

A Quantum Method to Match Vector Boolean Functions Using Simon’s Solver

Marco Venere, Alessandro Barenghi, and Gerardo Pelosi

Department of Electronics, Information and Bioengineering

Politecnico di Milano

20133 Milano, Italy

marco.venere@polimi.it, alessandro.barenghi@polimi.it, gerardo.pelosi@polimi.it

Abstract—The Boolean Matching Problem is a fundamental step in modern Electronic Design Automation toolchains, which allow the efficient design of large classical computers. In particular, the equivalence under negation-permutation-negation of two n -to- n vector Boolean functions requires the exploration of a super-exponential number of possible negations and permutations of input and output variables, and is widely regarded as a daunting challenge. Its classical complexity ($\mathcal{O}(n!2^{2n})$, where n is the number of input and output variables) is rarely tolerated by EDA tools, which are typically solving small instances of the Boolean Matching Problem for n -to-1 Boolean functions. In this work, we present a method to exploit the solver for Simon’s problem to speed-up the matching of n -to- n vector Boolean functions, as we show that, despite its higher complexity, it is friendlier to a quantum solver than matching single-output Boolean functions. Our solution allows saving a factor 2^n in the overall worst-case computational effort, and is amenable to combined approaches such as the so-called Grover-meets-Simon, which have the potential of reducing it below the cost of classical n -to-1 matching. We provide a fully detailed quantum circuit implementing our proposal, and compute its cost, both counting the required amount of qubits and quantum gates. We conducted an experimental evaluation employing the ISCAS benchmark suite, a de-facto standard for classical EDA to derive our sample Boolean functions.

Index Terms—Boolean Matching, NPN Equivalence, Electronic Design Automation, Quantum Computing

I. INTRODUCTION

Electronic Design Automation (EDA) defines several workflows with the goal to synthesize and implement functional and efficient digital components. Starting from a high-level description of the circuit, such workflows solve specific optimization tasks to produce the schematic diagram of the physical circuit, so as to maximize the overall performance, while minimizing power consumption and resource usage. Among these tasks, the Boolean Matching Problem (BMP) asks to test the equivalence between a portion of the designed circuit, in the form of a netlist, and the elements of the available technology libraries, which offer highly optimized digital logic components, from single gates to larger elements. This *mapping* phase allows the toolchain to exploit the resources available in the library, if matching ones are present. While the Boolean Matching Problem (BMP) is computationally demanding, common EDA toolchains are routinely called to solve instances of it. The BMP can be described as the Negation-Permutation-Negation (NPN) Boolean equivalence

problem, which asks to determine if two Boolean functions are equivalent under the negation and permutation of the inputs and the negation of the outputs. A classical computing approach enumerates all the possible negations and permutations for one of the two Boolean functions and tests exhaustively for a match.

For n -to- n vector Boolean functions, the *worst-case* computational cost of this approach requires $n!2^{2n}$ function evaluations, with n being the number of input and output variables. The said computational complexity is hardly manageable, save for very small values of n [1], [2], as a consequence, EDA toolchains usually resort to considering each output of a multi-output Boolean function as independent from the others, and solve n BMPs for n -input, single output Boolean Functions, a task with a classical worst-case complexity of $n!2^n$. The state of the art on BMP solvers employs heuristic approaches that improve the execution time against the exhaustive enumeration strategy only in the *average-case*, i.e., they exhibit a polynomial number of operations in n to establish a mismatch, while the time complexity to establish with certainty a match, given an arbitrary pair of functions (worst-case scenario), still requires a super-exponential number of operations in n . Quantum computing provides a different computation model, carrying out the computation on an exponential amount of states (derived from a polynomial number of inputs) at once, at the cost of presenting significant challenges in reading out (*measuring*, in quantum computing jargon) the correct result. While it is commonplace to achieve a square-root reduction of the computational complexity of solving a problem (e.g., via Grover’s algorithmic framework), few quantum algorithms provide an exponential speedup with respect to their classic counterparts. Among them, even fewer have a concrete impact on engineering applications, namely Shor’s factoring algorithm and Harrow, Hassidim, and Lloyd’s algorithm to solve sets of simultaneous linear equations. By contrast, Simon’s solver achieves an exponential speedup on a strawman problem, as its purpose was to show that there exists at least a case where quantum computers outperform their classical counterpart with an exponential speedup in computation, i.e., achieving an $\mathcal{O}(n)$ complexity while its classical counterpart requires $\mathcal{O}(2^n)$ operations to complete.

Contribution. In this work, we describe a quantum approach achieving a factor 2^n speedup in solving the BMP for n -to- n

vector Boolean functions. In particular, our approach builds on Simon's solver to tackle the N equivalence problem in $\mathcal{O}(n)$, yielding the corresponding exponential speedup. The key intuition resides in observing that, despite the original Simon's solver requires the input Boolean function to have specific properties (a *promise* in technical jargon), it is possible to employ it even when such properties do not hold. We extend our approach so that it also performs NP-equivalence checking with the same quantum circuit, in turn allowing its embedding in more complex solvers. Our proposed quantum algorithm solves with high probability, the NP equivalence problem in $\mathcal{O}(n^3 n!)$, which is little more than the computational effort of solving classically the P equivalence problem alone.

In doing so, we design a quantum permutator circuit that efficiently applies permutations to the information contained in a set of qubits. Our approach also allows to build a Grovermeets-Simon [3] solver for the NPN equivalence problem, combining our exponential gain with a further square-root gain to obtain an overall $\mathcal{O}(\sqrt{2^n n!})$ complexity in the worst case, i.e. below the cost for the classic solution to the BMP problem on n inputs, single output Boolean functions, which is routinely tackled by common EDA toolchains. Our approach can also be easily combined with the *signature*-based Boolean matching heuristics that are commonly employed to tackle the BMP on a classic computer. Finally, we validate the correctness of our approach on employing the Boolean functions that describe the circuits in the ISCAS benchmark suite as elements to be matched for N- and NP-equivalence, where the resource constraints of the current quantum computer simulators allow us to validate our exponential speedup.

II. PRELIMINARIES

In the following, we recall the definition of the NPN-equivalence between two n -to- n vector Boolean functions, and the structure of Simon's solver.

We consider completely specified n -to- n vector Boolean functions, $f : \{0,1\}^n \rightarrow \{0,1\}^n$, $y = f(x)$, where $x = (x_1, x_2, \dots, x_n)$ denotes the sequence of n binary input variables, $x_i \in \{0,1\}$, $1 \leq i \leq n$, while $y = (y_1, y_2, \dots, y_n)$ with $y_i \in \{0,1\}$, $1 \leq i \leq n$ denotes the sequence of n outputs. The complement of a binary variable x_i is denoted as $\bar{x}_i = 1 \oplus x_i$, $1 \leq i \leq n$, where \oplus is the exclusive-or operator. The *truth table* of a n -to- n vector Boolean function f , $T(f)$, is a vector with length 2^n , where each element is an n bit long bit string, corresponding to the output of the function on a specific n bit input value. In particular, assuming the bits taken as input are the natural binary encoding of an integer number $m \in \{0, 1, 2, \dots, 2^n - 1\}$, denoted as $x_{(m)}$, the table lists each input configuration paired to the corresponding output bitstring, $f(x_{(m)})$, in increasing order of m , i.e.: $T(f) = \langle (x_{(0)}, f(x_{(0)})), (x_{(1)}, f(x_{(1)})), \dots, (x_{(2^n-1)}, f(x_{(2^n-1)})) \rangle$.

Definition 1 (NPN-transformation). *Given a Boolean function $f : \{0,1\}^n \rightarrow \{0,1\}^n$ such that $y = f(x)$, with $x = (x_1, \dots, x_n)$, $y = (y_1, y_2, \dots, y_n)$, $x_i, y_i \in \{0,1\}$, $1 \leq i \leq n$, and $n \geq 1$, an NPN-transformation τ is defined*

as a triple $\tau = (\pi, s, o)$ composed by a permutation map $\pi : (x_1, \dots, x_n) \rightarrow (x_{\pi(1)}, \dots, x_{\pi(n)})$, and two binary strings $s \in \{0,1\}^n$ and $o \in \{0,1\}^n$. The application of an NPN-transformation τ to the function f , yields a Boolean function $g = f \circ \tau$ such that: for all $x \in \{0,1\}^n$ its input/output mappings are derived as $g(x) = f(\pi(x \oplus s)) \oplus o$.

As an example, let $f(x_1, x_2, x_3) = [x_1 x_2 \oplus x_2 x_3, x_2 x_3, x_1 \oplus x_3]$ be a Boolean function with $n = 3$ inputs and outputs, where the multiplication and addition of two terms correspond to the Boolean \wedge (and) and \oplus (xor) operations, respectively. The application of the NPN transformation $\tau = (\pi : (x_1, x_2, x_3) \rightarrow (x_2, x_3, x_1), s = 010, o = 100)$ to the function f , $g = f \circ \tau$, results in the replacement of x_2 with \bar{x}_2 (i.e., $f(x \oplus s)$), the application of a permutation that replaces x_1 with \bar{x}_2 , \bar{x}_2 with x_3 , and x_3 with x_1 (i.e., $f(\pi(x \oplus s))$), and to the application of the negation mask o to the result (i.e., $f(\pi(x \oplus s)) \oplus o$), allowing to yield $g(x_1, x_2, x_3) = [\bar{x}_2 x_3 \oplus x_3 x_1, x_3 x_1, \bar{x}_2 \oplus x_1]$.

Definition 2 (NPN-equivalence). *Two n -to- n vector Boolean functions, $f(x)$ and $g(x)$, with $x = (x_1, \dots, x_n)$, $y = (y_1, y_2, \dots, y_n)$, $x_i, y_i \in \{0,1\}$, $1 \leq i \leq n$, and $n \geq 1$, are NPN-equivalent if there exists an NPN transformation $\tau = (\pi, s, o)$, such that $g(x) = f(x) \circ \tau(x) = f(\pi(x \oplus s)) \oplus o$.*

Definition 3 (Decision Boolean Matching Problem). *Given two n -to- n , $n > 1$, vector Boolean functions, f and g , determine if there exists at least one NPN-transformation τ such that $f \circ \tau$ is equal to g .*

The decision BMP is a well-known computationally hard problem. Indeed, the problem can be shown to be *complement-non deterministic polynomial time* hard (or co-NP hard, employing the lexicon of complexity theory). This can be seen since, if it is possible to determine whether a generic Boolean function is a tautology (i.e., true for all of its possible input values), then it is also possible to determine the Boolean Matching between two functions and vice-versa. While the former reduction is immediate, as checking for the unsatisfiability of *miter* formulae are the common way to test for Boolean equivalence of two functions (i.e., checking if the formula $f \oplus g$ is such that $f \oplus g = 0$ for all possible input values, where the functions are denoted as f and g), the latter reduction is obtained extending the candidate single-output Boolean function of which it must be tested if it is a tautology, with $n - 1$ constant outputs, and call the oracle solving the n -to- n BMP asking whether the said extended function matches the constant one $g(x) = [0, 0, \dots, 1]$ with $s = o = [0, \dots, 0]$ and π being the identity permutation. Determining if a Boolean function is a tautology is a co-NP complete problem, i.e., solving it allows to solve any problem in co-NP, the set of complementary problems to the ones admitting a solving algorithm which resides in the complexity class of *non-deterministic polynomial time* (NP) problems. Problems in co-NP-complete (co-NP-C) are in close relation to the ones in the NP-complete class (NP-C), and therefore it is widely believed that co-NP-C \neq P, that is, no polynomial time

algorithm solving them on a deterministic machine (which would put the problem in P), is possible [4]. Since BMP is at least as hard as any co-NP problem (indeed, solving it, also solves any co-NP problem by virtue of solving the tautology problem), any classical polynomial time solver for BMP is extremely unlikely to exist.

Similarly, it is largely believed that the complexity class of *bounded-error quantum polynomial time* (BQP) algorithms, i.e., the class of computational problems that can be solved in polynomial time by a quantum computer with bounded error in the answers, has null intersection with the co-NP-C problem set [4]. As a consequence, it is also extremely unlikely that a quantum polynomial time solver for BMP exists. Therefore, the best expectations which can be set for a quantum speedup in the solution of the BMP are improvements over its classical computational complexity, currently rated to be $\mathcal{O}(n!2^{2n})$, which do not yield a polynomial time (in n) algorithm.

Furthermore, no polynomial time search-to-decision reduction is known for the BMP. Indeed, assuming the existence of a fast solver for the *decision BMP* applied to two NPN-equivalent functions, there is no general algorithm able to exhibit the corresponding NPN-transformation, i.e., solve the *search BMP* for the given functions. Indeed, the current state of the art approaches to solving the decision BMP in the worst case do not help the user in solving the corresponding search BMP. Indeed, they still require the user to find on his own which one is the NPN-transformation making the equivalence hold, with a computational cost of $\mathcal{O}(n!2^{2n})$ bit operations.

We note that, even restricting the output of the Boolean functions being matched to a single output bit, the BMP does not have a subexponential classical complexity. Indeed, classical approaches based on canonical forms retain a complexity which is significantly higher than exponential. For instance, in [5] the computational complexity is $\mathcal{O}(N^2 + Nn^3)$, where N is the number of nodes of the Boolean Decision Diagram representing the function, which, in the worst-case is 2^n , resulting in an overall complexity of $\mathcal{O}(2^{2n} + 2^n n^3)$. Similarly, recent SAT based approaches also exhibit a superexponential complexity, as, for instance [6], which solves the BMP as an incremental Boolean function learning problem, obtaining a worst-case complexity of $\mathcal{O}(2^{2n^2+n})$. Finally, we point out that [7] introduces a quantum approach to solve NP-equivalence, yet with the strong assumption that it is a-priori known whether the circuits are equivalent or not.

A. Simon's quantum solver

In the following, we recall the structure of Simon's problem and its details, together with its corresponding quantum solver [8].

Definition 4 (Simon's Problem). *Consider $x, y \in \{0, 1\}^n$, with $n \geq 1$, a Boolean function $f : \{0, 1\}^n \rightarrow \{0, 1\}^n$, and the promise that there is a binary string $s \in \{0, 1\}^n$ such that $f(x) = f(y)$ if and only if $x \oplus y \in \{0^n, s\}$; find s .*

A different way of stating Simon's problem promise is: whenever f has collisions among its outputs, these are pair-

wise and the corresponding preimages are x and $x \oplus s$ for all $x \in \{0, 1\}^n$ (where $s \in \{0, 1\}^n$ is obviously unique). A triple (or more) collision cannot take place. Indeed, assuming a Boolean function f as per the Simon's problem, if $x, y, z \in \{0, 1\}^n, n \geq 1$ are such that $f(x) = f(y) = f(z)$, then this chain of equalities implies $x = y \oplus s$ and $x = z \oplus s$, for a proper $s \in \{0, 1\}^n$. Applying a bitwise xor between the right sides and left sides of the latter equalities, respectively, it can be easily derived that $y = z$, proving the impossibility of a triple collision for the given Simon's function. As a consequence, in the worst case a deterministic classical solver for Simon's problem must evaluate the given function for half of all possible input configurations plus one, and in case of collisions, check if the input pairs have the same xor difference. Consequently, the corresponding worst-case computational cost is $2^{n-1} + 1$ evaluations of the function f (also referred to as queries to the oracle f). A probabilistic classical solver can apply a birthday paradox based approach yielding the solution with a probability of at least $\frac{3}{4}$ employing $\Omega(2^{n/2})$ oracle queries [9]. By contrast, Simon's quantum algorithm finds the value s (promised to exist) employing $\mathcal{O}(n)$ evaluations of the circuit implementing function f , i.e., employing $\mathcal{O}(n)$ queries to the quantum oracle O_f , in the worst case. Figure 1 provides an example of the quantum circuit for Simon's Algorithm with $n = 3$.

The n -qubit Hadamard gate applied to a quantum state $|x\rangle$, with $x \in \{0, 1\}^n$, yields $H^{\otimes n}|x\rangle = \frac{1}{\sqrt{2^n}} \sum_{y \in \{0, 1\}^n} (-1)^{x \cdot y} |y\rangle$, where $x \cdot y$ denotes the inner product between x and y interpreted as vectors in \mathbb{Z}_2^n , i.e., given $x = (x_1, \dots, x_n) \in \mathbb{Z}_2^n, y = (y_1, \dots, y_n) \in \mathbb{Z}_2^n$, with $x_i, y_i \in \{0, 1\}, 1 \leq i \leq n$, their inner product is computed as $x \cdot y = \bigoplus_{i=1}^n x_i y_i$. The quantum solver for Simon's problem [8] operates in five steps:

- (i) Initialize a $2n$ qubit register as $|0\rangle^n |0\rangle^n$ and apply $H^{\otimes n}$ to the first n qubits obtaining $\frac{1}{\sqrt{2^n}} \sum_{x \in \{0, 1\}^n} |x\rangle |0\rangle$;
- (ii) query the oracle O_f employing the first n qubits as the inputs of the function, yielding $\frac{1}{\sqrt{2^n}} \sum_{x \in \{0, 1\}^n} |x\rangle |f(x)\rangle$;
- (iii) Apply another $H^{\otimes n}$ to the first n qubits to obtain the state $\frac{1}{\sqrt{2^n}} \sum_{x \in \{0, 1\}^n} \left(\frac{1}{\sqrt{2^n}} \sum_{y \in \{0, 1\}^n} (-1)^{x \cdot y} |y\rangle \right) |f(x)\rangle = \sum_{y \in \{0, 1\}^n} |y\rangle \left(\frac{1}{2^n} \sum_{x \in \{0, 1\}^n} (-1)^{x \cdot y} |f(x)\rangle \right)$;
- (iv) measure the first n qubits (in the computational basis). The probability of getting a specific state $|y\rangle$ is computed summing up all the probabilities of possible measurements of the second n qubits that must have the first ones in the state $|y\rangle$, i.e.:

$$p_y = \text{Prob}[\text{measure state } y] = \left\| \frac{1}{2^n} \sum_{x \in \{0,1\}^n} (-1)^{x \cdot y} |f(x)\rangle \right\|^2.$$

We now analyze the effects of being either in the $s = 0^n$ or in the $s \neq 0^n$ cases of Simon's promise.

Case $s = 0^n$. In this case the $f(x)$ is a one-to-one function, therefore iterating over all the possible input bitstrings x , yields all possible bitstrings as the range of $f(x)$. Thus, the corresponding kets are all mutually orthogonal. As a consequence $p_y = 2^n \left(\frac{(-1)^{x \cdot y}}{2^n} \right)^2 = \frac{1}{2^n}$, i.e., the value measured from the first register, which ranges from 0 to $2^n - 1$, is modeled as a uniformly distributed random variable.

Case $s \neq 0^n$. For each $x \in \{0,1\}^n$ it is promised that $f(x) = f(x \oplus s) = z \in \text{range}(f)$, therefore

$$p_y = \left\| \frac{1}{2^n} \sum_{z \in \text{range}(f)} \left((-1)^{x \cdot y} + (-1)^{(x \oplus s) \cdot y} \right) |z\rangle \right\|^2 = \left\| \frac{1}{2^n} \sum_{z \in \text{range}(f)} (-1)^{x \cdot y} (1 + (-1)^{s \cdot y}) |z\rangle \right\|^2.$$

This latter expression allows to easily observe that $p_y(x) = 0$, when the internal product of the measured state value y is such that $s \cdot y = 1$, since the $1 + (-1)^{s \cdot y}$ factor vanishes. This in turn implies that only y values such that $s \cdot y = 0$ can be measured. We now determine the probability distribution of measuring one such y value. The $1 + (-1)^{s \cdot y}$ factor is then equal to 2, therefore

$$p_y = \left\| \frac{1}{2^n} \sum_{z \in \text{range}(f)} (-1)^{x \cdot y} |z\rangle \right\|^2.$$

As it can be noted, the term $(-1)^{x \cdot y}$ is equal to 1 in case $x \in \{0, s\}$, while it takes values 1 and -1 the same number of times in case $x \notin \{0, s\}$, as they represent the parity of the bits in $y \in \{0,1\}^n$, when considering only the positions where they are also set in x . As a consequence, when $s \neq 0^n$, the probability of observing the measured value $y \in \{0,1\}^n$ is $p_y = \left(\frac{1}{2^n} + 0 + \dots + \frac{1}{2^n} + 0 + \dots + 0 \right)^2 = \frac{1}{2^{n-1}}$ and $s \cdot y = 0$.

(v) Running the quantum part of the Simon's solver $n + c$ times (see steps (i)-(iv)), with $c \geq 0$, allow us to collect the bitstrings $y^{(1)}, y^{(2)}, \dots, y^{(n+c)} \in \{0,1\}^n$, which in turn can be employed to establish the following set of equations in \mathbb{Z}_2^n , i.e.: $s \cdot y^{(i)} = 0 \Leftrightarrow \bigoplus_{j=1}^n s_j y_j^{(i)} = 0$, $1 \leq i \leq n + c$, where $s_j y_j^{(i)}$ denotes the multiplication modulo 2 of the binary values s_j and $y_j^{(i)}$.

Noting that $n - 1$ linearly independent equations would allow to derive a unique non-null value for the unknown bitstring s , the probability that some subset of $n - 1$ equations of the $(c - 1)n + 1$ collected ones are actually linearly independent can be derived as follows. Let us proceed by randomly picking a sequence of $n - 1$ equations out of the cn available ones, and by placing the vector of coefficients of the equation at hand as a column vector of a $(c + n) \times (n - 1)$ matrix.

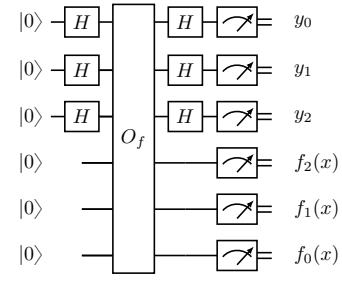


Fig. 1: Quantum circuit solving Simon's problem for a $f : \{0,1\}^3 \rightarrow \{0,1\}^3$ function. O_f is a quantum oracle for f

The chances for the first column to be linearly independent coincide with the probability of observing a non-zero vector, i.e., $1 - \frac{1}{2^{n+c}}$. The chances for the second column to be linearly independent from the first column are $1 - \frac{1}{2^{n+c-1}}$; the chances for the third column to be linearly independent from the first and the second column are $1 - \frac{1}{2^{n+c-2}}$. Continuing in this way, the probability q of having $n - 1$ columns being linearly independent is $q = \left(1 - \frac{1}{2^{n+c}}\right) \left(1 - \frac{1}{2^{n+c-1}}\right) \dots \left(1 - \frac{1}{2^c}\right)$. Observing that $q > 1 - \sum_{i=0}^c \frac{1}{2^{n+c-i}} > 1 - \frac{1}{2^{c+1}}$, it is easy to determine the constant c such that the chance of failing to derive s is less than $10^{-6} \approx 2^{-20}$. Indeed, the disequality $\frac{1}{2^{c+1}} < 2^{-20}$ is equivalent $c > 19$, which in turn points toward the choice of collecting a number of equations $n + c$, with $c = 20$.

Once a system of $n - 1$ linearly independent equations has been successfully solved, a candidate bit-mask $s' \neq 0^n$ is found, and it is possible to confirm it checking if $f(0^n) = f(s')$ is true. If this is the case the Simon bitmask is $s = s'$, otherwise $s = 0^n$.

III. QUANTUM NPN-EQUIVALENCE SOLVER

In this section, we describe our quantum approach to the BMP on n -inputs, n -outputs Boolean functions by first tackling input negations (N-equivalence), and subsequently augmenting our quantum algorithm to tackle input permutations (NP-equivalence).

A. Quantum Solver for N-equivalence

We solve the N-equivalence problem exploiting the quantum solver for the *Simon's Problem*, which benefits from an exponential speedup w.r.t. a classical solver [8], taking care of the fact that the promise in the statement of the Simon's problem does not hold in our case.

Given two generic n -input, n -output Boolean functions $f : \{0,1\}^n \rightarrow \{0,1\}^n$ and $g : \{0,1\}^n \rightarrow \{0,1\}^n$, $n \geq 2$, functionally representing the two digital components to compare for N-equivalence, we design a vectorial function $h : \{0,1\}^{n+1} \rightarrow \{0,1\}^{n+1}$ with an additional auxiliary input variable $x_0 \in \{0,1\}$ and an additional output, constant with respect to f and g . Such a function can be specified as a sequence of $n + 1$ single-output functions h_i , $0 \leq i \leq n$, each of which is defined over the same domain $\{0,1\}^{n+1}$.

Specifically, $h_i(x_0, x_1, \dots, x_n) =$

$$= \begin{cases} x_0 \oplus \bar{x}_0, & i = 0 \\ \bar{x}_0 f_i(x_1, \dots, x_n) \oplus x_0 g_i(x_1, \dots, x_n), & i \in \{1, \dots, n\} \end{cases}$$

It is worth noting that each h_i can be interpreted also as a Shannon cofactor decomposition of a generic Boolean function as f, g are two generic functions.

If f and g are N-equivalent, i.e., there is an n -bit binary string $s' \in \{0, 1\}^n$ such that for all $x' \in \{0, 1\}^n$ it holds that $f(x') = g(x' \oplus s')$, then also for all $x = x_0 || x'$, with $x_0 \in \{0, 1\}$, it holds that $h(x \oplus 1 || s') = h(x)$, where $||$ denotes a binary string concatenation. As a consequence, if f and g are N-equivalent, there are 2^n colliding pairs among the output bitstrings of h , each of which corresponding to a pair of pre-images fulfilling the promise of Simon's problem with $s=1 || s'$. This in turn implies that, if it were the case that no other binary string $t \in \{0, 1\}^{n+1}$ were such that $h(x) = h(x \oplus t)$, Simon's solver applied to h would yield with certainty the string s . Contrarywise, if f and g were not N-equivalent, Simon's solver would never yield any non null s .

However, for random choices of f and g , regardless of the fact that f and g are N-equivalent or not, it may be the case that h exhibits collisions on an arbitrary subset of input configurations, i.e.:

$$\exists t \in \{0, 1\}^{n+1}, \exists x \in S \subseteq \{0, 1\}^{n+1} \text{ s.t. } h(x) = h(x \oplus t).$$

This condition corresponds to a generalization of Simon's problem known as Simon's problem with *approximate promise*, which matches the one tackled in [10], Theorem 1, where the purpose of the generalization was to break the security of authenticated modes of operation for block ciphers.

In the case of Simon's problem with approximate promise, the function h is such that there might be more than one input difference causing collisions on some or every output values. If the number additional collisions is too high, measuring the outputs of the quantum circuit in Simon's solver will not yield equation coefficients allowing to build a full-rank linear system from which the correct value of the binary string s is derived. The authors in [10] prove that it is still possible to employ Simon's solver even in this case of the approximate promise, tolerating the fact that the retrieved bitmask t may be different from the one matching the Simon's promise, with some probability. Indeed, to quantify the number of measurements, required by Simon's solver to obtain the bitmask s for the h function, instead of another $t \neq s$, they introduced the following definition of error probability

$$\varepsilon(h, s) = \max_{t \in \{0, 1\}^{n+1} \setminus \{0^{n+1}, s\}} \Pr[h(x) = h(x \oplus t)], x \in \{0, 1\}^{n+1}$$

which captures the maximum probability of an output collision being induced by an input difference t not equal to the one promised in the statement of Simon's problem, s . In all cases where the probability $p = \varepsilon(h, s)$ is such that $0 < p < 1$, the authors of [10] prove that one execution of Simon's solver returns the correct value of s after cn queries, with probability

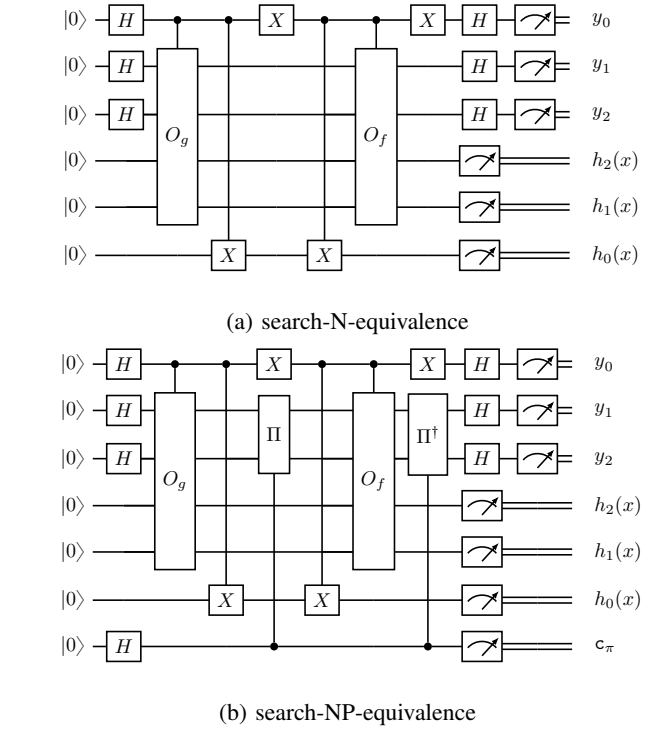


Fig. 2: Quantum circuits for Simon's solver employed for the search-N-equivalence (a) and the search-NP-equivalence (b).

$\geq 1 - 2^n \left(\frac{1+p}{2}\right)^{cn}$. This allows us to state that applying Simon's solver will indeed yield the correct solution of the problem with a probability which converges to 1 exponentially fast in the number of the inputs of h , as long as the value of the multiplicative factor c on the number of Simon's solver measurements is at least 4. Concretely, this states that, with $4n$ or more measurements from Simon's solver, our approach will provide the correct negation bitmask s' which makes f N-equivalent to g with an error probability which is exponentially decreasing in the number of function inputs.

Concerning a quantitative evaluation of $\varepsilon(h, s)$, we note that its expected value, assuming a uniform random pick for both h and s , is $\Theta(\frac{n}{2^n})$, as proved in [11]. This in turn states that, in the average case (over all possible function pairs $f, g : \{0, 1\}^n \rightarrow \{0, 1\}^n$, constituting h) $\varepsilon(h, s)$ is significantly far from 1. While practical cases for f and g are not uniformly randomly sampled, the experimental results we provide show that the obtained results match the aforementioned probability.

Summing up, applying Simon's solver to h , whenever f, g are N-equivalent functions, will yield the correct value of the input difference with exponentially low error probability in n .

We therefore employ Simon's solver, to search for an input string s for the h function we defined, which in turn will, whenever equal to 1. $s', s' \in \{0, 1\}$ yield the negation transformation mapping f into g . The quantum circuit employed by Simon's solver in this case embeds the oracle for h , implementing $|x\rangle_{n+1}|0\rangle_{n+1} \mapsto |x\rangle_{n+1}|h(x)\rangle_{n+1}$, which in turn embeds the oracles O_f and O_g , implementing $|x\rangle_n|0\rangle_n \mapsto |x\rangle_n|f(x)\rangle_n$ and $|x\rangle_n|0\rangle_n \mapsto |x\rangle_n|g(x)\rangle_n$, respectively. Fig-

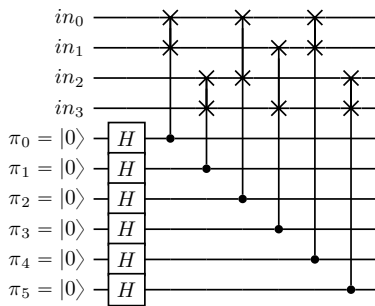


Fig. 3: Quantum permutation network computing the superposition of all permutations of $n = 4$ input qubits

ure 2a shows an example circuit where $n = 2$. The first qubit from the top, encoding the value of the first input bit x_0 , selectively enables O_f and O_g , and determines the constant output of $h_0(x) = x_0 \oplus \bar{x}_0$, as defined previously. The following two qubits, representing the remaining part of the input to h , $x \in \{0, 1\}^{2+1}$, are input to the oracles. The $n + 1 = 3$ qubits starting from the bottom are the ones on which the output of the h function is computed. Such an approach has a computational complexity of $2(cn) \in \mathcal{O}(n)$ (with $c \geq 4$ as stated before) queries to the oracle for h (i.e., $2(cn)$ calls to the oracles O_f and O_g), plus $2(n+1)$ Hadamard gates. This computation is in turn followed by a $\mathcal{O}(n^3)$ classical computation to solve sets of simultaneous Boolean linear equations, yielding s . Overall, this corresponds to a polynomial computation effort in the time taken to compute f and g , plus a polynomial time in the input size. Compared to classical heuristics from Section II, such a solver has stronger correctness guarantees, together with an exponential speedup.

B. Solving NP-equivalence

To extend the proposed approach to tackle N-equivalence to a Grover-meets-Simon solver for NP and NPN equivalence, it is needed to have a single quantum circuit tackling both NP and NPN equivalences. To this end, we start by augmenting our N-equivalence solving strategy to cope with permutations of the input bits. Indeed, solving the NP-equivalence is equivalent to solving the N-equivalence between f and g , while applying to g any possible permutation of the input bits.

In order to allow the application of a Grover-meets-Simon approach, we need to adopt our solver so that it computes the solution to Simon's problem on all possible permutations on the inputs in superposition. To do so, we modify the oracle for function h to apply all the possible permutations to the input qubits of function g , before evaluating O_g . We compute this employing a quantum permutation network [12], obtained repurposing the non-quantum *sorting network* proposed in [13], a parallel structure able to sort an array of n integer values. A classic sorting network is built with compare-and-swap units that act on two variables, and swap their values according to which one is bigger. In the design by Beneš [13], the network is built placing $(n/2)(2\log(n) - 1)$ compare-and-swap units in a circuit with logarithmic depth (in the number of elements

to be sorted n). To build our quantum permutation circuit, we replace each classical compare-and-swap component in a Beneš sorting network with a controlled-swap gate, having its control qubit prepared as $H|0\rangle$. In this fashion, the control qubits of the entire network, which drive whether a given swap is made or not, will be in a uniform superposition of all their basis states. This, in turn, induces an essentially uniform superposition of all the possible permutations of the input qubits corresponding to the elements to be sorted. Figure 3 provides an example of a quantum permutator.

The quantum NP-equivalence circuit is built by inserting a Beneš-based quantum permutation network Π right before the component computing the oracle O_f , acting on the qubits representing the inputs of f , and a second network Π^\dagger right after O_f for uncomputation, which is required by the quantum circuit of Simon's solver. This design requires $2n+2+(n/2)(2\log(n)-1)$ qubits, of which $2n+2$ are needed for the computation of the oracle for h (namely, $n+1$ for the inputs of h and $n+1$ its outputs), plus $(n/2)(2\log(n)-1)$ control qubits for the quantum permutation network. Figure 2b depicts the quantum circuit to establish the NP-equivalence between 2-inputs 2-output Boolean functions f and g .

This circuit can be employed in Simon's solver by measuring the Boolean function outputs $h_i(x)$, for $0 \leq i \leq n-1$ in Figure 2b) on a random x , and the control qubits of the n -bit permutator π (a single control qubit denoted as c_π in our case). This measurement collapses the states of the remaining qubits, $|y_3y_2y_0\rangle$, so that a measure will yield a bit-vector y such that its internal product with the unknown bit-vector s of the Simon's problem on h equals zero (i.e., $y \cdot s = 0$), i.e., the negation mask, if existing, that makes f equivalent to $\pi \circ g$. To derive the value of the bit-vector s , it is necessary to collect at least (cn) measurements, with $c \geq 4$, for each possible permutation applied to the inputs of g . If the described approach is employed as a standalone solution, such a collection is an instance of the *coupon collector's problem*, it is therefore necessary to collect at least $n! \log(n!) cn = n!(c \log(n)) n$, measurements, and classify them according to which permutation they pertain. The classification can be easily performed as we measure the c_π permutation network control qubits, which uniquely identify the applied permutation. Once this is done, we solve the equation sets according to Simon's Algorithm, and find whether f and g are NP-equivalent under a given permutation π and an input negation string s , or not. In this approach, we exhaustively explore the whole permutation space. This results in a $\mathcal{O}(n!n^3)$ overall worst-case running time, with a factor $\frac{2^n}{n^3}$ on the classical worst-case computation time of $n! 2^n$ for NP equivalence checking.

We note that it is possible to start solving one of the simultaneous equations sets as soon as cn equations are gathered and, as soon as a single NP-equivalence is found, terminate the procedure earlier. We also point out that the execution of the quantum circuit can be parallelized, thus benefiting from a linear performance scaling, by subdividing the permutation space to be explored across different quantum

TABLE I: Number of qubits and depth of the NP-equivalence circuits for $n = 4$ and N-equivalent circuits for $n = 8$. We show depth both with and without oracles. Output bits marked with \dagger denote custom modifications required for testing. Marked output bits with the same name may generally denote different random components.

Assembled Component	Number of inputs (n)	Number of qubits	Depth With Oracles	Depth W/O Oracles
(74182-PBo, 74182-CN _x , 74283-S0, 74181-F2 \dagger)	4	16	1,614	2
(74182-PBo, 74283-S0, 74182-CN _y \dagger , 74283-S1 \dagger)	4	16	1,154	2
(74182-GBo \dagger , 74182-CN _y \dagger , 74182-CN _z \dagger , 74283-S1 \dagger)	4	16	665	4
(74182-CN _x , 74181-F2 \dagger , 74182-GBo \dagger , 74182-CN _z \dagger)	4	16	1,268	2
(74182-PBo, 74182-CN _x , 74283-PBo \dagger , 74181-F2 \dagger)	4	16	2,181	4
(74283-S2 \dagger , 74181-F0, 74182-GBo \dagger , 74182-CN _z \dagger , 74182-CN _y \dagger , 74182-GBo \dagger , 74283-S1 \dagger , 74283-S1 \dagger),	8	18	206,749	-1
(74182-GBo \dagger , 74182-CN _y \dagger , 74182-CN _z \dagger , 74283-S1 \dagger , 74182-GBo \dagger , 74182-CN _y \dagger , 74182-CN _z \dagger , 74283-S1 \dagger),	8	18	266,802	3
(74283-S1 \dagger , 74283-S2 \dagger , 74182-CN _y \dagger , 74182-GBo \dagger , 74181-F0, 74283-S1 \dagger , 74182-CN _z \dagger , 74182-GBo \dagger),	8	18	206,676	-2

computers. This can be done by fixing some of the swap gates control qubits to a fixed value, instead of initializing them to $H|0\rangle$. Each quantum circuit may even consider only one specific permutation. Such approach reduces the number of required measurements to $n! \cdot cn$. Yet, the logarithmic improvement is negligible, if compared to the cost required to synthesize $n!$ quantum circuits. Analogously, our method can be paired with classical heuristics that reduce the number of admissible permutations beforehand.

In the case of employing a Grover-meets-Simon approach to solve the NP-equivalence problem, the presented circuit will represent Grover’s oracle. We defer the full investigation of a Grover-meets-Simon approach to further works.

IV. EXPERIMENTAL RESULTS

In this section, we report the results of our experimental campaign considering realistic Boolean functions.

We employed the IBM Qiskit quantum programming and simulation toolkit [14] version 1.0.2 on a server with two AMD Epyc 7551 32-Core CPUs and 512 GiB of RAM, as our testing environment, and the Qiskit ClassicalFunction utility for the synthesis of the Boolean oracles required in III, based on Tweedledum library [15]. We ran all our tests on Qiskit Aer version 0.14.1, the software quantum circuit simulation backend of Qiskit. Being a simulator, it is free from geometric neighbourhood constraints among qubits, thus producing architecture-independent results. We compiled our circuits with the maximum available optimization level.

Our approach using Simon’s solver may yield, with a probability growing exponentially small in the number of inputs n , a solution t which is unrelated to f and g being equivalent. We are able to obtain reliable results observing that i) if f and g are N-equivalent, spurious solution(s) t will be obtained with a significantly lower probability than the correct one s and, ii) if f and g are not N-equivalent, Simon’s

solver will provide spurious t solution(s) with essentially the same probability. This implies that, in case of equivalence, multiple executions of Simon’s solver will yield an essentially deterministic bitmask s , plus some rare spurious values, i.e., it will output strings with a *skew statistical distribution*; otherwise, the output s will exhibit essentially a *uniformly random statistical distribution*. As a consequence, we filter the solution of the N- and NP-equivalence checks by evaluating the empirical Boolean metric $\text{MODE}(\mathcal{H}) \geq \sum_{s \in \mathcal{H}} v(s)$, which proved to discriminate the skewness of Simon’s solver output distribution in practice. Here, \mathcal{H} is the measured histogram of the distribution of s , $v(s)$ denotes the value of a column of the histogram, and $\text{MODE}(\mathcal{H})$ denotes the desired result as the mode of the histogram.

To validate our approach, we selected our Boolean function benchmarks from the 74X-Series belonging to the ISCAS Circuits benchmark suite [16]. In particular, we considered 3 components: 74181, a 4-bit ALU; 74182, the circuit for the carry-generation signals of a 4-bit carry-look-ahead adder; and 74283, a 4-bit adder. In order to increase the number of comparisons to test, we built n -input, n -output Boolean functions f and g out of these components as follows. First of all, we split the components considering each of their output bits as a separate n -to-1 Boolean function. Subsequently, we assemble the obtained functions in fresh n -input, n -output components to test for NP-equivalence. To obtain a test set acting as a ground truth for NP-equivalence, we generated, for each of these components, additional NP-equivalent versions, by randomly applying input negations and permutations. To further augment our dataset, we also apply small modifications to the obtained functions. Through these procedures, we obtained a total of 200 different components with $n \in \{4, 8\}$. To obtain the quantum oracles required for Simon’s solver for all the components, we quantum synthesized each one of them providing the n Boolean functions with n -inputs and 1-

output independently onto quantum circuits. This choice was dictated by limits of the `ClassicalFunction` method in Qiskit. Furthermore, since Qiskit Aer 0.14.1 only supports the simulation of up to 34 qubits on our configuration, we tested NP-equivalence for components with $n = 4$, and N-equivalence with components with $n = 8$. For the same reasons, we cannot simulate higher values of n . Regarding correctness, we observed an equivalence detection rate of 98.5%, which is lower than the error probability $\frac{n}{2^n}$ derived in Section III. We ascribe such improvement to the MODE heuristic, which selects the result coming from the highest amount of linear systems, thus improving robustness to errors.

Willing to evaluate the quantum circuit complexity of our solver, we report in Table I the number of qubits and circuit depth required to realize the NP-equivalence testing circuit for 8 pairs of compared Boolean components. In particular, the first component of the pairs is produced from the benchmark as previously described, and referred to as Assembled Component in the table. The second one is instead obtained from the first through random negations and permutation of input bits, for positive cases, while it is a different assembled component for negative cases. Due to the limits of classical simulators, we do not consider execution time. We note that the number of qubits required for the solver circuit does only depend on the number of inputs n , and grows as $2n+2+\frac{n}{2}(2\log_2(n)-1)$ for the NP-case ($n = 4$) and as $2n+2$ for the N-case ($n = 8$). The depth of the implemented circuits is essentially dominated by the one of the implemented Boolean functions. To provide an estimation of the circuit complexity required by the algorithm, independently from the size of the oracles, we report both the depth of the whole quantum circuit and its depth after subtracting the depth of the used oracles, transpiled for the same backend and with the same optimization level. We can observe that the complexity of the circuit, save for the oracles, is negligible. The reported differences are affected by small variations: compiler optimizations may produce a final circuit whose depth is less than the sum of the depth of the transpiled components. This also justifies the presence of negative depths.

V. CONCLUDING REMARKS

We presented a quantum circuit to solve the Boolean Matching Problem. Our approach relies on reducing the N-equivalence problem to Simon’s problem on a specifically crafted Boolean function, while it deals with the NP-equivalence case by making use of a quantum sorting-network. We provide a super-polynomial speedup compared to classical worst-case complexity, moving a step towards the efficient matching of large n -input n -output Boolean circuits. The approach can either be integrated with classical ones, acting as a pre-filter to reduce the search space or working on a permutation subspace, or embedded in a full-quantum Grover-meets-Simon approach, while retaining the same speedup. We tested our approach employing functions from the widely adopted ISCAS circuit benchmark [16] on the Qiskit Aer simulator, and provide the main key performance indicators for the synthesis of the quantum circuit.

ACKNOWLEDGEMENTS

This work has been partially supported by the ICSC - Italian “National Research Center in High Performance Computing, Big Data and Quantum Computing”, and the SERICS project (PE000014) in the Italian NRRP MUR program funded by the EU - NGEU.

REFERENCES

- [1] C. Lai, J. R. Jiang, and K. Wang, “Boolean matching of function vectors with strengthened learning,” in *2010 International Conference on Computer-Aided Design, ICCAD 2010, San Jose, CA, USA, November 7-11, 2010*, L. Scheffer, J. R. Phillips, and A. J. Hu, Eds. IEEE, 2010, pp. 596–601. [Online]. Available: <https://doi.org/10.1109/ICCAD.2010.5654215>
- [2] H. Katebi and I. L. Markov, “Large-scale Boolean matching,” in *Design, Automation and Test in Europe, DATE 2010, Dresden, Germany, March 8-12, 2010*, G. D. Micheli, B. M. Al-Hashimi, W. Müller, and E. Macii, Eds. IEEE Computer Society, 2010, pp. 771–776. [Online]. Available: <https://doi.org/10.1109/DATE.2010.5456949>
- [3] G. Leander and A. May, “Grover Meets Simon - Quantumly Attacking the FX-construction,” in *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part II*, ser. Lecture Notes in Computer Science, T. Takagi and T. Peyrin, Eds., vol. 10625. Springer, 2017, pp. 161–178. [Online]. Available: https://doi.org/10.1007/978-3-319-70697-9_6
- [4] S. Arora and B. Barak, *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009. [Online]. Available: <http://www.cambridge.org/catalogue/catalogue.asp?isbn=9780521424264>
- [5] J. R. Burch and D. E. Long, “Efficient Boolean function matching,” in *IEEE/ACM ICCAD 1992*, 1992, pp. 408–411. [Online]. Available: <https://doi.org/10.1109/ICCAD.1992.279337>
- [6] C. Lai, J. R. Jiang, and K. Wang, “BooM: a decision procedure for boolean matching with abstraction and dynamic learning,” in *Proceedings of the 47th Design Automation Conference, DAC 2010, Anaheim, California, USA, July 13-18, 2010*, S. S. Sapatnekar, Ed. ACM, 2010, pp. 499–504. [Online]. Available: <https://doi.org/10.1145/1837274.1837398>
- [7] T.-F. Chen and J.-H. R. Jiang, “Boolean matching reversible circuits: Algorithm and complexity,” *arXiv preprint arXiv:2404.12184*, 2024.
- [8] P. Kaye, R. Laflamme, and M. Mosca, *An Introduction to Quantum Computing*. Oxford University Press, 2006.
- [9] R. Cleve, “Quantum Information Processing - Quantum Algorithms (I),” Lecture Notes - Institute for Quantum Computing and Cheriton School of Computer Science University of Waterloo. <https://cleve.iqc.uwaterloo.ca/resources/QIC-710-F21/Qic710QuantumAlgorithmsPart1.pdf>, 2021.
- [10] M. Kaplan, G. Leurent, A. Leverrier, and M. Naya-Plasencia, “Breaking Symmetric Cryptosystems Using Quantum Period Finding,” in *CRYPTO 2016*, vol. 9815. Springer, 2016, pp. 207–237. [Online]. Available: https://doi.org/10.1007/978-3-662-53008-5_8
- [11] J. Daemen and V. Rijmen, “Probability distributions of correlation and differentials in block ciphers,” *J. Math. Cryptol.*, vol. 1, no. 3, pp. 221–242, 2007. [Online]. Available: <https://doi.org/10.1515/JMC.2007.011>
- [12] S. Perriello, A. Barenghi, and G. Pelosi, “Improving the Efficiency of Quantum Circuits for Information Set Decoding,” *ACM Transactions on Quantum Computing*, Jul. 2023. [Online]. Available: <https://doi.org/10.1145/3607256>
- [13] V. E. Beneš, “On rearrangeable three-stage connecting networks,” *The Bell System Technical Journal*, vol. 41, no. 5, pp. 1481–1492, 1962.
- [14] Qiskit contributors, “Qiskit: An Open-source Framework for Quantum Computing,” <https://qiskit.org/>, 2023.
- [15] B. Schmitt and G. De Micheli, “tweedledum: A compiler companion for quantum computing,” in *2022 Design, Automation & Test in Europe Conference & Exhibition, DATE 2022, Antwerp, Belgium, March 14-23, 2022*. IEEE, 2022. [Online]. Available: <https://doi.org/10.23919/DATE54114.2022.9774510>
- [16] M. Hansen, H. Yalcin, and J. P. Hayes, “ISCAS High-Level Models,” <http://web.eecs.umich.edu/~jhayes/iscas.restore/benchmark.html>, 1999, accessed: May, ’24.