Forty-Ninth Annual Allerton Conference
Allerton House, UIUC, Illinois, USA
September 28 - 30, 2011

Expected Query Complexity of Symmetric Boolean Functions

Jayadev Acharya Ashkan Jafarpour Alon Orlitsky
University of California San Diego
La Jolla, CA 92093
{jacharya,ajafarpo,alon}@ucsd.edu

***Abstract*— The problem of finding optimal querying policy, for expected query complexity of symmetric boolean threshold functions was solved in [1] in the context of collocated networks. In this paper, instead of considering the optimal policy to compute the functions, we define the problem of verification of the function value. We use this idea to provide a simpler proof of the optimal querying policy for threshold functions. The method is more generic and is extended to delta and some other symmetric functions. We also provide some partial results for interval functions and finally address a question posed in [1]. Recently we have extended these results to any symmetric function of boolean inputs, which we mention at the end.**

I. Introduction

The worst-case *query* complexity or *decision-tree* complexity [2], [3], [4], [5], [6], of a multi-variate function, is the number of function inputs that must be revealed in the worst case to determine its value. For example, the worst-case query complexity of the 3-variable boolean function $f(x, y, z) = xy \vee \overline{x}z$ is 2, as the value of x determines which of y or z needs to be queried to find the value of f.

A function $f : \{0, 1\}^n \to \{0, 1\}$ is *symmetric* if the output remains the same under all permutations of the inputs, namely if it depends only on the number of 0's and 1's in the input. Several popular functions, such as *parity*, *threshold*, and *delta* functions, are symmetric. It is easy to see that the worst-case query complexity of all non-constant symmetric functions is n.

Motivated by sensor networks, Kowshik and Kumar [1] recently considered the *expected query complexity* of symmetric functions when the inputs X_i are distributed $Bernoulli(p_i)$, independent of each other. The objective is to find the optimal query order given the distributions of the inputs and the observations that minimizes the average number of inputs to be queried. They determined the optimal order of queries for boolean threshold functions. In particular, they showed that for threshold functions the query order depends only on the relative values of p_i's, not on their precise values.

In this paper we propose a method for establishing the optimal query order for more general functions.

The idea is to consider the expected *verification* query complexity of a function, which is the expected number of bits that need to be revealed to convince an observer of the value of the function. In other words, it is the expected query complexity of the function when the value of the function is known in advance. We clarify that the verification complexity is different from the *certificate* complexity [3], [4] where there is access to the values of the inputs in advance, rather than only the value of the function.

For example, consider the OR function $x_1 \vee \ldots \vee x_n$. To convince that the function value is 1, it suffices to show that some $x_i = 1$, whereas when the output is 0, one has to convince that there are no ones in the input. In other words, when the output is 0 the verification and computation query complexity are both n, since one needs to query all inputs (and make sure they are all zeros).

We use this method and determine the optimal query order for a class of symmetric boolean functions. More precisely, using this technique we obtain:

1) Simpler proof for the optimal query order of threshold functions.
2) Optimal query order for all delta functions.
3) Optimal query order for all functions that are not constant over any three consecutive input weights.

For the above functions, the query order is independent of the input probabilities p_i. We show that for interval functions, the query order depends on the p_i's. Using this we also show that for general functions, the querying order depends on p_i's.

Also, resolving in the negative a question asked in [1], we show that a simple extension of the optimal query order for single instantiation is suboptimal for block queries.

Perhaps a fundamental and interesting result we show is that for the functions mentioned above, the verification and computation query complexity are the same. We have recently extended this result to the class of all symmetric boolean functions, and show that they have the same verification and computation complexity under independent Bernoulli distributions.

The paper is organized as follows. In section II we provide formal definition of the problem of computation and verification. In III we see how the problem of verification can be related and used to get optimal computation policies. Sections IV, V provide optimal policies for threshold and delta functions. In section VI we look at general interval functions and show that there are simple functions for which the order of queries depends on the actual input probabilities. In section VII we address a question about block computation of functions from [1] and answer it in the negative. We provide the optimal query order for functions which are not constant for 3 consecutive input weights. Finally in section IX we mention a result about all symmetric functions of boolean inputs.

II. Notation and Problem Formulation

Consider a collection of n binary inputs 1, 2, ..., n, where input i independently generates Bernoulli distributed random variable X_i with $P(X_i=1)=p_i$. Without loss of generality we assume $p_1 \geq p_2 \geq \ldots \geq p_n$ throughout the paper. We denote $1-p_i$ by $\bar{p_i}$ and the vector of random variables $X_i, X_{i+1}, \ldots, X_j$ by X_i^j.

We are interested in computing symmetric boolean functions of X_1^n, the random variables generated by the inputs. As an example, consider the inputs as senators voting on a legislation which needs at least half of the votes in favor to pass. Let $X_i = 1$ if senator i votes for the legislation and 0 otherwise. We are thus interested in the random variable Y which is 1 if $\sum_i X_i \geq n/2$ and 0 otherwise. This is a symmetric function since the order of votes is immaterial once we know the number of votes for the legislation. So, the value of a symmetric boolean function can be specified by the number of 1's in the input, in other words by the sum of the inputs. Thus, any symmetric boolean function f can be completely specified as a function $g : \{0, 1, \ldots, n\} \rightarrow \{0, 1\}$, where $f(X_1, X_2, \ldots, X_n) = g(X_1 + X_2 + \ldots + X_n)$. From now on we define functions on $\mathbb{N}$ knowing that they can be easily mapped to symmetric functions on $\{0, 1\}^n$.

In this work, we study the boolean threshold, delta and interval functions, all of which are symmetric.

The threshold function $\Pi_\theta(x)$ is defined as

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

Threshold function is a general case of the example with senators and the OR function. This problem is relevant in scenarios where we want to know if there is a consensus or majority.

The delta function $\Delta_\theta(x)$ is defined as

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

Interval function $I_{\theta_1,\theta_2}(x)$ is defined as,

$$I_{\theta_1,\theta_2}(x) = \begin{cases} 1 & \text{if } \theta_1 \leq x \leq \theta_2, \\ 0 & \text{otherwise.} \end{cases}$$

Let $\mathcal{I}_j$ denote the set of all symmetric boolean functions which are not constant for any consecutive $j+1$ input weights. Note that $\mathcal{I}_j \subset \mathcal{I}_{j+1}$.

We now state the problem formally. Initially input i has a binary value X_i (for brevity we refer to input i as X_i), known to itself. An input can be queried at a unit cost. The inputs are queried sequentially. The next input to be queried is decided based on the p_i's and the values of the queried inputs up to that point. The computation is to be done with zero error, which means that the queried values can determine the function. The probability of an input X_1^n can be easily calculated from the p_i's. The expected cost or expected computation complexity is the number of queries averaged over all n-length binary inputs. The problem is to determine the query order that minimizes the expected cost to evaluate the function. We denote the optimal querying policy as $\mathcal{P}$ and the expected complexity is denoted by $L(\mathcal{P})$.

The problem of finding expected number of queries is more interesting than the worst case owing to the following observation [7].

Observation 1: For any non-constant symmetric function $f : \{0, 1\}^n \rightarrow \{0, 1\}$, the number of queries is n in the worst case.

To motivate and clarify the expected cost consider the following problem. Let $n = 2$, $p_1 = 0.8$ and $p_2 = 0.1$. Suppose the function is $\Pi_1(X_1 + X_2)$, i.e., if at least one of the two random variables is 1 then output is 1, 0 otherwise. If X_i is queried first, the expected cost is $1 \cdot p_i + 2 \cdot (1 - p_i) = 2 - p_i$. Thus, it is optimal to query X_1 first. In general for any function one can write the expected query complexity in terms of the input probabilities and optimize it. But the problem may not be trivial for general functions since the possible number of orderings is exponential.

We now define the problem of verification which we use to solve the problem of computation. For a function f, and an instantiation of X_1^n, suppose we are given the value of f. The objective is to then verify the value using queries on the inputs in the same setting as before. The problem here is to decide the order of queries that minimize the expected verification complexity. We denote the optimal verification policy with $\mathcal{P}_V$ and the expected complexity with $L(\mathcal{P}_V)$. Since function computation is one way of verification, the expected computation complexity is at least the expected verification complexity. Thus, for any f, its optimal computation and verification policies satisfy $L(\mathcal{P}) \geq L(\mathcal{P}_V)$. It is easy to come up with non-symmetric, functions with dependent inputs where strict inequality holds.

In the next section we describe how the problem of verification can provide a simpler way of coming up with the optimal query order compared to direct optimization of the expected computation complexity.

III. Proof Technique

In the last section we saw that for any f, $L(\mathcal{P}) \geq L(\mathcal{P}_V)$. Suppose f is such that the inequality holds with equality. This proves that the optimal computation complexity equals verification complexity. Since verification policies are usually easier to come up with, it might provide us with the optimal computation policy as we show for some specific functions.

For threshold and delta functions, for any instantiation of inputs, we show that if $S \subseteq [n]$ is the final set of inputs queries by $\mathcal{P}_V$, then we can come up with policy $\mathcal{P}$ which can compute the function by querying the same set S (may be in a different order). This shows that verification complexity equals computation complexity, thus proving optimality of the scheme.

In the next two subsection, for threshold and delta functions we find a policy $\mathcal{P}_V$ and use it to find a policy $\mathcal{P}$ such that $L(\mathcal{P}) = L(\mathcal{P}_V)$.

IV. THRESHOLD FUNCTION: $\Pi_\theta(x)$

To come up with the optimal policy we first find the optimal verification policy $\mathcal{P}_V$.

Consider the case when the threshold function has value 1. This can be verified only when θ 1's have been observed. So an optimal policy should minimize the expected time for observing θ 1's. It's intuitive to first query the input with highest probability i.e., p_1 and so on. The following theorem formalizes this intuition.

Theorem 1: For $\Pi_\theta = 1$, an optimal verification policy is to query in the order $X_1, X_2, \ldots$ until θ 1's are observed and for $\Pi_\theta = 0$, the order is $X_n, X_{n-1}, \ldots$ until $(n - \theta + 1)$ 0's are observed.

Proof: We prove the optimality when $\Pi_\theta = 1$. A similar argument holds when $\Pi_\theta = 0$. It suffices to prove that for an optimal policy, there is a policy with no more expected cost and where input 1 is queried first. This is because after the first query, the problem reduces to computing another threshold function over the rest of the inputs. We prove this by induction on (n, θ). If $n < \theta$ the function is trivially zero. If $n = \theta$, all inputs must be queried to verify that $\Pi_\theta = 1$. So without loss of generality, input 1 can be queried first.

For $\theta \geq 2$, suppose X_k is queried first. If $X_k = 1$, then we have to find optimal query order for $n - 1$ inputs and threshold $\theta - 1$, and if $X_k = 0$, then we have to find optimal verification policy for $n - 1$ inputs and threshold θ. By induction hypothesis, for both these cases there is an optimal hypothesis in which X_1 is queried first. This implies that there is an optimal policy for our problem in which the first two observed inputs are X_k and X_1 respectively, and since $\theta \geq 2$ at least two inputs are queried. Thus the order of X_k and X_1 is immaterial and X_1 could be queried first, followed by X_k and no further changes to the policy.

This leaves us with the case when $\theta = 1$. For $n = 1$ it is trivial to ask X_1, for $n = 2$, we proved as an example that input 1 should be queried first. We prove this case again by induction on n. Suppose the optimal policy queries $X_k \neq X_1$ first. If we observe 1 then we stop. If $X_k = 0$, then by induction hypothesis there is an optimal policy where the next query is X_1. Now consider the policy where we query X_1 and then X_k if needed, and from the third step onwards the policy follows the decision of the optimal policy. When $X_k = X_1$, then the two policies query the same set of inputs, since at any stage given the set of queried inputs, the next input depends only on the number of 1's observed. Now the remaining cases are $(X_k, X_1) = (0, 1)$ or $(1, 0)$. These events happen with probability $p_k(1 - p_1)$ and $p_1(1 - p_k)$. Since $p_1 \geq p_k$ it is easy to see that the new policy's expected complexity is less than or equal to the expected complexity of the optimal policy. ∎

Given this theorem, we consider the following policy $\mathcal{P}$ and show that it is optimal. In the first round we query X_θ. Suppose up to round $i-1$, i_{min} and i_{max} are the smallest and largest inputs which have been queried. If at round $i - 1$ the value observed is 1 then the next input queried is $i_{min} - 1$, else it is $i_{max} + 1$. This process continues until θ 1's or $n - \theta + 1$ 0's are observed.

Theorem 2: $\mathcal{P}$ is optimal for Π_θ.

Proof: It is clear that at each stage the set of queried random variables is of the form X_i^j for some i and j. We show that for any instantiation of X_1^n, the final set of queried inputs is the same as those observed for the optimal verification of threshold function. Consider the case when $\Pi_\theta = 1$. We show that at the termination of the policy, input 1 has been queried, and if k is the largest input index queried, then $X_k = 1$. We start with X_θ and for every 1 we observe, an input with smaller index than all queried inputs up to that time is queried. Since $\Pi_\theta = 1$, when $(\theta - 1)$th 1 is queried, X_1 is queried. Now from the policy it is clear that we go to a smaller index for every value of 1 queried, and thus the value queried from the largest index is 1. This shows that when the policy terminates, the set of queried inputs is X_1^k where k is such that $X_k = 1$ and $\sum_{i=1}^{k} X_i = \theta$. The case of $\Pi_\theta = 0$ follows similarly. ∎

V. DELTA FUNCTIONS

For delta functions, we define a similar verification problem. Since we are interested in zero error computation, any correct policy can not only tell us the value of the delta function but also whether $\sum_{i=1}^{n} X_i < \theta$, $\sum_{i=1}^{n} X_i = \theta$ or $\sum_{i=1}^{n} X_i > \theta$. This can be seen easily by invoking Observation 1. We consider the problem of verification for these cases separately. Using the results obtained for threshold functions, we obtain the following optimal verification policy.

$\sum_{i=1}^{n} X_i = \theta$: In this case any order is optimal since all inputs must be queried to verify that the sum is exactly θ.

$\sum_{i=1}^{n} X_i < \theta$: This problem is identical to the threshold problem with θ as the threshold value. We know from the previous section that an optimal verification policy is to query X_n, X_{n-1}, ... until $(n - \theta + 1)$ 0's are observed.

$\sum_{i=1}^{n} X_i > \theta$: An optimal verification policy for this case is to query X_1, X_2, ... until $(\theta + 1)$ 1's are queried.

Now suppose the function to be computed is Π_θ, instead of Δ_θ. We found an optimal query policy $\mathcal{P}$ for Π_θ. We claim that the same policy with slightly different stopping criterion is optimal for the delta function. The new stopping criteria is when $(\theta + 1)$ 1's or $(n - \theta + 1)$ 0's are observed, or all inputs have been queried, whichever occurs first. For simplicity we also call this policy $\mathcal{P}$.

Theorem 3: $\mathcal{P}$ is optimal for Δ_θ.

Proof: The proof is along the lines of the argument for threshold functions. We consider the final set of queried inputs and show that it is identical to the set of queried inputs for the verification in all the three cases above. ∎

VI. INTERVAL FUNCTIONS

For threshold and delta functions we observe that the query order depends on the relative values of the probabilities and not on their exact values. In this section, we provide a simple function, where the query order depends on the exact values of p_i's. Furthermore we show a way for generalizing the example to a large n.

The example considers the case when $n = 4$, and the interval function $I_{1,3}$. This function value is 0 whenever all inputs are all zeros or all ones, else its value is 1.

Suppose X_k is queried first. If $X_k = 0$, the problem reduces to finding an optimal policy for Π_1 over the 3 remaining inputs, and if $X_k = 1$, then we have to find an optimal policy for Π_3 over the 3 remaining inputs. We know an optimal policy for these problems from section IV.

For each value of k we can calculate the expected query complexity of the best policy given that X_k is queried first. The expressions for these values are given below.

$$L(\mathcal{P}|k=1) = 1 + p_1(p_4 + p_3p_4) + \bar{p_1}(\bar{p_2} + \bar{p_2}\bar{p_3}),$$
$$L(\mathcal{P}|k=2) = 1 + p_2(p_4 + p_3p_4) + \bar{p_2}(\bar{p_1} + \bar{p_1}\bar{p_3}),$$
$$L(\mathcal{P}|k=3) = 1 + p_3(p_4 + p_2p_4) + \bar{p_3}(\bar{p_1} + \bar{p_1}\bar{p_2}),$$
$$L(\mathcal{P}|k=4) = 1 + p_4(p_3 + p_2p_3) + \bar{p_4}(\bar{p_1} + \bar{p_1}\bar{p_2}).$$

It is easy to check that for $p_4 \geq \bar{p_1}$, $L(\mathcal{P}|k=3)$ is the smallest among the four values above, and for $p_4 \leq \bar{p_1}$ $L(\mathcal{P}|k=2)$ is smallest. This shows that the first input to be queried depends on the values and not only on the order.

We can easily extend this example for larger n, by adding inputs with $p_i = 1$ or $p_i = 0$.

VII. Observations on Block Computation

Consider the problem where each input is a k-length vector. The question of optimal strategy of querying can now be asked in this setting where each input is a block of binary random variables. In [1] it was asked whether the following is an optimal strategy for block computation of threshold functions. The θth input is queried using a Huffman coding. At the next stage the inputs $\theta + 1$ and $\theta - 1$ are queried for those inputs where the first value queried was a 0 or 1 respectively. This recursive policy was a proposed optimal policy for all n and k. We come up with a simple example where there is a better strategy. The idea is to observe that in this policy there could be inputs which are queried multiple times. If the number of times an input is queried could be reduced we can hope to reduce the expected cost.

For example, consider $n = 3$ and $k = 2$, i.e., each input has 2 values. Let $p_1 = 0.8-\epsilon$, $p_2 = 0.8$ and $p_3 = 0.8+\epsilon$. Let $\theta = 2$. For $\epsilon \to 0$ consider the following policy. Query inputs 1 and 2 using Huffman code. The probabilities associated with each inputs is $\{0.64, 0.16, 0.16, 0.04\}$ and the Huffman codewords have lengths $\{1, 2, 3, 3\}$. The expected length is 1.56 per input and thus 3.12 for querying two inputs . Now with probability 0.4624, the observed values are identical, and the third input need not be queried. With probability 0.1024 the third input needs be queried, using 1.56 bits. With probability 0.4352 the final input is queried for only one input, thus taking 1 bit. The total expected cost is thus $2 \times 1.56 + .1024 \times 1.56 + .4352 = 3.71$.

For the policy conjectured in [1], a similar computation shows that the query complexity is 3.88.

VIII. Optimal Ordering for $\mathcal{I}_2$

Theorem 4: For any function in $\mathcal{I}_2$ the optimal querying policy is to first query all the inputs X_2^{n-1}. Let $\sum_{i=2}^{n-1} X_i = s$. If $f(s) = f(s+1)$ then the next input queried is X_n, and if needed X_1 is queried. If $f(s) \neq f(s+1)$ and $f(s+1) = f(s+2)$ then X_1 is queried and then X_n if needed. If none of these are true then all nodes are queried.

Proof: Consider the optimal verification policy given the value of $\sum_{i=1}^{n} X_i$. Thus we know the value of the function and also the exact number of zeros and ones in the input.

We can once again do an induction on the number of inputs and prove this result. The argument is similar to the previous proof and is omitted here. ■

IX. Recent Results

We conclude the paper with a more general result (without proof), which states that for all symmetric functions of independent boolean inputs the computation and verification complexities are same. We also note that all the conditions (e.g., independence, boolean, symmetric) are necessary for the result to hold.

Theorem 5: For any symmetric function of independent boolean inputs,

$$\mathcal{L}(\mathcal{P}) = \mathcal{L}(\mathcal{P}_V).$$

References

[1] 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.

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

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

[4] S. Arora and B. Barak, *Computational Complexity: A Modern Approach*, 1st ed. New York, NY, USA: Cambridge University Press, 2009.

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

[6] 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.

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