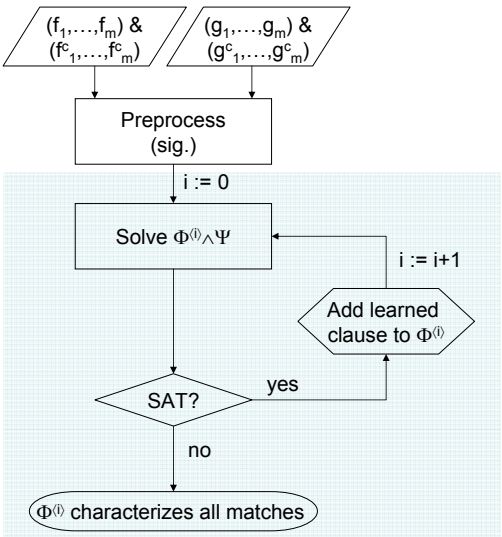

Figure 3: Search all Boolean matching solutions

When the Boolean matching objective is to find all matching solutions rather than just one, applying the procedure of Figure 2 to find the solutions one by one can be overkill. Figure 3 sketches a more effective procedure for this purpose. This procedure is the same as that of Figure 2 except that there is only one SAT solving kernel for one combined formula $\Phi^{\langle i \rangle} \wedge \Psi$.

Unlike the procedure of Figure 2 (where, effectively, variables $\vec{a}$, $\vec{b}$, $\vec{c}$, and $\vec{d}$ have higher decision orders than the other variables in making truth assignments), the procedure of Figure 3 imposes no restriction on the variable decision order. This freedom makes it much more effective. On the other hand, since this procedure terminates only when all infeasible solutions are pruned, sometimes its termination may take a long time. Nevertheless its number of SAT solving iterations has the same upper bound as that of Figure 2 for a similar reason.

PROPOSITION 4. *The procedure of Figure 3 for Boolean matching under NPNP-equivalence terminates within $O(2^{2n})$ iterations.*

Note that the computation of Figure 3 can be understood as performing quantifier elimination of variables $\vec{x}$ and $\vec{y}$ of Formula (9). The resultant formula (in terms of variables $\vec{a}, \vec{b}, \vec{c}, \vec{d}$) characterizes all matching solutions.

3.2 Strong Learning with Partial Assignment

In prior discussion of learned-clause generation, we assumed that a satisfying assignment to the miter constraint Ψ (in searching one matching solution) or $\Psi \wedge \Phi^{\langle i \rangle}$ (in searching all matching solutions) is fully assigned, that is, every variable of $\vec{x}$ and $\vec{y}$ receives some truth assignment. This assumption however makes learning conservative. Often the valuation of a small subset of the variables is sufficient to conclude the satisfiability. In fact, as to be shown, by exploiting the essential variable valuations responsible for satisfiability, the number of learned clauses and the number of literals can be substantially reduced. Thus the deductive power of learning can be much strengthened to improve Boolean matching.

A SAT solver may or may not be capable of producing partially assigned satisfying solutions. Nevertheless a minimal partial assignment can be identified from a full assignment [20]. We focus on learning from a given partial assignment.

FACT 2. *Given two function vectors $\vec{f}(\vec{x})$ and $\vec{g}(\vec{y})$ for Boolean matching under NPNP-equivalence, if $f_i(\vec{u}) \neq g_j(\vec{v})$ (respectively $f_i(\vec{u}) = g_j(\vec{v})$) for $\vec{u} \in [\![\vec{x}]\!]^{\perp}$ and $\vec{v} \in [\![\vec{y}]\!]^{\perp}$, then any mapping $\nu \circ \pi$ with $\nu_O \circ \pi_O$ having $g_i^* = g_j$ (respectively $g_i^* = \neg g_j$) and with $\nu_I \circ \pi_I$ such that there exists some $\vec{u'} \in [\![\vec{x}]\!]$ satisfying $u'_i = u_i$ if $u_i \neq \perp$ and satisfying $v'_i = v_i$ if $v_i \neq \perp$ for $\vec{v'} = \nu_I \circ \pi_I(\vec{u'})$ is infeasible.*

EXAMPLE 4. *Continue Example 1. Suppose $x_1 = 1, x_2 = 0$, and $y_1 = 0$ are the only necessary assignments for $f_1 = 0$ and $g_1 = 1$. Then any input mapping with $y_1 = \neg x_1$ or $y_1 = x_2$ under output mapping with $g_1^* = g_1$ is infeasible.*

Fact 2 can be rephrased in the language of formulas $\Phi^{\langle i \rangle}$ and Ψ as follows.

PROPOSITION 5. *Given two function vectors $\vec{f}(\vec{x})$ and $\vec{g}(\vec{y})$ for Boolean matching under NPNP-equivalence, if under $\Phi^{\langle i \rangle}$ $f_p(\vec{u}) \neq g_q(\vec{v})$ (respectively $f_p(\vec{u}) = g_q(\vec{v})$) for $\vec{u} \in [\![\vec{x}]\!]^{\perp}$ and $\vec{v} \in [\![\vec{y}]\!]^{\perp}$, then conjuncting $\Phi^{\langle i \rangle}$ with $\kappa = (\neg c_{pq} \vee \bigvee_{i,j=1}^{n} l_{ij})$ (respectively $\kappa = (\neg d_{pq} \vee \bigvee_{i,j=1}^{n} l_{ij})$) for literals*

$$l_{ij} = \begin{cases} \emptyset, & \text{if } v_i = \perp \text{ or } u_j = \perp, \\ a_{ij}, & \text{else if } v_i \neq u_j, \\ b_{ij}, & \text{else,} \end{cases}$$

excludes from $\Phi^{\langle i \rangle}$ any mapping $\nu \circ \pi$ with $\nu_O \circ \pi_O$ having $g_p^* = g_q$ (respectively $g_p^* = \neg g_q$) and with $\nu_I \circ \pi_I$ such that there exists some $\vec{u'} \in [\![\vec{x}]\!]$ satisfying $u'_i = u_i$ if $u_i \neq \perp$ and satisfying $v'_i = v_i$ if $v_i \neq \perp$ for $\vec{v'} = \nu_I \circ \pi_I(\vec{u'})$

EXAMPLE 5. *Continue Example 1. Suppose only the assignment of $x_1 = 1, x_2 = 0$ is responsible for $f_1 = 0$; $x_3 = 1$*

for $f_2 = 1$; $y_1 = 0$ for $g_1 = 1$; $y_2 = 1$ for $g_2 = 1$. Then the learned clauses are $\{(\neg c_{11} \vee a_{11} \vee b_{12}), (\neg c_{12} \vee b_{21} \vee a_{22}), (\neg d_{21} \vee a_{13}), (\neg d_{22} \vee b_{23})\}$.

4. MATCHING FOR OTHER EQUIVALENCES

The general results for NPNP-equivalence can be applied for other specialized equivalences. Moreover learning can possibly be further strengthened. As an example, we discuss only NPP-equivalence. While the partial assignment reduction can be extended straightforwardly, we elaborate learning under full assignment.

For NPP-equivalence, ν_O becomes an identity mapping, and thus d_{ij} of matrix M_O equals 0 for $i, j = 1, \ldots, m$. Proposition 1 can be accordingly simplified. Moreover learning can be further strengthened by the following observation.

FACT 3. *Given two function vectors $\vec{f}(\vec{x})$ and $\vec{g}(\vec{y})$ for Boolean matching under NPP-equivalence, if $wt(\vec{f}(\vec{u})) \neq wt(\vec{g}(\vec{v}))$ with $\vec{u} \in [\![x]\!]$ and $\vec{v} \in [\![y]\!]$, then any mapping with $\nu_I \circ \pi_I(\vec{u}) = \vec{v}$ is infeasible.*

EXAMPLE 6. *Consider the function vectors of Example 1. They cannot be matched under NPP-equivalence by any of the six input mappings with $\vec{y} = (\neg x_1, \neg x_2, x_3)$, $(\neg x_1, x_3, \neg x_2)$, (x_2, x_1, x_3), (x_2, x_3, x_1), $(\neg x_3, x_1, \neg x_2)$, and $(\neg x_3, \neg x_2, x_1)$ under any output mappings.*

Fact 3 can be rephrased as follows.

PROPOSITION 6. *Given two function vectors $\vec{f}(\vec{x})$ and $\vec{g}(\vec{y})$ for Boolean matching under NPP-equivalence, if $wt(\vec{f}(\vec{u})) \neq wt(\vec{g}(\nu_I \circ \pi_I(\vec{u})))$ for $\vec{u} \in [\![\vec{x}]\!]$ and $\nu_I \circ \pi_I$ satisfying $\Phi^{\langle i \rangle}$, then conjuncting $\Phi^{\langle i \rangle}$ with $\kappa = \bigvee_{i,j=1}^{n} l_{ij}^I$ for literals*

$$l_{ij}^I = \begin{cases} a_{ij}, & \text{if } v_i \neq u_j; \\ b_{ij}, & \text{otherwise,} \end{cases}$$

excludes from $\Phi^{\langle i \rangle}$ exactly the input mappings $\{\nu_I' \circ \pi_I' \mid \nu_I' \circ \pi_I'(\vec{u}) = \nu_I \circ \pi_I(\vec{u})\}$ under any output mappings.

5. IMPLEMENTATION ISSUES

Our current implementation adopts (input/output) support signature [13], unateness signature, and symmetry signature for preprocessing to prune infeasible matching solutions. Similar to [13], the pruning process alternates between input matching and output matching, and iterates until a fixed-point is reached. The final candidate matching solutions are used to create $\Phi^{\langle 0 \rangle}$, and passed to the subsequent main computation loop to skip invalid output matching pairs in learned-clause generation.

To realize learned-clause reduction of Section 3.2, we identify a partial assignment from a fully assigned satisfying solution to a miter constraint Ψ returned by the SAT solver. Since the miter constraint is originally in AIG, we exploit the circuit structure to derive a partial assignment. Specifically, given an AIG and a truth assignment on it, we selectively traverse it in a reverse topological order starting from a specified output towards the inputs. For each node visited, if it is an input node, then its corresponding truth value is claimed to be responsible for the output value and thus constitutes our partial assignment. Otherwise, we decide which of its fanin nodes to proceed according to the following three cases: 1) When both of its fanins are of non-controlling value, both fanins need to be visited. 2) When exactly one of its fanins is of controlling value, this fanin needs to be visited. 3) When both of its fanins are of controlling value, only one needs to be visited and we heuristically choose the one with a smaller fanin cone.

6. EXPERIMENTAL RESULTS

Our method was implemented in the C language within the ABC [6] package using MiniSAT [12] as the underlying solver. All experiments were conducted on a Linux machine with Xeon 2.5GHz CPU and 18GB RAM.

Circuits from the MCNC, ISCAS89 and ITC99 benchmark suites were chosen. Sequential circuits were converted to combinational ones by the ABC command `comb`. A circuit is matched against its synthesized version with its inputs and outputs permuted and/or negated randomly. Boolean matching under NPNP-equivalence was evaluated.

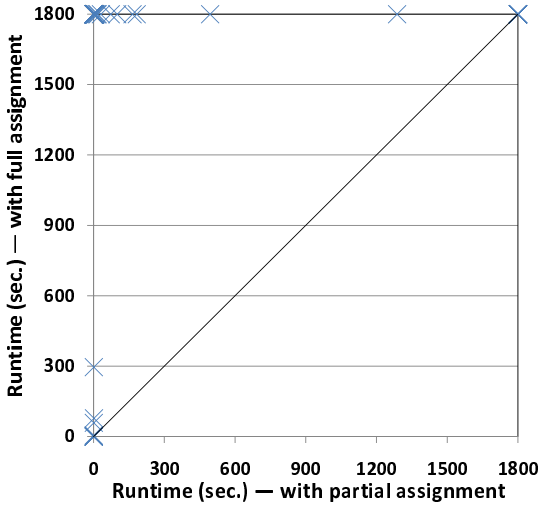

Figure 4: Runtime with and without partial assignment for NPNP-equivalence

To evaluate strengthened learning, for searching one matching solution with partial assignment reduction, Boolean matching under NPNP-equivalence solved 55 out of 67 circuits within 1800 seconds. Without partial assignment reduction, only 10 out of the 55 circuits remain solvable within the same time limit. Figure 4 compares the runtimes with (indicated in x-axis) and without (y-axis) partial assignment reduction under NPNP-equivalence. A spot in the figure corresponds to the result of matching a circuit. As can be seen, strengthened learning by partial assignment reduction achieved remarkable improvement. As a matter of fact, reducing literal counts of learned clauses strengthens the deductive power of learning, and results in fewer learning iterations. By partial assignment reduction, the average number of literals per learned clause is reduced by 71.59% and the average number of learning iterations is reduced by 89.95% under NPNP-equivalence. Since computing partial assignment costs only 0.46% of the total runtime, the overhead is negligible.

Table 1 shows the detailed information for some of the circuits (with input or output sizes greater than 60) solvable within a 1800-second limit under NPNP-equivalence. Columns "#I," "#O" and "#Node" correspond to the numbers of inputs, outputs, and AIG nodes, respectively, of each circuit. Columns "%var M_I" and "%var M_O" show the percentages of non-constant variables in matrices M_I and M_O, respectively, after signature-based preprocessing. Column "%cls" shows the percentage of remaining clauses after preprocessing. Columns "%lits" list the percentages of essential literals of partial assignment reduction. Columns "Time" list the total matching time for every circuit. Columns 8 and 9 show the data for searching one matching solution; Columns 10 and 11 show those for searching all matching solutions. As can be seen from these tables, after preprocess-

Table 1: Results for Boolean matching under NPNP-equivalence

							One Sol		All Sols	
Circuit	#I	#O	#Node	%var M_I	%var M_O	%cls	%lits	Time (sec)	%lits	Time (sec)
b04	77	74	546	1.30%	1.35%	0.02%	0.33%	4.12	6.73%	3.14
b12	126	127	1002	26.44%	26.07%	12.91%	0.05%	164.62	0.20%	315.47
b13	63	63	261	4.71%	3.25%	0.33%	0.66%	2.03	2.11%	2.08
b14	277	299	6069	1.65%	1.72%	0.12%	-	>1800	0.37%	941.74
C2670	233	140	717	12.37%	30.32%	5.75%	-	>1800	-	>1800
C5315	178	123	1773	1.04%	1.65%	0.02%	0.08%	30.31	1.67%	18.81
C7552	207	108	2074	2.68%	1.59%	0.15%	0.06%	72.04	0.63%	60.27
C880	60	26	327	2.28%	4.14%	0.07%	0.43%	1.61	8.89%	1.18
s1423	91	79	462	1.41%	1.65%	0.03%	0.14%	7.43	3.88%	5.94
s1512	86	78	470	2.49%	2.20%	0.13%	0.42%	4.08	2.15%	3.5
s15850.1	611	684	3560	0.77%	0.75%	0.03%	-	>1800	-	>1800
s3271	142	130	1102	0.93%	0.92%	0.01%	0.14%	13.28	3.10%	10.35
s3330	172	205	907	15.22%	1.38%	2.11%	0.03%	493.78	0.07%	186.68
s3384	226	209	1070	4.63%	4.21%	0.52%	0.09%	182.66	0.12%	211.48
s5378	214	228	1389	0.72%	0.83%	0.01%	0.001%	1287.14	-	>1800
s838.1	66	33	336	24.29%	3.03%	10.08%	2.67%	3.26	0.63%	4.63
s938	66	33	336	24.29%	3.03%	10.08%	2.67%	3.49	0.63%	4.68
s991	84	36	297	2.55%	2.78%	0.07%	0.13%	3.45	4.96%	2.43

ing a substantial portion of the variables of M_I and M_O is removed by asserting to constants 0 or 1. It also reduces the number of clauses representing the cardinality constrains of M_I and M_O to less than 5% on average. Although the initial CNFs can be greatly simplified, some circuit instances can still be difficult to solve. For example, even though the remaining M_I variables of C7552 is only 2.68%, there are still 2294 ($2 \times 207^2 \times 2.68\%$) variables. In addition to the preprocessing reduction, the literal counts for most learned clauses can be reduced by partial assignment to less than 10%. Preprocessing and learning strengthening are very effective to reduce search space.

7. CONCLUSIONS AND FUTURE WORK

We have formulated a decision procedure for Boolean matching of function vectors under the most general NPNP-equivalence. Essentially, the particular problem structure of Boolean matching under various equivalences allows effective conflict-driven learning beyond the reach of general QBF solving. By exploiting essential partial truth assignment in SAT solving, learned clauses can be substantially simplified and strengthened. Experimental results confirmed the indispensable power of learning and its strengthening techniques. Using our decision procedure as a platform for development, we anticipate tight integrations with other complementary techniques to make Boolean matching a practical engineering technique for broad applications.

Acknowledgments

The authors are grateful to Alan Mishchenko for suggestion about finding partial assignments, and to National Science Council for grant 99-2221-E-002-214-MY3.

REFERENCES

[1] A. Abdollahi. Signature based Boolean matching in the presence of don't cares. In *Proc. DAC*, pp. 642-647, 2008.

[2] G. Agosta, F. Bruschi, G. Pelosi, and D. Sciuto. A trasnform-parametric approch to Boolean matching. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 28, no. 6, pp. 805-817, Jun. 2009.

[3] A. Abdollahi and M. Pedram. Symmetry detection and Boolean matching utilizing a signature-based canonical form of Boolean functions. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 27, no. 6, pp. 1128-1137, Jun. 2008.

[4] M. Agrawal and T. Thierauf. The Boolean isomorphism problem. In *Proc. IEEE Symp. on Foundations of Computer Science*, pp. 422-430, 1996.

[5] L. Benini and G. De Micheli. A survey of Boolean matching techniques for library binding. *ACM Trans. on Design Automation of Electronic Systems*, vol. 2, no. 3, pp. 193-226, Jul. 1997.

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

[7] B. Borchert, D. Ranjan, and F. Stephan. On the computational complexity of some classical equivalence relations on Boolean functions. *Forschungsberichte Mathematische Logik*, Universität Heidelberg, Bericht Nr. 18, Dec. 1995.

[8] J. Cong and Y.-Y. Hwang. Boolean matching for LUT-based logic blocks with applications to architecture evaluation and technology mapping. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 20, no. 9, pp. 1077-1090, Sep. 2001.

[9] M. Cadoli, M. Schaerf, A. Giovanardi, and M. Giovanardi. An algorithm to evaluate quantified Boolean formulae and its experimental evaluation. *Journal of Automated Reasoning*, 28(2), pp.101-142, 2002.

[10] M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the Association for Computing Machinery*, vol. 7, no. 3, pp.201-215, 1960.

[11] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Communications of the ACM*, vol. 5, no. 7, pp.394-397, 1962.

[12] N. Eén and N. Sörensson. An extensible SAT-solver. In *Proc. SAT*, pp. 502-518, 2003.

[13] H. Katebi and I. Markov. Large-scale Boolean matching. In *Proc. DATE*, 2010.

[14] S. Krishnaswamy, H. Ren, N. Modi, and R. Puri. DeltaSyn: An efficient logic-difference optimizer for ECO synthesis. In *Proc. ICCAD*, 2009.

[15] C.-F. Lai, J.-H. R. Jiang, and K.-H. Wang. BooM: A decision procedure for Boolean matching with abstraction and dynamic learning. In *Proc. DAC*, pp. 499-504, 2010.

[16] A. Mishchenko, S. Chatterjee, J.-H. R. Jiang, and R. K. Brayton. FRAIGs: A unifying representation for logic synthesis and verification. Tech. Rep., EECS Dept., UC Berkeley, 2005.

[17] J. Mohnke, P. Molitor, and S. Malik. Application of BDDs in Boolean matching techniques for formal logic combinational verification. *International Journal on Software Tools for Technology Transfer*, vol. 3, no. 2, pp. 1-10, Springer, May 2001.

[18] M. Moskewicz, C. Madigan, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *Proc. DAC*, pp. 530-535, 2001.

[19] J. Marques-Silva and K. Sakallah. GRASP: A search algorithm for propositional satisfiability. *IEEE Trans. on Computers*, vol. 48, no. 5, pp. 506-521, May 1999.

[20] K. Ravi and F. Somenzi. Minimal assignments for bounded model checking. In *Proc. TACAS*, pp.31-45, 2004.

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

[22] K.-H. Wang and C.-M. Chan. Incremental learning approach and SAT model for Boolean matching with don't cares. In *Proc. ICCAD*, pp. 234-239, 2007.

[23] Z. Wei, D. Chai, A. Kuehlmann, and A. R. Newton. Fast Boolean matching with don't cares. In *Proc. Int. Symp. on Quality Electronic Design*, pp. 346-351, 2006.

[24] K.-H. Wang, C.-M. Chan, and J.-C. Liu. Simulation and SAT-based Boolean matching for large Boolean networks. In *Proc. DAC*, pp. 396-401, 2009.