Evaluation of DNF Formulas

Sarah R. Allen
Carnegie Mellon University
Computer Science Department
Pittsburgh, PA, USA

Lisa Hellerstein and **Devorah Kletenik**
Polytechnic Institute of NYU
Dept. of Computer Science and Engineering
Brooklyn, NY, USA

Tonguç Ünlüyurt
Sabanci University
Fac. of Engineering and Natural Sciences
Istanbul, Turkey

Abstract

We consider the Stochastic Boolean Function Evaluation (SBFE) problem for classes of DNF formulas. The SBFE problem for DNF formulas is a "sequential testing" problem where we need to determine the value of DNF formula on an initially unknown input $x = (x_1, ..., x_n)$, when there is a cost c_i associated with obtaining the value of x_i, each x_i is equal to 1 with known probability p_i, and the x_i are independent. The goal is to minimize expected cost. The SBFE problem is inapproximable for general DNF formulas. We give approximate and exact algorithms for two subclasses of DNF formulas: monotone k-DNF and monotone k-term DNF. We also prove a lower bound result for evaluation of monotone CDNF formulas.

1 Introduction

Stochastic Boolean Function Evaluation (SBFE) is the problem of determining the value of a given Boolean function f on an unknown input x, when each bit x_i of x can only be determined by paying a given associated cost c_i. Further, x is drawn from a given product distribution: for each x_i, $Prob[x_i = 1] = p_i$, and the bits are independent. The goal is to minimize the expected cost of evaluation. This problem has been studied in the Operations Research literature, where it is known as "sequential testing" of Boolean functions (cf. (Ünlüyurt 2004)). It has been studied in learning theory in the context of learning with attribute costs (Kaplan, Kushilevitz, and Mansour 2005).

In this paper, we study the complexity of the SBFE problem for classes of DNF formulas. We consider both exact and approximate versions of the problem for subclasses of DNF, for arbitrary costs and product distributions, and for unit costs and/or the uniform distribution. Because of the NP-hardness of satisfiability, the general SBFE problem is easily shown to be NP-hard for arbitrary DNF formulas (Greiner et al. 2006).

We consider the SBFE problem for monotone k-DNF and k-term DNF formulas. We use a simple reduction to show that the SBFE problem for k-DNF is NP-hard, even for $k = 2$. We present an algorithm for evaluating monotone k-DNF that achieves a solution that is within a factor of $\frac{4}{\rho^k}$ of optimal, where ρ is either the minimum p_i value, or the minimum $1 - p_i$ value, whichever is smaller. We present an algorithm for evaluating monotone k-term DNF with an approximation factor of $\max\{2k, \frac{2}{\rho}(1 + \ln k)\}$. We also prove that the SBFE problem for monotone k-term DNF can be solved exactly in polynomial time for constant k.

Previously, Kaplan et al. gave an approximation algorithm solving the SBFE problem for CDNF formulas (and decision trees) for the special case of unit costs, the uniform distribution, and monotone CDNF formulas (Kaplan, Kushilevitz, and Mansour 2005). CDNF formulas are formulas consisting of a DNF formula together with an equivalent CNF formula, so the size of the input depends both on the size of the CNF and the size of the DNF. Having both formulas makes the evaluation problem easier. They showed that their algorithm achieves a solution whose cost is within an $O(\log kd)$ factor of the expected certificate cost, where k is the number of terms of the DNF, and

d is the number of clauses in the CNF. The expected certificate cost is a lower bound on the cost of the optimal solution. Deshpande et al. subsequently gave an algorithm solving the unrestricted SBFE problem for CDNF formulas, whose solution is within a factor of $O(\log kd)$ of optimal, for arbitrary costs, arbitrary probabilities, and without the monotonicity assumption (Deshpande, Hellerstein, and Kletenik 2013). Thus the Deshpande et al. result solves a more general problem than that of Kaplan et al., but their approximation bound is weaker because it is not in terms of expected certificate cost.

The Kaplan et al. algorithm uses a round robin technique that alternates between two processes, one of which attempts to achieve a 0-certificate and one which attempts to achieve a 1-certificate. The technique requires unit costs. We modify the technique to handle arbitrary costs with no change in the approximation factor. The algorithm can also be trivially extended to remove the uniform distribution restriction, changing the approximation bound to $O(\frac{1}{\rho}\log kd)$.

We do not know how to remove the assumption of Kaplan et al. that the CDNF formula is monotone while still achieving an approximation factor that is within $O(\log kd)$ of the expected certificate cost. We do show, however, that this approximation factor is close to optimal, even for the special case they considered. We prove that, with respect to the expected certificate cost, the approximation factor must be $\Omega((\log kd)^{\epsilon})$, for any constant ϵ where $0 < \epsilon < 1$.

This proof also implies that the (optimal) average depth of a decision tree computing a Boolean function can be exponentially larger than the average certificate size for that function (i.e., the average of the minimum-size certificates for all 2^n assignments). In contrast, the depth complexity of a decision tree for a function, (a worst-case measure) is at most quadratic in its certificate complexity (cf. (Buhrman and Wolf 1999)).

The proofs of the results are omitted in this extended abstract. The interested reader is referred to (Allen et al. 2013) for the proofs.

2 Stochastic Boolean Function Evaluation

The formal definition of the Stochastic Boolean Function Evaluation (SBFE) problem is as follows. The input is a representation of a Boolean function $f(x_1, \ldots, x_n)$ from a fixed class of representations C, a probability vector $p = (p_1, \ldots, p_n)$, where $0 < p_i < 1$, and a real-valued cost vector $(c_1, \ldots, c_n)$, where $c_i \geq 0$. An algorithm for this problem must compute and output the value of f on an $x \in \{0,1\}^n$, drawn randomly from the product distribution D_p, i.e., the distribution where $p_i = \textit{Prob}[x_i = 1]$ with $p_i + q_i = 1$ and the x_i are independent. However, the algorithm is not given direct access to x. Instead, it can discover the value of any x_i only by "testing" it, at a cost of c_i. The algorithm must perform the tests sequentially, each time choosing the next test to perform. The algorithm can be adaptive, so the choice of the next test can depend on the outcomes of the previous tests. The expected cost of the algorithm is the cost it incurs on a random x from D_p. (Note that since each p_i is strictly between 0 and 1, the algorithm must continue doing tests until it has obtained a 0-certificate or 1-certificate for the function.) The algorithm is optimal if it has the minimum possible expected cost with respect to D_p. An interesting special case is when we have unit-costs and uniform distribution where $c_i = 1$ and $p_i = 0.5$ for all $i = 1, ..., n$.

We consider the running time of the algorithm to be the (worst-case) time it takes to determine the single next variable to be tested, or to compute the value of $f(x)$ after the last test result is received. The algorithm corresponds to a Boolean decision tree (testing strategy) computing f, indicating the adaptive sequence of tests.

SBFE problems arise in many different application areas. For example, in medical diagnosis, the x_i might correspond to medical tests performed on a given patient, where $f(x) = 1$ if the patient should be diagnosed as having a particular disease. In query optimization in databases, f could correspond to a Boolean query, on predicates corresponding to $x_1, \ldots, x_n$, that has to be evaluated for every tuple in the database in order to find tuples satisfying the query (Ibaraki and Kameda 1984; Krishnamurthy, Boral, and Zaniolo 1986; Deshpande and Hellerstein 2008; Srivastava et al. 2006).

There are polynomial-time algorithms solving the SBFE problem exactly for a small number of classes of Boolean formulas, including read-once DNF formulas and k-of-n formulas (see (Ünlüyurt

2004) for a survey of exact algorithms). There is a naive approximation algorithm for evaluating any function under any distribution that achieves an approximation factor of n: Simply test the variables in increasing order of their costs. This follows easily from the fact that the cost incurred by the naive algorithm in evaluating function f on an input x is at most n times the cost of the min-cost certificate for f, contained in x (cf. (Kaplan, Kushilevitz, and Mansour 2005)).

Deshpande et al. explored a generic approach to developing approximation algorithms for SBFE problems, called the Q-value approach. It involves reducing the problem to an instance of Stochastic Submodular Set Cover and then solving it using the Adaptive Greedy algorithm of Golovin and Krause (Golovin and Krause 2011). They proved that the Q-value approach does not yield a sublinear approximation bound for evaluating k-DNF formulas, even for $k = 2$. They also developed a new algorithm for solving Stochastic Submodular Set Cover, called Adaptive Dual Greedy, and used it to obtain a 3-approximation algorithm solving the SBFE problem for linear threshold formulas (Deshpande, Hellerstein, and Kletenik 2013).

Table 1 summarizes work on the SBFE problem for classes of DNF formulas, and for monotone versions of those classes. The table includes both previous results and the results in this paper.

3 Preliminaries

A *literal* is a variable or its negation. A *term* is a possibly empty conjunction ($\wedge$) of literals. If the term is empty, all assignments satisfy it. A clause is a possibly empty disjunction ($\vee$) of literals. If the clause is empty, no assignments satisfy it. The *size* of a term or clause is the number of literals in it.

A *DNF* (disjunctive normal form) formula is either the constant 0, the constant 1, or a formula of the form $t_1 \vee \cdots \vee t_k$, where $k \geq 1$ and each t_i is a term. Likewise, a *CNF* (conjunctive normal form) formula is either the constant 0, the constant 1, or a formula of the form $c_1 \wedge \cdots \wedge c_k$, where each c_i is a clause.

A k-term DNF is a DNF formula consisting of at most k terms. A k-DNF is a DNF formula where each term has size at most k. The *size* of a DNF (CNF) formula is the number of its terms (clauses); if it is the constant 0 or 1, its size is 1. A DNF formula is *monotone* if it contains no negations. A *read-once* DNF formula is a DNF formula where each variable appears at most once.

Given a Boolean function $f : \{0,1\}^n \rightarrow \{0,1\}$, a partial assignment $b \in \{0,1,*\}^n$ is a *0-certificate (1-certificate)* of f if $f(a) = 0$ ($f(a) = 1$) for all a such that $a_i = b_i$ for all $b_i \neq *$. It is a certificate for f if it is either a 0-certificate or a 1-certificate. Given a cost vector $c = (c_1, \ldots, c_n)$, the cost of a certificate b is $\sum_{j:b_j \neq *} c_j$. We say that input x *contains* certificate b if $x_i = b_i$ for all i with $x_i \neq *$. The *variables in a certificate* b are the x_i such that $b_i \neq *$. If S is a superset of the variables in b, then we say that S *contains* b.

The expected certificate cost of a function f, with respect to cost vector c and probability vector p, is $E_f[\mathit{CERT}]$, where $E_f[\mathit{CERT}] = \sum_x \mathit{CERT}_f(x)\mathit{Prob}[x]$. The expectation is with respect to x drawn from product distribution D_p, and $\mathit{CERT}_f(x)$ is the minimum cost of a certificate b of f contained in x.

Given a Boolean function f, let $E_f[\mathit{OPT}]$ denote the minimum expected cost of any algorithm solving the SBFE for f.

The *set covering problem* is as follows: Given a ground set $A = \{e_1, \ldots, e_m\}$ of elements, a set $\mathcal{S} = \{S_1, \ldots, S_n\}$ of subsets of A, and a positive integer k, does there exist $\mathcal{S}' \subseteq \mathcal{S}$ such that $\bigcup_{S_i \in \mathcal{S}'} = \mathcal{S}$ and $|\mathcal{S}'| \leq k$? Each set $S_i \in \mathcal{S}$ is said to *cover* the elements it contains. Thus the set covering problem asks whether A has a "cover" of size at most k.

4 Hardness of the SBFE problem for monotone DNF

Before presenting approximation algorithms solving the SBFE problem for classes of monotone DNF, we begin by discussing the hardness of the exact problem.

Greiner et al. (Greiner et al. 2006) showed that the SBFE problem for CNF formulas is NP-hard, as follows. If a CNF formula is unsatisfiable, then no tests are necessary to determine its value on an assignment x. If there were a polynomial-time algorithm solving the SBFE problem for CNF for-

[1]This follows from the fact that any DNF formula with at most k terms can be expressed as a CNF formula with at most n^k clauses.(Deshpande, Hellerstein, and Kletenik 2013)

Table 1: Complexity of the SBFE Problem for DNF Formulas

DNF formula	general case	monotone case
read-once DNF	• $O(n \ln n)$-time algorithm (Kaplan, Kushilevitz, and Mansour 2005; Greiner et al. 2006; Boros and Ünlüyurt 2000)	• $O(n \ln n)$-time algorithm (Kaplan, Kushilevitz, and Mansour 2005; Greiner et al. 2006; Boros and Ünlüyurt 2000)
k-DNF	• inapproximable even under *ud* (§ 4)	• NP-hard, even with *uc* (§ 4) • poly-time $(\frac{4}{\rho k})$-approx. algorithm (§ 5)
k-term DNF	• poly-time $O(k \log n)$-approx. [1]	• $O(n^{2^k})$-time algorithm for general case (§ 6) • $O(2^{2^k})$-time algorithm for *uc/ud* case (§ 6) • poly-time $\max\{2k,\ \frac{2}{\rho}(1 + \ln k)\}$-approx. (§ 5)
CDNF	• poly-time $O(\log(kd))$-approx. (wrt $E_f[OPT]$) (Deshpande, Hellerstein, and Kletenik 2013)	• No known poly-time exact algorithm or NP-hardness proof • poly-time $O(\log(kd))$-approx. for *uc* and *ud* (Kaplan, Kushilevitz, and Mansour 2005) • poly-time $O(\log(kd))$-approx. (wrt $E_f[OPT]$)(Deshpande, Hellerstein, and Kletenik 2013)
general DNF	• inapproximable even under *ud* (§ 4)	• NP-hard, even with *uc* (§ 4) • inapproximable within a factor of $c \ln n$ for a constant c (§ 4)

The abbreviations *uc* and *ud* are used to refer to unit costs and uniform distribution, respectively. k refers to the number of terms in the DNF, d refers to the number of clauses in the CNF. ρ is the minimum value of any p_i or $1 - p_i$. Citations of results from this paper are enclosed in parentheses and include the section number. All approximation factors are with respect to $E_f[CERT]$, the expected certificate cost, except for the CDNF bound of (Deshpande, Hellerstein, and Kletenik 2013). That bound is with respect to $E_f[OPT]$, the expected cost of the optimal strategy, which is lower bounded by $E_f[CERT]$.

mulas, we could use it to solve SAT: given CNF Formula ϕ, we could run the SBFE algorithm on ϕ (with arbitrary p and c), and just observe whether the algorithm begins by choosing a variable to test, or whether it immediately outputs 0 as the value of the formula. Thus the SBFE problem on CNF formulas is NP-hard, and by duality, the same is true for DNF formulas.

Moreover, if $\mathsf{P} \neq \mathsf{NP}$, we cannot approximate the SBFE problem for DNF within any factor $\rho > 1$. If a ρ-approximation algorithm existed, then on a tautological DNF ϕ, the algorithm would have to immediately output 1 as the value of ϕ, because $\rho \times 0 = 0$. On non-tautological ϕ, the algorithm would instead have to specify a variable to test.

The SBFE problem for DNF is still NP-hard even when the DNF is monotone. To show this, we use an approach used by Cox (Cox, Qiu, and Kuehner 1989) in proving NP-hardness of linear threshold evaluation. Intuitively, in an instance of SBFE with unit costs if the probabilities p_i are very close to 0 (or 1), then the expected cost of evaluation is dominated by the cost of evaluating the given function f on a specific input x^*. That cost is minimized by testing only the variables in a minimum-cost certificate for f on x^*. The idea, then, is to show hardness of the SBFE problem for a class of formulas C by reducing an NP-hard problem to the problem of finding, given $f \in C$ and a particular input x^*, a smallest size certificate of f contained in x^*. Cox reduced from Knapsack, and here we reduce from Vertex-Cover.

Theorem 1 *If $\mathsf{P} \neq \mathsf{NP}$, there is no polynomial time algorithm solving the SBFE problem for monotone DNF. This holds even with unit costs, and even for k-DNF where $k \geq 2$. Also, if $\mathsf{P} \neq \mathsf{NP}$, the SBFE problem for monotone DNF, even with unit costs, cannot be approximated to within a factor of less than $c \ln n$, for some constant c.*

Given the difficulty of exactly solving the SBFE problem for monotone DNF formulas, we now consider approximation algorithms.

5 Approximation algorithms for the evaluation of monotone k-DNF and k-term DNF

In this section, we will present a polynomial time algorithm for evaluating monotone k-DNF formulas. To evaluate f we will alternate between two algorithms, Alg0 and Alg1, each of which performs tests on the variables. Alg0 tries to find a min-cost 0-certificate for f, and Alg1 tries to find a min-cost 1-certificate for f. As soon as one of these algorithms succeeds in finding a certificate, we know the value of $f(x)$ and can output it.

This basic approach was used previously by Kaplan et al. (Kaplan, Kushilevitz, and Mansour 2005) in their algorithm for evaluating monotone CDNF formulas in the unit cost, uniform distribution case. They used a standard greedy set-cover algorithm for both Alg0 and Alg1, with a strict round-robin policy that alternated between doing one test of Alg0 and one test of Alg1. Our algorithm uses a dual greedy set-cover algorithm for Alg0 and a different, simple algorithm for Alg1. The strict round-robin policy used by Kaplan et al. is only suitable for unit costs, and our algorithm has to handle arbitrary costs. Our algorithm uses a modified round-robin protocol instead. We begin by presenting that protocol.

Although we will use the protocol with a particular Alg0 and Alg1, it works for any Alg0 and Alg1 that "try" to find 0-certificates and 1-certificates respectively. In the case of Alg0, this means that Alg0 will succeed in outputing a 0-certificate of f contained in x if $f(x) = 0$, and will eventually terminate and report failure otherwise. Similarly, Alg1 will output a 1-certificate contained in x if $f(x) = 1$, and will report failure otherwise.

The modified round-robin protocol works as follows. It maintains two values: K_0 and K_1, where K_0 is the cumulative cost of all tests performed so far in Alg0, and K_1 is the cumulative cost of all tests performed so far in Alg1. At each step of the protocol, each of Alg0 and Alg1 independently determines a test to be performed next and the protocol chooses one of them. (Initially, the two tests are the first tests of Alg0 and Alg1 respectively.) Let C_0 and C_1 denote the respective costs of these tests. Let x_{j_1} denote the next test of Alg1 and let x_{j_0} denote the next test of Alg0. To choose which test to perform, the protocol uses the following rule: *if $K_0 + C_0 \leq K_1 + C_1$ it performs test x_{j_0}, otherwise it performs test x_{j_1}.*

The result of the test is given to the algorithm to which it belongs, and that algorithm continues until it either (1) computes a new next test, (2) terminates successfully and outputs a certificate, or (3) terminates by reporting failure. In the first case, the protocol again chooses between the next test of Alg0 and Alg1, using the rule above. In the second, the protocol terminates because one of the algorithms has output a certificate. In the third, the protocol runs the other algorithm (the one that did not terminate) until completion, performing all of its remaining tests. That algorithm is guaranteed to output a certificate, because if x doesn't have a 0-certificate for f, it must have a 1-certificate, and vice-versa.

Note that it would be possible for the above protocol to share information between Alg0 and Alg1, so that if x_i were tested by Alg0, Alg1 would not need to retest x_i. However, to simplify the analysis, we do not have the protocol do such sharing. It can be shown that the following invariant holds at the end of each step of the protocol, provided that neither Alg0 nor Alg1 terminated in that iteration.

Lemma 1 *At the end of each step of the above modified round-robin protocol, if x_{j_1} was tested in that step, then $K_1 - c_{j_0} \leq K_0 \leq K_1$. Otherwise, if x_{j_0} was tested, then $K_0 - c_{j_1} \leq K_1 \leq K_0$ at the end of the step.*

We can now prove the following lemma:

Lemma 2 *If $f(x) = 1$, then at the end of the modified round-robin protocol, $K_1 \geq K_0$. The lemma holds true symmetrically if $f(x) = 0$.*

We now describe the particular Alg0 and Alg1 that we use in our algorithm for evaluating monotone k-DNF. We describe Alg0 first. Since f is a monotone function, the variables in any 0-certificate for f must all be set to 0. Consider an assignment $x \in \{0,1\}^n$ such that $f(x) = 0$. Let $Z = \{x_i | x_i = 0\}$. Finding a min-cost 0-certificate for f contained in x is equivalent to solving the set-cover instance where the elements to be covered are the terms $t_1, \ldots, t_m$, and for each $x_i \in Z$, there is a corresponding subset $\{t_j | x_i \in t_j\}$.

Suppose $f(x) = 0$. If Alg0 were given both Z and f as input, it could find an approximate solution to this set cover instance using Hochbaum's Dual Greedy algorithm for (weighted) set cover (Hochbaum 1982). This algorithm selects items to place in the cover, one by one, based on a certain greedy choice rule.

Alg0 is not given Z, however. It can only discover the values of variables x_i by testing them.

We get around this as follows. Alg0 begins running Hochbaum's algorithm, using the assumption that all variables are in Z. Each time that algorithm chooses a variable x_i to place in the cover, Alg0 tests the variable x_i. If the test reveals that $x_i = 0$, Alg0 continues directly to the next step of Hochbaum's algorithm. If, however, the test reveals that $x_i = 1$, it removes the x_i from consideration, and uses the greedy choice rule to choose the best variable from the remaining variables. The variables that are placed in the cover by Alg0 in this case are precisely those that would have been placed in the cover if we had run Hochbaum's algorithm with Z as input.

Hochbaum's algorithm is guaranteed to construct a cover whose total cost is within a factor of α of the optimal cover, where α is the maximum number of subsets in which any ground element appears. Since each term t_j can contain a maximum of k literals, each term can be covered at most k times. It follows that when $f(x) = 0$, Alg0 outputs a certificate that is within a factor of at most k of the minimum cost certificate of f contained in x.

If $f(x) = 1$, Alg0 will eventually test all elements without having constructed a cover, at which point it will terminate and report failure.

We now describe Alg1. Alg1 begins by evaluating the min-cost term t of f, where the cost of a term is the sum of the costs of the variables in it. (In the unit-cost case, this is the shortest term. If there is a tie for the min-cost term, Alg1 breaks the tie in some suitable way, e.g., by the lexicographic ordering of the terms.) The evaluation is done by testing the variables of t one by one in increasing cost order until a variable is found to equal 0, or all variables have been found to equal 1. (For variables x_i with equal cost, Alg1 breaks ties in some suitable way, e.g., in increasing order of their indices i.) In the latter case, Alg1 terminates and outputs the certificate setting the variables in the term to 1.

Otherwise, for each tested variable in t, Alg1 replaces all occurrences of that variable in f with its tested value. It then simplifies the formula (deleting terms with 0's and deleting 1's from terms, and optionally making the resulting formula minimal). Let f' denote the simplified formula. Because t was not satisfied, f' does not contain any satisfied terms. If f' is identically 0, x does not contain a 1-certificate and Alg1 terminates unsuccessfully. Otherwise, Alg1 proceeds recursively on the simplified formula, which contains only untested variables.

Having presented our Alg0 and Alg1, we are ready to state the main theorem of this section.

Theorem 2 *The evaluation problem for monotone k-DNF can be solved by a polynomial-time approximation algorithm computing a strategy that is within a factor of $\frac{4}{\rho^k}$ of the expected certificate cost.*

We can use techniques from the previous subsection to obtain results for the class of monotone k-term DNF formulas as well. In Section 6, we will present an exact algorithm whose running time is exponential in k. Here we present an approximation algorithm that runs in time polynomial in n, with no dependence on k.

Theorem 3 *The evaluation problem for monotone k-term DNF can be solved by a polynomial-time approximation algorithm computing a strategy that is within a factor of* $\max\{2k, \frac{2}{\rho}(1 + \ln k)\}$ *of the expected certificate cost.*

In the next section, we show that the problem of exactly evaluating monotone k-term DNF can be solved in polynomial time for constant k.

6 Evaluation of monotone k-term DNF

In this section, we provide an exact algorithm for evaluating k-term DNF formulas in polynomial time for constant k. First, we will adapt results from Greiner et al. (Greiner et al. 2006) to show some properties of optimal strategies for monotone DNF formulas. Then we will use these properties to compute an optimal strategy monotone k-term DNF formulas. Greiner et al. (Greiner et al. 2006) consider evaluating *read-once* formulas with the minimum expected cost. Each *read-once* formula can be described by a rooted and-or tree where each leaf node is labeled with a test and each internal node is labeled as either an or-node or an and-node. The simplest *read-once* formulas are the simple AND and OR functions, where the depth of the and-or tree is 1. Other *read-once* formulas can be obtained by taking the AND or OR of other *read-once* formulas over disjoint sets of variables. In the and-or tree, an internal node whose children include at least one leaf is called a *leaf-parent*, leaves with the same parent are called *leaf-siblings* (or *siblings*) and the set of all children of a *leaf-parent* is called a *sibling class*. Intuitively, the siblings have the same effect on the value of the *read-once* formula. The ratio of a variable x_i is defined to be $R(i) = \frac{p_i}{c_i}$. Further, tests x_1 and x_2 are *R-equivalent* if they are *leaf-siblings* and $R(x_1) = R(x_2)$. An *R-class* is an equivalence class with respect to the relation of being *R-equivalent*. Greiner et al. show that, for any and-or tree, (WLOG they assume that leaf-parents are OR nodes), there is an optimal strategy S that satisfies the following conditions:

(a) For any sibling tests x and y such that $R(y) > R(x)$, x is not performed before y on any root-to leaf path of S.

(b) For any *R-class* W, S is contiguous with respect to W.

We observe that by redefining siblings and sibling classes, corresponding properties hold for general monotone DNF formulas. Let us define a maximal subset of the variables that appear in exactly the same set of terms as a *sibling class* in a DNF formula. All the other definitions can easily be adapted accordingly. In this case, the ratio of a variable x_i is $R(i) = \frac{q_i}{c_i}$. For instance, all variables are siblings for an AND function, whereas no two variables are siblings in an OR function.

It is possible to adapt the proof of Theorem 20 in (Greiner et al. 2006) to apply to monotone DNF formulas. All the steps of the proof can be adapted in this context, using the new definitions of siblings and the ratio of a variable.

Theorem 4 *For any monotone DNF, there exists an optimal testing strategy S that satisfies conditions (a) and (b) stated above.*

In other words, there exists an optimal strategy such that on any path from the root to the leaf, sibling tests appear in non-decreasing order of their ratios. Further, for this strategy, sibling tests with the same ratio (*R-class*) appear one after another on any path from the root to the leaf. By a duality argument, a similar result holds for monotone CNFs by defining the ratio of a variable as $\frac{c_i}{p_i}$ and *sibling class* as a set of variables that appear in exactly the same set of clauses. For a k-term monotone DNF there are at most $2^k - 1$ sibling classes, since each sibling class corresponds to a non-empty subset of the terms of the monotone DNF formula. It is possible to use a dynamic programming based method to find an optimal strategy by exploiting these results.

Theorem 5 *The evaluation problem for monotone k-term DNF formula ϕ over a product distribution on input x and with arbitrary costs can be solved exactly in polynomial time for constant k.*

Corollary 1 *The evaluation problem for monotone k-term DNF, restricted to the uniform distribution on input x and unit costs, can be solved exactly in polynomial time for $k = O(\log \log n)$.*

7 Expected certificate cost and optimal expected evaluation cost

Some of the approximation bounds discussed in this paper are with respect to the optimal expected cost of an evaluation strategy, while others are in terms of the expected certificate cost of the function, which lower bounds the former quantity. It has been previously observed in (Deshpande, Hellerstein, and Kletenik 2013) that for arbitrary probabilities, there can be a gap of $\Omega(\log n)$ between the two measures. In what follows, we prove that even in the unit-cost, uniform distribution case, the ratio between these two can be extremely large: $\Omega(n^\epsilon)$ for any constant ϵ where $0 < \epsilon < 1$. (Note that in the unit-cost case, both measures are at most n.) We also show near-optimality of the CDNF approximation bound of Kaplan et al. and give a gap between two complexity measures related to decision trees for Boolean functions.

Theorem 6 *Let β be a constant such that $0 < \beta < 1$. Let f be a read-once DNF formula on n variables where each term is of length $\beta \log_2 n$, and every variable appears in exactly one term. Then $E_f[OPT] = \Omega(n^\beta)$, and $E_f[CERT] = O(\log n)$ for unit-cost, uniform distribution case.*

For any constant ϵ, where $0 < \epsilon < 1$, there is a constant d where $0 < \epsilon < d < 1$. For large enough n, $n^\epsilon \log n < n^d$. We thus have the following corollary.

Corollary 2 *There exists a Boolean function f such that for any constant ϵ, where $0 < \epsilon < 1$, $E_f[OPT]/E_f[CERT] = \Omega(n^\epsilon)$.*

Theorem 6 and the above corollary can also be interpreted as results on average-case analogs of depth-complexity and certificate-complexity. The depth complexity of a Boolean function f is the minimum, over all decision trees for f, of the depth of that tree. Note that the depth of the tree is the worst-case (i.e., maximum), over all 2^n assignments x to the variables of that function, of the number of tests (decisions) induced by the tree on assignment x. The certificate complexity of a Boolean function is the worst-case (i.e., maximum), over all 2^n input assignments x, of the smallest 0-certificate or 1-certificate of f that is contained in x. The average depth-complexity and average certificate-complexity of a Boolean function can be defined analogously, with worst-case replaced by average case. Thus the average depth-complexity is equal to $E_f[\mathit{OPT}]$, and the average certificate-complexity is equal to $E_f[\mathit{CERT}]$.

We can also use Theorem 6 to show near-optimality of the $O(\log kd)$ approximation bound achieved by Kaplan et al. for monotone CDNF evaluation, with respect to $E_f[\mathit{CERT}]$, the expected certificate cost under unit costs and the uniform distribution. The function computed by the formula in Theorem 6 has a CNF formula with $(\beta \log_2 n)^{n/(\beta \log_2 n)}$ clauses. Thus in this case $O(\log kd)$ is $O(\frac{n \log\log n}{\log n})$, which is $O(n)$. The strategy computed by any approximation algorithm for this problem cannot do better than the optimal strategy, so its expected cost must be at least $E_f[\mathit{OPT}]/E_f[\mathit{CERT}]$ times larger than $E_f[\mathit{OPT}]$. It follows that the approximation $O(\log kd)$ bound of Kaplan et al. has a matching lower bound of $\Omega((\log kd)^{\epsilon})$ (for $0 < \epsilon < 1$), with respect to the expected certificate cost.

We do not know, however, whether it is possible for a *polynomial-time* algorithm to achieve an approximation factor much better than $O(\log kd)$ with respect to the expected cost of the optimal strategy, $E_f[\mathit{OPT}]$. We have no non-trivial lower bound for the approximation algorithm in this case; clearly such a lower bound would have to depend on complexity theoretic assumptions.

8 Acknowledgments

Sarah R. Allen was partially supported by an NSF Graduate Research Fellowship under Grant 0946825 and by NSF grant CCF-1116594. Lisa Hellerstein was partially supported by NSF Grants 1217968 and 0917153. Devorah Kletenik was partially supported by NSF Grant 0917153. Tonguç Ünlüyurt was partially supported by TUBITAK 2219 programme. Part of this research was performed while Tonguç Ünlüyurt was visiting faculty at Polytechnic Institute of NYU and Sarah Allen was a student there.

References

Allen, S.; Hellerstein, L.; Kletenik, D.; and Ünlüyurt, T. 2013. Evaluation of DNF formulas. http://arxiv.org/abs/1310.3673.

Boros, E., and Ünlüyurt, T. 2000. *Computing Tools for Modeling, Optimization and Simulation*, volume 12 of *Operations Research/Computer Science Interfaces Series*. Springer. chapter Sequential Testing of Series-Parallel Systems of Small Depth, 39–73.

Buhrman, H., and Wolf, R. D. 1999. Complexity measures and decision tree complexity: A survey. *Theoretical Computer Science* 288:2002.

Cox, L.; Qiu, Y.; and Kuehner, W. 1989. Heuristic least-cost computation of discrete classification functions with uncertain argument values. *Annals of Operations Research* 21:1–29.

Deshpande, A., and Hellerstein, L. 2008. Flow algorithms for parallel query optimization. In *ICDE*.

Deshpande, A.; Hellerstein, L.; and Kletenik, D. 2013. Approximation algorithms for stochastic boolean function evaluation and stochastic submodular set cover. http://arxiv.org/abs/1303.0726.

Golovin, D., and Krause, A. 2011. Adaptive submodularity: Theory and applications in active learning and stochastic optimization. *JAIR*.

Greiner, R.; Hayward, R.; Jankowska, M.; and Molloy, M. 2006. Finding optimal satisficing strategies for and-or trees. *Artif. Intell.* 170(1):19–58.

Hochbaum, D. S. 1982. Approximation algorithms for the set covering and vertex cover problems. *SIAM J. Comput.* 11(3):555–556.

Ibaraki, T., and Kameda, T. 1984. On the optimal nesting order for computing n-relational joins. *ACM Trans. Database Syst.* 9(3):482–502.

Kaplan, H.; Kushilevitz, E.; and Mansour, Y. 2005. Learning with attribute costs. In *STOC*, 356–365.

Krishnamurthy, R.; Boral, H.; and Zaniolo, C. 1986. Optimization of nonrecursive queries. In *VLDB*.

Srivastava, U.; Munagala, K.; Widom, J.; and Motwani, R. 2006. Query optimization over web services. In *VLDB*.

Ünlüyurt, T. 2004. Sequential testing of complex systems: a review. *Discrete Applied Mathematics* 142(1-3):189–205.