# On the Query Computation and Verification of Functions

Hirakendu Das Ashkan Jafarpour Alon Orlitsky Shengjun Pan Ananda Theertha Suresh
University of California San Diego
{hdas,ashkan,alon,s1pan,asuresh}@ucsd.edu

***Abstract*—In the query model of multi-variate function computation, the values of the variables are queried sequentially, in an order that may depend on previously revealed values, until the function's value can be determined. The function's computation query complexity is the lowest expected number of queries required by any query order. Instead of computation, it is often easier to consider verification, where the value of the function is given and the queries aim to verify it. The lowest expected number of queries necessary is the function's verification query complexity. We show that for all symmetric functions of independent binary random variables, the computation and verification complexities coincide. This provides a simple method for finding the query complexity and the optimal query order for computing many functions. We also show that if the symmetry condition is removed, there are functions whose verification complexity is strictly lower than their computation complexity, and mention that the same holds when the independence or binary conditions are removed.**

## I. Introduction

The worst-case *query* complexity or *decision-tree* complexity [2, 3, 4, 5, 8] of a multi-variate function is the number of function inputs that must be queried in the worst case to determine the function's value. For example, the worst-case query complexity of $xy \vee \overline{x}z$ is 2, as the value of $x$ determines which of $y$ or $z$ needs to be queried to determine the function's value.

A multi-variate function is *symmetric* if its output remains unchanged under all input permutations. Many functions encountered in engineering and science are symmetric, including *parity*, *threshold*, and *delta*, as well as most statistical measures such as *median*, *mode*, *max*, and order statistics. Essentially all results in this paper concern symmetric functions.

It is easy to see, e.g. [6], that the worst-case query complexity of all non-constant symmetric functions is $n$. Kowshik and Kumar [7] recently considered the *expected* query complexity of computing symmetric functions. For expected complexity, the optimal query order depends not only on the function, but also on the underlying distribution. They found an optimal query order for threshold functions of independent $Ber(p_i)$ random variables. In particular, they showed that for these functions the optimal query order does not depend on the precise probabilities $p_i$, but only on which is largest, second largest, etc.

To simplify and extend arguments for finding the optimal query order, [1] defined the expected *verification* query complexity of a function to be the lowest expected number of inputs that need to be revealed to convince an observer of the value of the function. For example, consider the OR function $X_1 \vee \ldots \vee X_n$, where each $X_i \sim Ber(p_i)$ independently of each other. To verify that the OR is 1, it suffices to show that one of the $X_i$'s is 1, hence for moderate values of the $p_i$'s, the expected number of variables that need to be revealed is small, whereas verifying that the OR is 0, requires checking that all variables are 0, hence all $n$ variables must always be queried. Note that verification complexity differs from *certificate* complexity [5, 2], where all input values are known in advance and can be used to determine the optimal query order.

Verification complexity was used in [1] to provide a simpler proof that the query order for computing threshold functions presented in [7] is optimal, to derive an optimal query order for computing delta functions, and, observing that the value of all binary-input symmetric functions depends only on the number of ones or *weight* of the input, to find an optimal query order for symmetric functions that vary over any three consecutive input weights.

In this paper, we extend the results in [1] and show that for all symmetric functions of independent binary inputs, optimal expected verification and computation complexity are equal. The symmetry, independence, and binary conditions are necessary in the sense that if any of them is relaxed then there are functions whose expected verification complexity is strictly lower than their expected computation complexity.

The rest of the paper is organized as follows. In Section II, we formally define the problem of computation and verification. In Section IV, we show the equality of verification and computation for general symmetric functions of independent binary inputs. In Section V, we give an example of a non-symmetric function of independent binary inputs where the verification and computation complexities differ. Similar examples showing that the independence and binary assumptions are necessary will be provided in the paper's full version.

## II. Notation and Formulation

Throughout the paper, we assume that $f$ is a symmetric function of $n$ independent binary random variables $X_1, X_2, \ldots, X_n$, where $X_i \sim Ber(p_i)$ and the $p_i$'s are known in advance, and without loss of generality, $1 > p_1 \geq p_2 \geq \ldots \geq p_n > 0$. We also let $\bar{p}_i \stackrel{\text{def}}{=} 1 - p_i$ and $X^j \stackrel{\text{def}}{=} X_1, \ldots, X_j$.

To compute $f(x^n)$, we query the inputs sequentially. A *policy* $\mathcal{P}$ is a rule that at any given time, based on prior

978-1-4673-2579-0/12/$31.00 ©2012 IEEE

query outcomes, determines whether querying should stop or continue, and in the latter case, which input should be queried next. Once an input is queried, its full value is revealed. $\mathcal{P}$ *computes* $f$, if for all $x^n$, when $\mathcal{P}$ stops querying, $f$ can be determined.

Let $N(x^n)$ be the number of inputs a policy $\mathcal{P}$ queries for input $x^n$. The expected query complexity of $\mathcal{P}$ is

$$C(\mathcal{P}) \stackrel{\text{def}}{=} E[N(X^n)] = \sum_{x^n \in \{0,1\}^n} P(x^n)N(x^n),$$

and the *computation complexity* of $f$ is

$$C(f) \stackrel{\text{def}}{=} \min_{\mathcal{P}} C(\mathcal{P}),$$

where the minimum is taken over all policies computing $f$. Any policy for $f$ with complexity $C(f)$ is called *optimal*. In general, there might be several optimal policies.

***Example** 1:* Consider the *threshold* function

$$\Pi_\theta(x) = \begin{cases} 1 & \text{if } x \geq \theta, \\ 0 & \text{otherwise,} \end{cases}$$

and let $f(X_1, X_2) = \Pi_1(X_1 + X_2)$. If $X_i$ is queried first, the expected number of queries is $1 \cdot p_i + 2 \cdot (1 - p_i) = 2 - p_i$. Since we assume $p_1 \geq p_2$, the policy querying $X_1$ first is always optimal. □

For general functions, it is possible to express the computation complexity in terms of the input probabilities and optimize the query order. But since the possible number of policies is exponential in $n$, the problem may not be easy to solve for all functions.

An alternative approach for this problem was proposed in [1]. Instead of finding the optimal policy for computing a function, the simpler problem of finding an optimal policy for verifying the function's value was considered. It was shown that for a class of functions, the two policies coincide.

In the *verification* of a function $f$, we are given the value of $f(x^n)$, and are asked to query the inputs to verify that this is indeed the function's value. As with computation, we apply a policy that determines which input variable to query and when to stop, the value of the function can be determined. But the difference is that now we can use different policies based on the value of $f(x^n)$. It is therefore easy to see that a *verification policy* is just a collection of computation policies, one for each value of $f(x^n)$, and the advantage of verification is that for each value of $f$ we can choose a policy that minimizes the expected number of queries for the given value of $f(x^n)$.

The difference between verification and computation complexity is perhaps easiest to demonstrate via a non-symmetric function of dependent random variables.

***Example** 2:* Let $f : \{0,1\}^n \to \{1, ..., n\}$ be such that $f(x^n) = i$ iff $x_i = 1$ and all other $x_j$'s are 0. And let $X^n$ be distributed such that for all $1 \leq i \leq n$, $X_i = 1$ and all other $X_j$'s are 0 with probability $1/n$. For example, for $n = 3$, $P(100) = P(010) = P(001) = 1/3$, and $f(100) = 1$, $f(010) = 2$, and $f(001) = 3$.

It is easy to see that for computation we need to query the $n$ variables till we find one whose value is 1, hence the expected computation complexity is $(n+1)/2$. By contrast, for verification, we are told the value $j$ of $f(x^n)$ and just need to verify that $X_j = 1$, hence the expected verification complexity is 1.

Note $f$ is not symmetric, and the $X_i$'s are dependent, we have relaxed the symmetry and independence conditions as the point of the paper is to show that with them, the computation and verification complexities coincide. □

For a more precise definition, the expected query complexity of policy $\mathcal{P}$ when $f(X^n) = j$ is

$$\begin{aligned} C(\mathcal{P}|f = j) &\stackrel{\text{def}}{=} E[N(X^n)|f(X^n) = j] \\ &= \frac{\sum_{x^n : f(x^n)=j} P(x^n)N(x^n)}{P(f(X^n) = j)}, \end{aligned}$$

and the verification complexity of $f$ when $f(X^n) = j$ is the smallest expected number of bits that need to be queried to verify the value of $f$ when $f(X^n) = j$.

$$V_j(f) \stackrel{\text{def}}{=} \min_{\mathcal{P}} C(\mathcal{P}|f = j),$$

where the minimum is taken over all policies that verify the value $j$ of $f$. It is easy to see that the minimum can be equivalently taken over all policies computing $f$.

The minimum expected verification query complexity or simply verification complexity of $f$ is therefore

$$V(f) \stackrel{\text{def}}{=} \sum_j V_j(f)P(f(X^n) = j).$$

An optimal verification policy is one whose expected query complexity is $V(f)$. As with its computation counterpart, $f$ may have several optimal verification policies.

Since computation is one way of verification, or equivalently, since a *verification* policy is a set of computation policies, one for each value of $f$, we see that the verification complexity is at most the computation complexity.

***Observation** 3:* For all $f$, $V(f) \leq C(f)$. □

In Section IV we show that for all symmetric functions of independent binary inputs, $V(f) = C(f)$.

## III. Preliminary observations

Recall that symmetric functions of binary variables are determined by the input weight $w(x^n) \stackrel{\text{def}}{=} \sum_{i=1}^n x_i$. Divide the range $[0, n]$ of possible weights into contiguous intervals over which the function is constant. The following observation shows that when a policy computing a function stops, the value of the function is constant in a possible interval.

***Observation** 4:* If for an input $x^n$ a policy stops after querying $n_0$ zeros and $n_1$ ones, then $w(x^n)$ can take any value in the contiguous interval $[n_1, n - n_0]$. Furthermore, if the policy computes $f$, then $f(x^n)$ is constant for all $x^n$ with weight in this interval.

*Proof:* After querying $n_0$ 0's and $n_1$ 1's, $\sum_i x_i$ can take any value in the range $[n_1, n - n_0]$, depending on the

values of the unknown inputs. Since when the policy stops, the value of the function is determined, regardless of the unknown inputs, $f(x^n)$ must be the same for all inputs with $w(x^n) \in [n_1, n - n_0]$. □

The *interval indicator function* of $f$ is the function $g : [0, n] \to [1, n + 1]$, defined by $g(0) \stackrel{\text{def}}{=} 1$ and

$$g(i+1) - g(i) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } f(i+1) = f(i), \\ 1 & \text{if } f(i+1) \neq f(i). \end{cases}$$

$g(x)$ indicates which $f$-interval $x$ belongs to. We divide intervals into two types. An interval is *large* if its length is at least $\frac{n+1}{2}$ and it is *small* otherwise.

Next observe that $f$ and $g$ have the same computation complexity.

***Observation** 5:* For every symmetric function $f$ of independent binary variables,

$$C(f) = C(g).$$

*Proof:* For any function $f$, the value of $g$ determines the value of $f$, hence any policy computing $g$ also computes $f$, and $C(f) \leq C(g)$. Conversely, while the value of $f$ may not determine the value of $g$, e.g., for the parity function, by Observation 4, when the value of $f$ is determined, the weight $x^n$ lies in a known interval over which $f$ is constant, and hence $g$ is determined as well, and $C(g) \leq C(f)$. □

Following observation compares the verification complexities of $f$ and $g$.

***Observation** 6:* For any symmetric function $f$ of independent binary variables,

$$V(g) \leq V(f).$$

*Proof:* Recall that verification policy for $f$ is a collection of computation policies one for each value of $f(X^n)$. Since $g$ determines $f$, the number of computation policies in $g$ is larger, and hence a verification policy for $f$ is also a verification policy for $g$. □

Combining the above results, we obtain

***Corollary** 7:* For all symmetric functions $f$ of independent binary random variables,

$$V(g) = V(f) = C(f) = C(g).$$

*Proof:* From Observations 3, 5, and 6,

$$V(g) \leq V(f) \leq C(f) = C(g).$$

Our main result, Theorem 14, shows that for all symmetric functions $f$ of independent binary random variables, $V(g) = C(g)$, and the corollary follows. □

In general, verification complexity appears to be easier to determine than computation complexity, and verification complexity of the interval indicator functions seems easier to determine than the verification complexity of symmetric functions. Hence in the rest of the paper we consider only the verification complexity of interval indicator functions.

## IV. Equality of Verification and Computation Query Complexity for Symmetric Functions of Independent Binary Inputs

We now prove the equality of verification and computation complexities of symmetric functions of independent binary inputs. Based on the size of the intervals, we find the inputs which can be queried first. We then show that no matter what the value of the function is, there are some inputs that can be queried first in an optimal verification policy.

To achieve this goal, in Lemma 12 we consider optimal verification policy when the weight of the inputs belongs to large interval and we find some inputs that one of them can be queried first in an optimal verification policy. In compare, when the weight of the inputs belongs to small interval, Lemma 13 finds some inputs that any of them can be queried first in an optimal verification policy. Combination of these two lemmas help us to find an input that can be queried first in an optimal verification policy for all values of function. Lemmas 9 and 10 are the main parts of the proof of Lemmas 13 and 12 which will be stated later. We begin with an observation.

Let $\mathcal{I}_j \stackrel{\text{def}}{=} \{x | g(x) = j\}$ be the $j^{\text{th}}$ interval of the function $g$. Then $\mathcal{I}_j$ can be written as $[s_j, e_j]$ where $s_j$ and $e_j$ are the interval's start and end points. Define,

$$g_j(x) = \begin{cases} 1 & \text{if } x < s_j, \\ 2 & \text{if } s_j \leq x \leq e_j, \\ 3 & \text{if } x > e_j. \end{cases}$$

The following observation shows equality of verification of $g_j(X^n) = j$ and verification of $g(X^n) = j$.

***Observation** 8:* A policy verifies $g(X^n) = j$ iff it verifies $g_j(X^n) = 2$.

Every policy starts by querying a fixed input $X_i$ and the next query is a function of the value of $X_i$. An optimal policy is called second-input-fixed if the second input queried is some fixed $X_j$ independent of the value of $X_i$. Suppose there is a function for which there exists a second-input-fixed policy, that queries some two inputs $X_i$ and $X_j$. Using the fact that the $p_i$'s are sorted and the function is symmetric, we show in the following lemma that for any index $k$ between $i$ and $j$ there exists an optimal policy that queries $X_k$ first.

***Lemma** 9:* Let $\mathcal{P}_{opt}$ be a second-input-fixed optimal policy that queries $X_i$ and $X_j$ as the first two inputs. Then for any $k$ between $i$ and $j$ (inclusive), there exists an optimal policy that queries $X_k$ first.

*Proof:* Figure 1 describes $\mathcal{P}_{opt}$ to compute the value of $g$ or to verify a specific value of $g$. Let $\mathcal{P}_{opt,1}, \mathcal{P}_{opt,2}$, and $\mathcal{P}_{opt,3}$ be policies followed after querying of $X_i$ and $X_j$, depending on the values observed. Since the function is symmetric, in the case when $X_i + X_j = 1$, the same policy works.

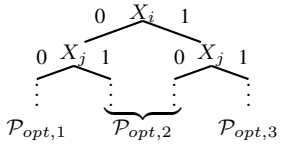

Figure 1

Without loss of generality we assume that $i < j$. For any $k \in [i, j]$, consider four policies that query $X_k$ first. They are shown in Figure 2.

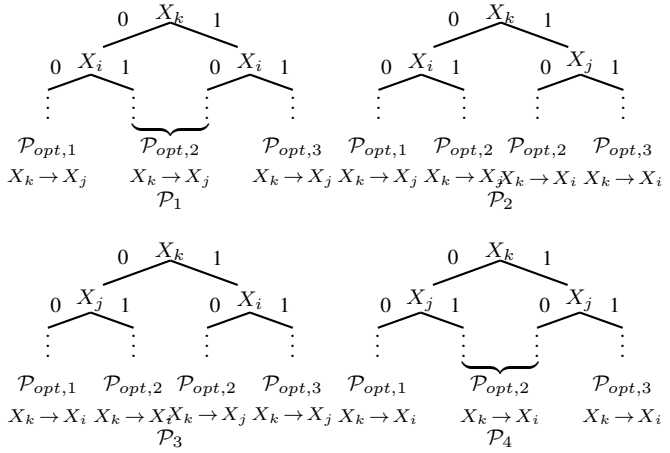


Figure 2

We briefly explain the four policies above. The first policy queries $X_k$ and then $X_i$. It then follows $\mathcal{P}_{opt}$ and queries $X_j$ when $\mathcal{P}_{opt}$ requires querying $X_k$. In the second policy $X_k$ is queried first, and depending on its value we query $X_i$ or $X_j$ and follow $\mathcal{P}_{opt}$. When $\mathcal{P}_{opt}$ requires querying $X_k$, we query one of $X_i$ and $X_j$ that has not been queried. The remaining two policies are similarly described.

We now describe the expected query complexities of the various policies described above and then show that at least one of the four policies defined performs as well as $\mathcal{P}_{opt}$.

Using the linearity of expectation we have

$$\begin{aligned} C(\mathcal{P}_{opt}) =& \bar{p}_i\bar{p}_j(a_{00}\bar{p}_k + b_{00}p_k) + \bar{p}_ip_j(a_{01}\bar{p}_k + b_{01}p_k) + \\ & p_i\bar{p}_j(a_{01}\bar{p}_k + b_{01}p_k) + p_ip_j(a_{11}\bar{p}_k + b_{11}p_k), \end{aligned}$$

where $a_{r,t}$ and $b_{r,t}$ are non-negative numbers depending on the structure of $\mathcal{P}_{opt}$, for $r, t \in \{0, 1\}$, independent of $p_i, p_j$ and $p_k$. The complexity of the four policies defined can be written as,

$$\begin{aligned} C(\mathcal{P}_1) =& \bar{p}_k\bar{p}_i(a_{00}\bar{p}_j + b_{00}p_j) + \bar{p}_kp_i(a_{01}\bar{p}_j + b_{01}p_j) + \\ & p_k\bar{p}_i(a_{01}\bar{p}_j + b_{01}p_j) + p_kp_i(a_{11}\bar{p}_j + b_{11}p_j), \\ C(\mathcal{P}_2) =& \bar{p}_k\bar{p}_i(a_{00}\bar{p}_j + b_{00}p_j) + \bar{p}_kp_i(a_{01}\bar{p}_j + b_{01}p_j) + \\ & p_k\bar{p}_j(a_{01}\bar{p}_i + b_{01}p_i) + p_kp_j(a_{11}\bar{p}_i + b_{11}p_i), \\ C(\mathcal{P}_3) =& \bar{p}_k\bar{p}_j(a_{00}\bar{p}_i + b_{00}p_i) + \bar{p}_kp_j(a_{01}\bar{p}_i + b_{01}p_i) + \\ & p_k\bar{p}_i(a_{01}\bar{p}_j + b_{01}p_j) + p_kp_i(a_{11}\bar{p}_j + b_{11}p_j), \\ C(\mathcal{P}_4) =& \bar{p}_k\bar{p}_j(a_{00}\bar{p}_i + b_{00}p_i) + \bar{p}_kp_j(a_{01}\bar{p}_i + b_{01}p_i) + \\ & p_k\bar{p}_j(a_{01}\bar{p}_i + b_{01}p_i) + p_kp_j(a_{11}\bar{p}_i + b_{11}p_i). \end{aligned}$$

Now we compare the complexity of the new four policies with the original optimal policy by subtracting $C(\mathcal{P}_{opt})$ out of them. After simplifying, we have

$$\begin{aligned} C(\mathcal{P}_1) - C(\mathcal{P}_{opt}) &= \bar{p}_i(p_j - p_k)(b_{00} - a_{01}) + p_i(p_j - p_k)(b_{01} - a_{11}), \\ C(\mathcal{P}_2) - C(\mathcal{P}_{opt}) &= \bar{p}_i(p_j - p_k)(b_{00} - a_{01}) + p_j(p_i - p_k)(b_{01} - a_{11}), \\ C(\mathcal{P}_3) - C(\mathcal{P}_{opt}) &= \bar{p}_j(p_i - p_k)(b_{00} - a_{01}) + p_i(p_j - p_k)(b_{01} - a_{11}), \\ C(\mathcal{P}_4) - C(\mathcal{P}_{opt}) &= \bar{p}_j(p_i - p_k)(b_{00} - a_{01}) + p_j(p_i - p_k)(b_{01} - a_{11}). \end{aligned}$$

By assumption, $p$'s are sorted in decreasing order and so $p_i \geq p_k \geq p_j$. If any of the $p_i, p_j$, and $p_k$ are equal, the lemma is trivial. So we assume that they are all different. Using these assumptions, one can see that either all of the above four quantities should be zero or at least one of them is negative. Therefore, at least one of these four policies performs as well as the optimal policy. □

Now consider an optimal policy that queries $X_i$ first and then queries $X_j$ or $X_k$ depending on whether $X_i = 0$ or $X_i = 1$. Such a policy is defined as second-input-varies. For given $X_i, X_j, X_k$ the middle index is the median of $i, j, k$. The corresponding $X$ is called the middle input.

***Lemma 10:*** For the second-input-varies policy defined above, there exists an optimal policy that queries the middle input of $\{X_i, X_j, X_k\}$ first.

*Proof:* The proof is omitted due to lack of space. □

Let $n_j^{\min}$ be the minimum number of inputs that need to be queried in order to compute $g$, when $\sum_i X_i \in \mathcal{I}_j$. The following lemma relates $n_j^{\min}$ to $g$.

***Lemma 11:*** $n_j^{\min} = n - |\mathcal{I}_j| + 1$.

*Proof:* From Observation 4, we conclude that $g(x) = j$ for $j \in [n_1, n - n_0]$, where $n_0$ is the number of queried 0's and $n_1$ is the number of queried 1's. Hence, $[n_1, n - n_0] \subset \mathcal{I}_j$. Therefore, $n - n_0 - n_1 + 1 \leq |I_j|$. Equality is achieved, when $[n_1, n - n_0] = \mathcal{I}_j$. □

Recall that we divided intervals into two typed of small and large intervals. We prove two following lemmas regarding the first queried input in an optimal policy when the input weight lies in a small or large interval.

***Lemma 12:*** Suppose $\sum_i X_i$ belongs to a large interval with size $L$. There exists an optimal policy to verify the value of $g$ that queries some $X_i$ first, $i \in [n - L + 1, L]$.

*Proof:* Let $j$ be the index of the interval such that $x \in \mathcal{I}_j$. By Lemma 8, we can consider the function $g_j(x)$ instead of $g(x)$. The proof is based on induction on $n$. Let $\mathcal{L}_{n,L} = [X_{n-L+1}, X_L]$. We use threshold functions as the base for induction. Depending on the position of the interval, we consider two cases. In the first case, we use a threshold function which has a jump at $L$ as the base for induction. The goal is to show that $\sum_i X_i \in [0, L - 1]$. In [1], it has been shown that any one of $[X_L, X_n]$ can be queried first in an optimal policy. Hence, $X_L \in \mathcal{L}_{n,L}$. In the second case, we use a threshold function with jump at $n - L + 1$ as the base. The goal is to show that $\sum_i X_i \in [n - L + 1, n]$. In [1] it has been shown that there exists an optimal verification policy, which queries $X_{n-L+1}$ first.

For induction, suppose the lemma is true for any function, $g'$ with $n' = n - 1$ inputs. Further, suppose that $g_j$ is not a

threshold function. By using Lemma 11 at least $n-L+1$ inputs need to be queried. Since it is not a threshold function, $L<n$ and at least two inputs need to be queried.

Let $\mathcal{P}_{opt}$ be an optimal policy, which queries $X_i$ first. If $X_i \in \mathcal{L}_{n,L}$ then the induction is complete. If $X_i \notin \mathcal{L}_{n,L}$, then either $i>L$ or $i<n-L+1$. We first consider the case, when $i>L$. By induction, there is an optimal policy $\mathcal{P}'_{opt}$, that chooses one of $\mathcal{L}_{n-1,L}$ as the second input. Either the policy $\mathcal{P}'_{opt}$ is second-input-fixed or second-input-varies. We consider them separately.

If the policy is second-input-fixed, then $X_j \in \mathcal{L}_{n-1,L}$. By Lemma 9, any one of $[X_j, X_i]$ can be queried first in an optimal policy and $[X_j, X_i] \cap \mathcal{L}_{n,L} \neq \emptyset$. If the policy is second-input-varies and suppose it queries $X_j$ if $X_i=0$, and $X_k$ if $X_i=1$, such that $X_j, X_k \in \mathcal{L}_{n-1,L}$. Then, by Lemma 10, there exist an optimal policy which queries the middle input of $\{X_i, X_j, X_k\}$ first. That input belongs to the set $\mathcal{L}_{n,L}$.

The proof for the case $i<n-L+1$ is very similar. □

***Lemma 13:*** Suppose $\sum_i X_i$ belongs to a small interval with size $l$, then querying any one of $[X_l, X_{n-l+1}]$ first, is optimal.

*Proof:* Proof is by induction and it is similar to the proof of Lemma 12. □

The following theorem is the main result of the paper. It states that there exists an optimal verification policy that is also an optimal computation policy for all symmetric functions of independent binary inputs.

***Theorem 14:*** $C(g)=V(g)$.

*Proof:* Let $\mathcal{A}_j$ denote the set of inputs that can be queried first in some optimal verification policy, when $\sum_i X_i \in \mathcal{I}_j$. Except $\Pi_{\frac{n+1}{2}}(x)$ for odd $n$, all other symmetric functions have at most one large interval. The above theorem is proved for these exceptions in [1]. So in the rest of the proof we assume that there is at most one large interval. Let $L$ and $M$ be the size and index of one of the longest intervals (and observe that such an interval may not be *large* by our definition). Let $l$ and $i$ denote the size and index of an interval other than $\mathcal{I}_M$. We consider the following two cases, based on the size of the longest interval.

**Case 1**: $L \geq \frac{n+1}{2}$

By Lemma 13, we have,

$l < \frac{n+1}{2} \implies [X_l X_{n-l+1}] \subset \mathcal{A}_i \implies [X_{n-L+1} X_L] \subset A_i$.

In addition, by Lemma 12, $[X_{n-L+1}, X_L] \cap \mathcal{A}_M \neq \emptyset$. As a result, $\bigcap_j A_j \neq \emptyset$, which implies that there is an input that can be queried first in all optimal policies for different values of $g$. After the first query, the similar argument can be used for the rest of the inputs. Hence, one can *track* one of the optimal verification policies without the knowledge of the function value. As a result, $C(g)=V(g)$.

**Case 2**: $L < \frac{n+1}{2}$

$\mathcal{A}_M = [X_L, X_{n-L+1}]$ and $A_M \subset A_i$. So $A_M \subset \bigcap_j A_j$. The rest of the proof is similar to that of case 1. □

From the proof of the theorem, it is clear that, when all the intervals are at most $\frac{n+1}{2}$, the set of inputs which can be queried first in an optimal verification policy is independent of the value of $g$. This result by itself can be used to find an optimal policy for several functions, whose their longest interval is small interval.

## V. Non-symmetric Functions

In the previous section we showed that all symmetric functions of independent binary variables have the same computation and verification complexity. The symmetry, independence, and binary conditions are necessary in that if any of the conditions is relaxed, there are functions whose verification complexity is strictly lower than their computation complexity. We show the result for the symmetry condition and leave the other two conditions for the full version of the paper.

***Example 15:*** Let $X_1, X_2$, and $X_3$ be independent $Ber(\frac{1}{2})$, and consider the following non-symmetric function $f(X_1, X_2, X_3)$,

| $X_1X_2X_3$ | 000 | 001 | 100 | 110 | 011 | 111 | 010 | 101 |
|---|---|---|---|---|---|---|---|---|
| $f(X^3)$ | 1 | 1 | 2 | 2 | 3 | 3 | 4 | 4 |

The computation complexity is,

$$V(f)=\sum_j V_j(f)P(f(X^n)=j)=\frac{2}{4}+\frac{2}{4}+\frac{2}{4}+\frac{3}{4}=\frac{9}{4}.$$

The first three terms correspond to function values 1, 2, and 3, each occurs with probability $1/4$. In each of these cases, the function value can be verified by querying exactly two inputs. The fourth term corresponds to function value 4. In that case, querying all the three inputs is necessary to verify the value of the function. On the other hand, it can be shown that the computation complexity is $\frac{10}{4}$. □

## References


[1] J. Acharya, A. Jafarpour, and A. Orlitsky, "Expected Query Complexity of Symmetric Boolean Functions," *Allerton Conference*, 2011, pp. 26-29.

[2] S. Arora and B. Barak, "Computational Complexity: A Modern Ap- proach," 1st ed. New York, NY, USA: Cambridge University Press, 2009.

[3] K. J. Arrow, L. Pesotchinsky, and M. Sobel, "On partitioning of a sample with binary-type questions in lieu of collecting observations," *Journal of the American Statistical Association*, vol. 76, pp. 402-409, 1981.

[4] Y. Ben-Asher and I. Newman, "Decision trees with boolean threshold queries," *Journal of Computer and System Sciences*, vol. 51, pp. -, 1995.

[5] H. Buhrman and R. de Wolf, "Complexity measures and decision tree complexity: A survey," *Theoretical Computer Science*, vol. 288, p. 2002, 1999.

[6] A. K. Dhulipala, C. Fragouli, and A. Orlitsky, "Silence-based com- munication," *IEEE Transactions on Information Theory*, vol. 56, pp. 350-366, January 2010.

[7] H. Kowshik and P. R. Kumar, "Optimal ordering of transmissions for computing boolean threshold functions," *in Proceedings of IEEE Symposium on Information Theory*, 2010, pp. 1863-1867.

[8] I. Wegener, "The complexity of Boolean functions," New York, NY, USA: John Wiley & Sons, Inc., 1987.