Search Space Reduction for the Non-Exact Projective NPNP Boolean Matching Problem

Feng Wang, Jiaxi Zhang, Lange Wu, Wentai Zhang and Guojie Luo
Center for Energy-efficient Computing and Applications, School of EECS, Peking University, China
Collaborative Innovation Center of High Performance Computing, NUDT, China
{yzwangfeng, zhangjiaxi, wlg}@pku.edu.cn, rchardx@gmail.com, gluo@pku.edu.cn

***Abstract*—The non-exact projective NPNP (NP3) Boolean matching problem is practically significant in the applications of technology mapping, logic verification, and hardware security. However, its difficulty compels an unsatisfactory running time, and conventional algorithms for the basic NPNP Boolean matching problem perform terribly on the non-exact and projective type. In this paper, we propose a novel algorithm to solve this newly rising NP3 Boolean matching problem, and multiple types of optimization were applied to further improve the performance. Experimental results indicate that our algorithm has outstanding outcomes.**

I. Introduction

The basic NPNP Boolean matching problem [1] concerns whether two circuits are equivalent under the permutation (P) and negation (N) of inputs and outputs. More specifically, the non-exact projective NPNP (NP3) Boolean matching problem is an extension of the basic one. The term "non-exact" means that not all the outputs can be matched, and thus, the goal is to match as many outputs of the two circuits as possible. The term "projective" means that some inputs can be bridged instead of the conventional one-to-one matching.

The task in this new problem is to find out a matching configuration consisting of permutation and negation of the inputs while maximizing the number of equivalent (original or negated) output pairs. Compared to the basic version, this extension is able to model recent industrial challenges. It is more helpful to solve actual problems by studying this new problem. In fact, there are plenty of applications related to the problem, such as logic verification [2] and technology mapping [3][4].

For basic NPNP Boolean matching problem, there are some mature algorithms developed in various aspects: Spectral [3], Signature [5] and SAT [6]. However, the NP3 Boolean matching problem legitimizes unmatched outputs and the one-to-several matching pattern, terribly enlarging the search space. As a result, most of the existing algorithms cannot be applied directly in the new problem.

In this paper, we give some properties of the problem and introduce an algorithm to reduce the search space to an endurable extent. We make three main contributions to solving this problem:

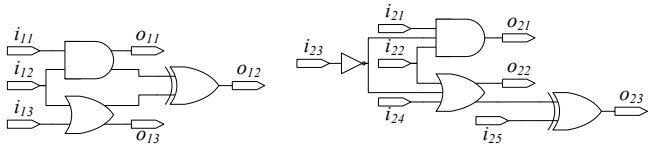
1) An algorithm framework of the NP3 Boolean matching problem is presented.
2) The support set theory is proposed so that many output matches can be excluded before further calculation.
3) Based on the symmetry detection, equivalent configurations are checked only once when matching an output pair.

The experiments are conducted on the benchmarks provided by the 2016 ICCAD Contest [7] and our algorithm greatly reduces the search space. Our algorithm can be utilized in industry tools and facilitate the Boolean matching process to some extent.

This paper is organized as follows. Section II gives some definitions and notations as well as the problem statement. Section III discusses the proposed algorithm and shows the details of the implementation. The proposed algorithms are evaluated with experimental results in Section IV. Finally Section V concludes the paper.

II. Preliminaries

A. Combinational Boolean Circuit



(a) Circuit 1 (b) Circuit 2

Fig. 1: A simple example of inputs

Fig.1 shows a simple example of inputs, which consists of two combinational Boolean circuits. A *combinational Boolean circuit* $G(V, E)$ is represented as a directed acyclic graph (DAG) specified with V, a set of vertexes, and $E \subset V \times V$, a set of edges.

A node n in a combinational Boolean circuit corresponds to a combinational gate (or an intermediate variable of its output), a primary input (PI) or a primary output (PO). It can represent a Boolean function of all PIs. For example, $o_{11} = i_{11} \wedge i_{12}$, $o_{21} = i_{21} \wedge i_{22} \wedge \neg i_{23}$.

The *support set* SS_v of a node v is defined as a set of PIs, which can reach v through some edges in E. $|SS_v|$ is the size of SS_v. For example, if v is a PI, $SS_v = \{v\}$ and $|SS_v| = 1$. In Fig. 1a, $SS_{o_{11}} = \{i_{11}, i_{12}\}$ and $SS_{o_{12}} = \{i_{11}, i_{12}, i_{13}\}$.

B. Match of Two Circuits

The match of two circuits is the core concept of this problem. A *match* $m = (m_I, m_O)$ of two circuits consists of

978-1-4673-6853-7/17/$31.00 ©2017 IEEE

an input match m_I and an output match m_O. An *input match* of two circuits cir_1, cir_2 is defined by a mapping function

$$m_I : PI_2 \rightarrow PI_1 \cup \overline{PI_1} \cup \{0, 1, \varnothing\}.$$

The value of a cir_2's PI i_2 can be decided as follows:

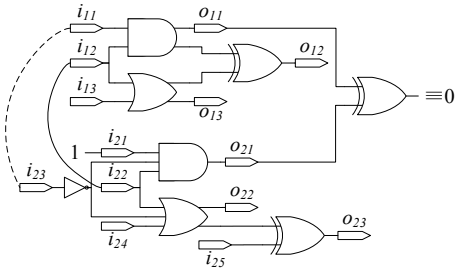
1) i_2's value equals to i_1's if $m_I(i_2) = i_1$.
2) i_2's value is different from i_1's if $m_I(i_2) = -i_1$.
3) i_2's value is a constant if $m_I(i_2) = 0$ or $m_I(i_2) = 1$.
4) i_2's value is arbitrary if $m_I(i_2) = \varnothing$.

There are mainly three notes related to this definition:

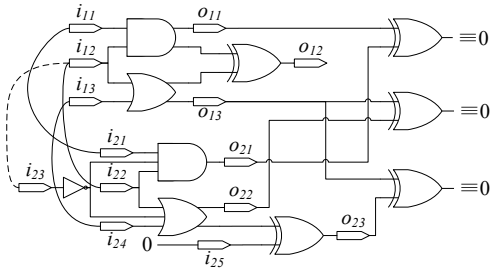
1) Different PIs in cir_2 can match the same PI in cir_1.
2) A PI in cir_2 can be bound to a constant.
3) Not all PIs need to be matched. Some of them can be ignored.

An *output match* has a similar definition where $m_O : PO_2 \rightarrow PO_1 \cup \overline{PO_1} \cup \{\varnothing\}$. $m = (m_I, m_O)$ is a *correct match* if POs' values satisfy m_O when PIs' values are decided by m_I.

Our algorithm mainly focuses on the calculation of a single PO pair match. A *PO pair match* m_{o_1,o_2} for two POs $o_2 \in cir_2$ and $o_1 \in cir_1$ is defined as the set of some m_I's, when either $(m_I, \langle o_2, o_1 \rangle)$ or $(m_I, \langle o_2, -o_1 \rangle)$ is correct.



(a) A correct match containing one PO pair $\langle o_{11}, o_{21} \rangle$.



(b) The best match containing 2 cir_1's POs and 3 cir_2's POs.

Fig. 2: Two correct matches of circuits shown in Fig. 1. PIs connected by solid(dotted) lines keep the same(different) value. Some cir_1's PIs are bound to constants. A matched PO pair is connected to a XOR gate.

Two correct matches are shown in Fig. 2. The correctness can be easily verified with a direct calculation. Fig. 2a shows that $\{\langle i_{21}, 1 \rangle, \langle i_{22}, i_{12} \rangle, \langle i_{23}, -i_{11} \rangle\} \in m_{o_{11},o_{21}}$. In fact, we can get $|m_{o_{11},o_{21}}| = 12$ which means there are 12 different configurations to match o_{11} and o_{21}.

C. Problem Statement

The input of the NP3 Boolean matching problem is two gate-level combinational circuits cir_1 and cir_2. The objective is to find a best match $m = (m_I, m_O)$.

A match m_1 is better than another one m_2 if one of the following three rules provided by the 2016 ICCAD Contest [7] is satisfied:

1) m_1 is a correct match while m_2 is not.
2) there are more cir_1's POs included in m_1 than in m_2 when Rule 1 does not apply.
3) there are more cir_2's POs included in m_1 than in m_2 when Rule 1 and 2 do not apply.

The match shown in Fig. 2b is better than the one in Fig. 2a according to the second rule. This match is one of the best matches of these two circuits as no cir_2's POs can match o_{12}.

III. Matching Algorithm

This section describes the algorithm for the NP3 Boolean matching problem. We start with a high-level overview of the algorithm in Section III-A and proceed to discuss two specific aspects: pruning by the support sets (Section III-B) and by the symmetric property (Section III-C).

A. Algorithm Overview

The flow of the matching algorithm is composed of the following steps, as shown in Fig. 3.

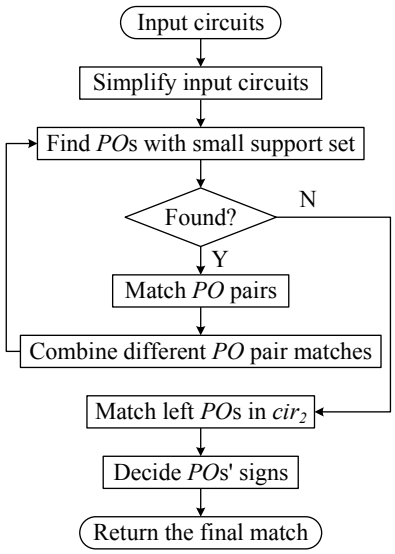


Fig. 3: The overall flow of the matching algorithm

1) At the beginning, the input circuits are simplified with ABC. The UC Berkeley ABC system [8] is a well-designed tool for logic synthesis and verification. Input circuits are converted into And-Inverter Graphs (AIG) for further use.
2) POs in both circuits with relatively small support sets are selected. PIs matched before will not be included in the next iterations. If there are no more POs, go to Step 5.
3) All PO pairs are matched by enumerating all input configurations. To check whether a configuration is correct or not, we enumerate the combination of input values and make simulation.

4) Different PO pair matches are combined to get global matches. The best global match is found by enumerating all the possibilities. Go to Step 2.
5) Unmatched POs in cir_2 are matched with POs in cir_1. Support sets of POs in cir_1 are small enough so that the brute-force method can work.
6) Finally, PO's signs are decided. This can be done by setting all PIs in cir_1 to 0 and simulating once.

It can be seen that Step 3 is most time-consuming. If $|SS_{o_1}| = |SS_{o_2}| = n$, the naive algorithm need to try $(n+1)^n$ different input configurations and spend exponential time to check each configuration. We find two methods to accelerate Step 3. The following two sections give a detailed description of these two methods.

B. Pruning by the Support Sets

The first method is related to the support set theory. Recent papers mentioned several important signatures [5] to help solve the basic NPNP problem. However, most of them do not work in this new problem due to the requirement of projective. We checked some signatures and finally developed the support set theory.

It is absolute that not all POs can be matched. The support set theory can help us exclude most of PO pairs without much calculation. When matching a PO pair, the support set theory also helps us to reduce the number of enumeration. The support set size is a good signature to represent a PO. Here gives an important theorem about the support set.

Theorem 1. *If $m_I \in m_{o_1,o_2}$, we have $\forall i_1 \in SS_{o_1}, \exists i_2 \in SS_{o_2}$, s.t. $m_I(i_2) \in \{\pm i_1\}$, supposing that there are no useless PIs in cir_1.*

Proof: (Proof by contradiction.) We assume that $i \in SS_{o_1}$ and $\forall i_2 \in SS_{o_2}, m_I(i_2) \notin \{\pm i\}$. As i is not useless, i's value can affect o_1's value when other PIs are set to certain values. However, o_2's value is a constant under the situation, which is in contradiction to $m_I \in m_{o_1,o_2}$. ■

According to Theorem 1, there are many impossible input configurations when calculating a certain m_{o_1,o_2}. For example, if $|SS_{o_1}| = |SS_{o_2}| = n$, our algorithm only need to try $n!$ different input configurations, which is much smaller than $(n+1)^n$.

Two corollaries can be derived from Theorem 1.

Corollary 1. $m_{o_1,o_2} \neq \emptyset \Longrightarrow |SS_{o_1}| \leq |SS_{o_2}|$.

Proof: (Proof by contradiction.) Otherwise at least one PI in SS_{o_1} cannot be included in m_I whatever the m_I is, which is in contradiction to Theorem 1. ■

Corollary 2. $m_O(o_2) \in \{\pm o_1\} \Longrightarrow |PI_1| - |SS_{o_1}| \leq |PI_2| - |SS_{o_2}|$. *Here PI_1 (PI_2) represents cir_1's (cir_2's) PIs included in the final input match.*

Proof: (Proof by contradiction.) Otherwise, $\exists i_1 \in PI_1 - SS_{o_1}$, $\nexists i_2 \in PI_2$, $m_I(i_2) = i_1$, which is in contradiction to Theorem 1. ■

According to Corollary 1, it is impossible for cir_1's POs to match cir_2's POs with a smaller support set. According to Corollary 2, to get a better match, we'd better match POs whose support sets have similar sizes. Thus we only try to match m_{o_1,o_2} in Step 3 where

$$|SS_{o_2}| - \delta \leq |SS_{o_1}| \leq |SS_{o_2}|.$$

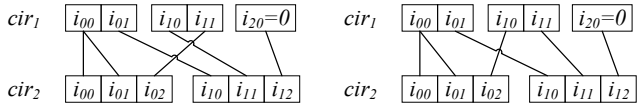
δ can be adjusted during execution of the algorithm. At the beginning of the algorithm, δ is set to 0. δ will increase gradually if there is enough time left.

C. Pruning by the Symmetric Property

The second method is related to the symmetric property of PIs. We find that for a certain PO pair, many PI matches are equivalent under the symmetry transformation. With the help of existing algorithms [9] that can detect the symmetry in the combinational Boolean circuit, we can check fewer configurations when matching a PO pair.

A PO o has a generalized form of symmetry [10] in two PIs $i_1, i_2 \in SS_o$ if the function represented by the circuit is not changed after swapping i_1 and i_2. The complementation is allowed. That is to say, $o(i_1, i_2, ...) = o(i_2, i_1, ...)$ or $o(i_1, i_2, ...) = o(-i_2, -i_1, ...)$ holds. We use $i_1 \Leftrightarrow i_2$ to represent this equivalence relation. SS_o can be divided into several equivalence classes according to it.

There is an efficient algorithm [9] to detect this type of symmetry in the combinational Boolean circuit. All symmetric PI pairs of a certain PO can be found in three steps, analysis of the circuit structure, simulation and proof with a SAT solver. When a PO's support set size is less than 100, this detecting algorithm takes less than a minute. Thus it can be included in our algorithm.



(a) A bad configuration (b) A good configuration

Fig. 4: Change the bad configuration to a good configuration. In 4a, lines started from i_{10} and i_{11} in cir_1 intersect with each other. Change their destinations to get 4b.

A natural idea is that symmetry detection can be used to reduce the number of configurations we need to check. For example, if i_{11} and i_{12} is symmetric, $\{\langle i_{21}, i_{11}\rangle, \langle i_{22}, i_{12}\rangle\}$ is equivalent to $\{\langle i_{21}, i_{12}\rangle, \langle i_{22}, i_{11}\rangle\}$ or $\{\langle i_{21}, -i_{12}\rangle, \langle i_{22}, -i_{11}\rangle\}$, which means both m_Is are correct or incorrect.

To check configurations with no repetition and no omission, we allocate each equivalence class a different label. $\{0, 1\}$ is a special equivalence class in cir_1 which is allocated the biggest label. Each PI i can get its label num_i by combining its class label and its position in the class. We only need to check the good configurations where PIs in cir_2 satisfy the property below:

$$\forall i_1, i_2 \in PI_2, i_1 \Leftrightarrow i_2 \wedge num_{i_1} < num_{i_2} \Longrightarrow num_{m_I(i_1)} \leq num_{m_I(i_2)}.$$

PIs in cir_1 should satisfy a dual property in which $m_I()$ is replaced with its inverse function $m_I^{-1}()$.

Fig.4 shows a simple example of configurations. A m_I can be visualized by connecting matched PIs with lines. PIs in an equivalence class have the same first bit in their subscripts. The property means that lines started from one equivalence class

cannot intersect. A short proof can be got from the figure. If two lines started from a equivalence class intersect, just change their destination and the match is still equivalent. The two configurations shown in Fig. 4 are equivalent and only the good one will be checked by our algorithm.

IV. Evaluation

We implemented our matching algorithm in C++. Test cases are based on the benchmarks provided by the 2016 ICCAD Contest [7], including 9 known cases. They are industrial and can represent the challenging problems in diverse applications. We used the latest ABC system [8] for circuit simplification. Experiments were run on a 2.9 GHz Intel Core i5 with 16 GB of RAM. The execution time limit was 30 minutes. Table I and II show the result.

TABLE I: The number of checked configurations

$\lvert SS_{o_1}\rvert$	$\lvert SS_{o_2}\rvert$	configurations checked when calculating m_{o_1,o_2}		
		naive algorithm	pruning by support sets	our algorithm
6	6	2.8×10^5	7.2×10^2	12
6	7	1.7×10^6	2.0×10^4	3.0×10^3
6	8	1.1×10^6	4.8×10^5	6.2×10^3
7	7	5.8×10^6	2.5×10^3	4
7	9	2.8×10^8	2.8×10^6	4.9×10^2
8	8	1.3×10^8	2.0×10^4	6.8×10^2
8	10	8.6×10^9	3.6×10^7	1.5×10^6
9	10	3.1×10^{10}	2.0×10^7	2.7×10^6
10	10	1.0×10^{11}	3.6×10^6	3.1×10^5
10	11	1.0×10^{12}	2.4×10^8	7.8×10^4
11	11	3.1×10^{12}	4.0×10^7	3.1×10^3
11	12	3.5×10^{13}	3.1×10^9	5.5×10^3

Table I lists the number of checked configurations when calculating a PO pair match. It can be seen that the number increases rapidly in the naive algorithm. The naive algorithm cannot work when the support set size comes up to 6 according to out experiments. If only pruning by support sets, we can deal with POs whose support set sizes are smaller than 9. The effect of pruning by symmetric property depends on the circuit structure and we list the best results in the benchmarks. It is possible to match POs when the support set size reaches 12 or more.

TABLE II: The performance of our algorithm

Case	Matched POs in cir_1	Matched POs in cir_2	Time(s)	The biggest support set size
1	2	3	0.21	3
2	4	4	17	8
3	3	3	5	10
4	11	11	9	1
5	2	14	10	6
6	2	2	0.35	4
7	5	5	2	11
8	8	8	6	2
9	1	1	20	12

The fifth column in Table II lists the biggest support set size of POs in the final match. It can be seen that our algorithm can deal with PO pairs whose support set is not too large. In Case 9, we successfully match a PO pair whose support set size is 12. Compared with the naive algorithm, our algorithm achieves a huge speed up.

There are some cases in which we cannot match any PO pair. After analysis of the circuit structure, we find these circuits too complicated. In most of the cases, the size of the smallest support set is more than 100 and there are over 1000 logic gates so that ABC cannot simplify them in a short time. The upper bound of the support set size is around 20 in our algorithm. A new algorithm should be designed for large cases.

V. Conclusion

In this paper, we present an effective algorithm to solve the NP3 Boolean matching problem. We propose two methods to accelerate the process of matching a PO pair. One is to exclude some matches according to the support set theory and the other one is to check less configurations based on symmetry detection. Benchmarks provided by the 2016 ICCAD Contest show that our algorithm can achieve a good result.

Our future work includes two aspects. Firstly, to improve our existing algorithm, we plan to discover more signatures that can work on the new problem, such as the distance from PIs to POs. In fact, we find that cir_1 and cir_2 are similar to each other in most test cases and matched PO pairs usually stay at the similar position. It is important how to represent this type of similarity.

Secondly, we also want to implement a new algorithm based on heuristic searching. It is time-consuming to check whether a PO pair can match and the time will increase exponentially when the support set size increases. Thus the algorithm which checks possible pairs one by one can be very slow. Heuristic searching may help us match POs with big support sets.

VI. Acknowledgements

The authors would like to thank the anonymous reviewers for their constructive comments. This work is partly supported by National Natural Science Foundation of China (NSFC) Grant 61520106004.

References

[1] H. Katebi and I. L. Markov, "Large-scale boolean matching," *Advanced Techniques in Logic Synthesis Optimizations & Applications*, pp. 771–776, 2010.

[2] 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. 207–216, 2001.

[3] L. Benini and G. D. Micheli, "A survey of boolean matching techniques for library binding," *Acm Transactions on Design Automation of Electronic Systems*, vol. 2, no. 3, pp. 193–226, 2003.

[4] J. Cong and Y. Y. Hwang, "Boolean matching for lut-based logic blocks with applications to architecture evaluation and technology mapping," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, no. 9, pp. 1077–1090, 2001.

[5] A. Abdollahi, "Signature based boolean matching in the presence of don't cares." in *Design Automation Conference, DAC 2008, Anaheim, Ca, Usa, June*, 2008, pp. 642–647.

[6] H. Savoj, M. J. Silva, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Boolean matching in logic synthesis," in *Design Automation Conference, 1992. EURO-VHDL '92, EURO-DAC '92. European*, 1992, pp. 168–174.

[7] C.-A. R. Wu, C.-J. J. Hsu, and K.-Y. Khoo, "ICCAD-2016 CAD contest in non-exact projective NPNP Boolean matching and benchmark suite," in *Proceedings of the 35th International Conference on Computer-Aided Design*, ser. ICCAD '16, 2016, pp. 40:1–40:5.

[8] R. Brayton and A. Mishchenko, "Abc: An academic industrial-strength verification tool," in *Computer Aided Verification, International Conference, CAV 2010, Edinburgh, Uk, July 15-19, 2010. Proceedings*, 2010, pp. 24–40.

[9] J. S. Zhang, A. Mishchenko, R. Brayton, and M. Chrzanowska-Jeske, "Symmetry detection for large boolean functions using circuit representation, simulation, and satisfiability," pp. 510–515, 2006.

[10] G. Wang, A. Kuehlmann, and A. Sangiovanni-Vincentelli, "Structural detection of symmetries in boolean functions," pp. 498–503, 2003.