QUANTUM LEAP: A POLYNOMIAL-TIME ALGORITHM FOR DUALIZATION PROBLEMS

Jyoti*

Open Scholar, Department of Mathematics
Jind, Haryana, India

***Email ID**: jyotimorptr@gmail.com*

Accepted: 09.02.2023 **Published**: 01.03.2023

Keywords: Quantum Leap, Polynomial.

Abstract

In the context of two prime monotone boolean functions, denoted as f : $\{0,1\}^n \rightarrow \{0,1\}$ and g : $\{0,1\}^n \rightarrow \{0,1\}$, the Dualization Problem arises when we aim to determine whether g is the dual counterpart of f. In other words, we seek to establish if, for all possible inputs (x1,...,xn) belonging to {0, 1}n, the values of f(x1,...,xn) and $\bar{g}(\overline{x1},...,\overline{xn})$ are identical. This Dualization Problem can also be formulated as a decision problem: when presented with two monotone prime boolean functions, f and g, our objective is to determine if g indeed serves as the dual function to f.

This paper introduces a quantum computing algorithm specifically designed to address the decision variant of the Dualization Problem efficiently. Our algorithm exhibits a polynomial time complexity, offering a significant improvement over classical methods.

Paper Identification

**Corresponding Author*

Introduction

A boolean function is considered to be monotone if, for any two boolean vectors v = (*v1*,...,*vn*) and *w* = (*w1*,...,*wn*), it holds that $vi \leq w_i$ for all $i \in \{1,...,n\}$, which implies that $f(v) \leq f(w)$.

The dualization problem involves a monotone boolean function f : $\{0,1\}^n \rightarrow \{0,1\}$ expressed in a prime disjunctive normal form (DNF), which means it is irredundant. The objective is to find the prime DNF of a monotone boolean function *g* that satisfies the condition f(x) = $\bar{g}(\bar{x})$ for all x $\in \{0,1\}^n$. The dualization problem, known as dual, refers to the decision version and is defined in the following manner: When presented with two prime monotone boolean functions, *f* and *g*, the question arises: is g the dual of *f*? The issue of dualization, along with its corresponding decision version, is a significant challenge that has garnered attention in various research domains, including machine learning, data mining, and artificial intelligence (AI) among others (refer to the relevant literature and citations within for further details). Utilizing the notation borrowed from the aforementioned source, we represent the monotone boolean functions *f* and *g* in Disjunctive Normal Form (DNF) as follows:

$$f = \bigvee_{I \in F} \bigwedge_{i \in I} x_i$$

and

$$g = \bigvee_{J \in G} \bigwedge_{j \in J} x_j$$

Let I and J be subsets of the set {1,2,...,n}, and let F and G represent the sets of prime implicants of functions f and g, respectively. The optimal deterministic classical computing algorithm for solving the dual problem exhibits a computational complexity of $O(N^{o(\log N)})$, where *N* represents the count of prime implicants for functions *f* and *g*. Specifically, *N* is calculated as the sum of the cardinalities of sets *F* and *G*, denoted as |*F*| and |*G*|, respectively. The assessment of the complexity status of the dualization problem and its corresponding decision version is a significant unresolved matter. The self-dualization problem is of equal interest, as it involves determining whether a monotone boolean function possesses self-duality. The complexity of the problem is equivalent to that of the dual problem, as it can be simplified by applying self-dualization to the function $yf \vee zg \vee yz$, where y and z represent two supplementary boolean variables. This paper presents a novel quantum computing algorithm for efficiently solving the dual (and self-dual) problem within a polynomial time complexity framework.

Methods

The variable x is interpreted in two different ways throughout the following. In some instances, it represents a boolean or binary n-dimensional vector. In other instances, it represents the decimal expression of the binary vector. If *x* represents the decimal value of the binary vector (*x*1,..., *x*n), then the decimal value of the binary vector ($\bar{x}$1,...,$\bar{x}$n) is $\bar{x}$= 2^n - *x* -1 can be obtained. The paper commences by introducing a set of propositions that will be frequently referenced throughout the subsequent sections.

Proposition 1: A necessary condition for two monotone boolean functions, *g* and *f*, expressed in their Disjunctive Normal Form (DNF), to be mutually dual is that

$$I \cap J \neq \emptyset \text{ for every } I \in F \text{ and } J \in G \qquad (1)$$

Proof. If there are implicants I ∈ F and J ∈ G such that I J =, then let x = (x1,...,xn) be such that xi = 1 if I ∈ I and xi = 0 if i /∉ I. This will allow us to determine whether or not there is a contradiction in the statement. Since *f(x)* = 1 = g($\bar{x}$), it is obvious that *f* and *g* cannot both be dual to each other.

According to Proposition 1, in order for f to be a self-dual, each implicant of F has to intersect each and every other implicant.

Lemma 2. *Let's assume that f is a self-dual. Therefore, f is in equilibrium, which means that for half of the values of x it is 0, and for the other half it is 1.*

Proof. If x is between 0 and 2n, then x must equal 2n minus x plus one. In addition, since f is a self-dual function, we have that *f(x)* ≠ *f*($\bar{x}$) for any values of x ranging from 0 to 2^n inclusive. Because of this

$$\sum_{x=0}^{2^{n-1}-1} f(x) + \sum_{x=2^{n-1}}^{2^n-1} f(x) =$$

$$\sum_{x=0}^{2^{n-1}-1} f(x) + \sum_{x=0}^{2^{n-1}-1} f(2^n - x - 1) =$$

$$\sum_{x=0}^{2^{n-1}-1} [f(x) + f(\overline{x})] = 2^{n-1}$$

Lemma 3. Let *f* be a monotone boolean function that satisfies both (1) and its DNF expression, and let it be $\sum_{x=0}^{2^n-1} f(x) = 2^{n-1}$ stated as a DNF. If and only if this condition holds, then *f* is a self-dual.

Proof. Lemma 2 demonstrates the need of the situation. As for the sufficiency, let's assume that, $\sum_{x=0}^{2^n-1} f(x) = 2^{n-1}$ and let's also assume by contradiction that *f(x)* = *f*($\bar{x}$) for any $0 \leq x < 2^n$ range of values of *x*. Given that (1) is true, there must be an implicant *I* such that *xi* = 1 for any *i* that fall inside *I* when *f(x)* = 1. But if *I* overlaps all of the other implicants of *F*, then *f(x)* must equal zero. To put it another way, *f(x)* + f($\bar{x}$) 1 for all values of *x*. Because of this, we need to ensure that *f(z)* = f($\bar{z}$) = 0 for any *z* with 0 to 2^n in the range. But given that

$$2^{n-1} = \sum_{x=0}^{2^n-1} f(x) = \sum_{x=0}^{2^{n-1}-1} [f(x) + f(\overline{x})] \leq 2^{n-1}$$

In order to avoid a contradiction, we must ensure that *f(x)* + *f*($\bar{x}$) = 1 for every possible value of *x* between 0 and 2^{n-1}.

The Hamming weight of the integer 0 up to and including *x* up to and including 2^n is denoted by the number of ones in the binary representation of *x*. Another way to put this is to say that if *x* = *(x1,...,xn)* is a binary vector, then $w(x) = \sum_{i=1}^{n} x_i$.

According to what we've discussed, the difficulty of the dualization issue is evaluated with regard to the total size of f and g, or more specifically, with respect to the formula *N* = |*F*| + |*G*|. In addition, it is stated in that the number n of variables that are used by boolean functions is always lower than the values represented by the symbols |*F*||*G*|. However, there are cases of the self-dual issue in which N = O(2^n), such as the one that is shown below as an illustration.

If *n* > 4 odd, then the following boolean function *φ*, whose collection of implicants is to be considered: *F* is the collection of all subsets of the set "{1,...,*n*}" with the cardinality "[*n/2*]," where [*a*] is the smallest integer that is either bigger than or equal to *a*.

Lemma 4. The function denoted by is a self-dual, and the total number of implicants it has is $\binom{n}{\lceil n/2 \rceil}$..

Proof. To a trivial degree $|F| = \binom{n}{\lceil n/2 \rceil}$. |I∩ J| = |I|+|J| = 2[n/2] > n This results in a contradiction due to the fact that the number of variables is n. If there are two implicants I and J such that I ∩ J = then |I∪ J| = |I|+|J| = 2[n/2] > n. Therefore, it follows that (1) is correct.
Because every implicant I of (x) has cardinality |I| = n/2, we may deduce that (x) = 0 for each x such that w(x) n/2. This is because w(x) is less than n/2. On the other hand, if w(x) is less than

n/2, then (x) is equal to 1, since if we think of x as a binary vector, we will always find an implicant I such that xi = 1 for all i that are less than I. This is because x is a binary vector. It is now a simple matter to confirm that the equation |x: w(x) n/2| equals 2n1. In accordance with Lemma 3, is a self-dual.

The quantum computing algorithm

If we are provided with two boolean functions, *f* and *g*, we can construct the function *h(x)* as follows: h(x) = f(x)⊕ $\bar{g}(\bar{x})$, where ⊕ represents the sum modulo two.

Take note that h may be derived from f and g by using a logic gate count that is linear in nature. If f(x) equals $\bar{g}(\bar{x})$ for all values of *x*, then *h(x)* must equal 0 for all values of x. We construct a black box called Uh that is capable of performing the transformation |x>|y> to |x ⟩ |y to h(x)i, where x may range from 0 to 2n. In the Deutsch-Joshua algorithm, the blackbox plays an important role. We haven't gotten to the qubits yet, but the measurements for the first

$$\frac{1}{2^n}\sum_{z=0}^{2^n-1}\sum_{x=0}^{2^n-1}(-1)^{x\cdot z+h(x)}|z\rangle$$

When h(x) = 0 for every x, the chance of measuring for |z⟩ i = |0⟩ is equal to 1 because of the fact that this is the case.

$$\frac{1}{2^n}\sum_{x=0}^{2^n-1}(-1)^{h(x)}|0\rangle = |0\rangle$$

Therefore, the next observation is as follows:

Remark 5. Let there be two monotones prime boolean functions called f and g, and let h equal f minus g. If we measure a value |x⟩ = |0⟩ at the conclusion of the Deutsch-Joshua method with the blackbox function h, then f is not the dual of g.

Following is an example of a straightforward quantum algorithm that can be derived from Lemma 1, Remark 2, and Lemma 3 to determine whether or not a function f is self-dual.

Algorithm Quantum Dual

Input: A black box≤ *x* < 2^n and U_f which performs the transformation *f*(*x*) ∈ {0,1}

$f(x)\rangle$, for $0 \le x < 2^n$ and $f(x) \in \{0,1\}$

|*x*⟩ |*y*⟩ → |*x*i|*y* ⊕

Output: *True* if *f* is self-dual and *False* otherwise.

Procedure:

1. Checking whether or not f is balanced may be done using the Deutsch-Joshua algorithm. If the result of running the Deutsch-Joshua method is equal to |0i, then the result should be False, and the program should end.

2. Let $h(x) = f(x) \oplus \overline{f}(\overline{x})$. Check whether or not equal to h remains constant by using the Deutsch-Joshua method. If the result of running the Deutsch-Joshua algorithm does not equal |0i, then the program will end with the result False and it will quit.

3. Use the Quantum Counting algorithm to count the number of x such that $f(x) = 1$ using $t = \lceil n/2 \rceil$ qubits to measure the phase angle. If the measurement at the end of the algorithm is $|y\rangle$ and if y 6= $2t{-}2$ then output *False* and exit.

4. Use the Grover algorithm to find an x such that $f(x) = f(\cancel{x})$. If such x is found then output *False* and exit.

5. Output *True*

The complexity of the algorithm is dominated by the complexity of the Quantum Counting and of the Grover algorithms. Both algorithms achieve a complexity on the number of quantum gates which is $O(2^{n/2})$ while the best deterministic classical computing algorithm has time complexity of $O(N^{o(\log N)})$. However, we saw in Lemma 4 that a self-dual function can have a number of implicants in its DNF equal to $\binom{n}{\lceil n/2 \rceil}$ which is asymptotic to $O(2^n)$. Therefore, we have that $N \leq 2^n$ from which we obtain that the complexity of our quantum algorithm for the dualization problem is $O(\sqrt{N})$.

References

1. D. Angluin. Queries and Concept Learning. Machine Learning, 2:319–342, 1996.
2. C. Bioch and T. Ibaraki. Complexity of identification and dualization of positive Boolean functions. Information and Computation, 123:50–63, 1995.
3. G. Brewka, J. Dix, and K. Konolige. Nonmonotonic Reasoning – An Overview. Number 73 in CSLI Lecture Notes. CSLI Publications, Stanford University, 1997.
4. N. H. Bshouty. Exact Learning Boolean Functions via the Monotone Theory. Information and Computation, 123:146–153, 1995.