Algorithmica
DOI 10.1007/s00453-015-0092-9

Evaluation of Monotone DNF Formulas

Sarah R. Allen[1] · **Lisa Hellerstein**[2] · **Devorah Kletenik**[3] · **Tonguç Ünlüyurt**[4]

Received: 19 May 2014 / Accepted: 5 November 2015

Abstract 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, $\mathbf{Pr}[x_i = 1] = p_i$ and the bits are independent. The goal is to minimize the expected cost of evaluation. 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.

Keywords DNF formulas · Sequential testing · Stochastic boolean function evaluation · Approximation algorithms

✉ Devorah Kletenik
kletenik@brooklyn.cuny.edu

Sarah R. Allen
srallen@cs.cmu.edu

Lisa Hellerstein
lisa.hellerstein@nyu.edu

Tonguç Ünlüyurt
tonguc@sabanciuniv.edu

1 Carnegie Mellon University, 5000 Forbes Ave, Pittsburgh, PA 15213, USA

2 NYU School of Engineering, 2 Metrotech Center, Brooklyn, NY 11201, USA

3 Brooklyn College, 2900 Bedford Avenue, Brooklyn, NY 11210, USA

4 Sabanci University, Orhanli, Tuzla, 34956 Istanbul, Turkey

Published online: 19 November 2015

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: each bit x_i is drawn independently from a Bernoulli distribution with $\mathbf{Pr}[x_i = 1] = p_i$. 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. [27]). It has also been studied in learning theory in the context of learning with attribute costs [20]. Recently, Deshpande et al. addressed the problem using tools from stochastic submodular optimization [11,12].

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 it is co-NP hard to test if a DNF formula is true on all assignments, the general SBFE problem is easily shown to be co-NP-hard for arbitrary DNF formulas [16].

We present results on the SBFE problem for monotone k-DNF and k-term DNF formulas. (A k-term DNF formula consists of at most k terms; a k-DNF formula consists of terms containing at most k literals.) 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 whose expected cost is within a factor of $4/\rho^k$ 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 $\max\left\{2k, \frac{2}{\rho}(1 + \ln k)\right\}$. 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. [20] and Deshpande et al. [11,12] provided approximation algorithms solving the SBFE problem for CDNF formulas (and decision trees). CDNF formulas are formulas consisting of a DNF formula together with its equivalent CNF formula, so the size of the input depends both on the size of the CNF and the size of the DNF. The algorithm of Kaplan et al. worked only for monotone CDNF formulas, unit costs, and the uniform distribution. The algorithm of Deshpande et al. worked for the general case. The approximation factor achieved by both algorithms is $O(\log kd)$, where k is the number of terms of the DNF and d is the number of clauses. The algorithm of Kaplan et al. used a round-robin approach. We modify their round-robin approach in our approximation algorithms to handle arbitrary costs.

While approximation factors for optimization problems are generally given with respect to the cost of the optimal solution, for SBFE problems one can also give approximation factors with respect to the *expected certificate cost*, which lower bounds the cost of the optimal solution. All of the new approximation bounds we give in this paper also hold with respect to expected certificate cost.

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 defined by p. Specifically, D_p is the distribution where $p_i = \mathbf{Pr}[x_i = 1]$ for each x_i and the x_i are independent. 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 a single test to perform next. 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 drawn from D_p. (Note that since each p_i is strictly between 0 and 1, the algorithm must continue performing tests until it has obtained either a 0-certificate or a 1-certificate for the function.) The algorithm is optimal if it has the minimum possible expected cost with respect to D_p.

We consider the running time of the algorithm on input x to be the time it takes to determine the value of $f(x)$, assuming that once each test is chosen, its result is available in constant time. The algorithm implicitly defines 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 with a particular disease. In a factory setting, the x_i might be the results of quality-control tests performed on a manufactured item, where $f(x) = 1$ if the item should be sold. 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 [10,19,21,26].

In what follows, unless otherwise noted, the term *evaluation problem* refers to the SBFE problem.

2.1 Related Work

There are polynomial-time algorithms that solve the evaluation problem exactly for a small number of classes of Boolean formulas, including read-once DNF formulas and k-out-of-n formulas (see [27] for a survey of exact algorithms). There is a naïve approximation algorithm for evaluating any function that achieves an approximation factor of n under any distribution: Simply test the variables in increasing order of their costs. Proving that the approximation factor of n holds follows easily from the fact that the cost incurred by this 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 [20].

There is a *sample* variant of the evaluation problem, where the input is a sample of size m of f (i.e, a set of m pairs $(x, f(x))$). The problem here is to build a decision tree in polynomial time such that the tree correctly computes f on all x in the sample

and minimizes the average cost of evaluation over the sample. There are $O(\log m)$-approximation algorithms solving this problem for arbitrary f [2,8,15]. Cicalese et al. recently gave an $O(\log m)$-approximation algorithm for a generalization of this problem, where a probability distribution is given on the pairs $(x, f(x))$ in the sample, and the goal is to minimize the expected evaluation cost, rather than just the average evaluation cost th [8]. That algorithm also achieves an $O(\log m)$-approximation for worst-case evaluation cost [8]. A simpler $O(\log m)$-approximation algorithm for worst-case evaluation cost was given earlier by Moshkov [23]. Moshkov and Chikalov proved a bound on average evaluation cost over a sample in terms of a combinatorial measure of the sample [22].

The *min-sum set cover* problem is equivalent to the problem of minimizing the average evaluation cost over a sample when f is a disjunction and the sample consists only of positive examples of f. There is a 4-approximation algorithm for this problem [1,13,24]. The *generalized min-sum set cover* problem is equivalent to the problem of minimizing the average cost over a sample when f is a k-out-of-n function and the sample consists only of positive examples; this problem has a 12.4-approximation algorithm [25].

Kaplan et al. [20] considered the problem of minimizing the expected cost of evaluating a Boolean function f with respect to a given *arbitrary* probability distribution, where the distribution is given by a conditional probability oracle. They gave results when f is a conjunction or a disjunction.

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 [14]. They used the Q-value approach to obtain their $O(\log kd)$-approximation algorithm for evaluating CDNF formulas. However, they also proved that the Q-value approach does not yield a sublinear approximation bound for evaluating k-DNF formulas, even for $k = 2$. They 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 [12].

Charikar et al. [4] considered the problem of developing algorithms for Boolean function evaluation that minimize *competitive ratio*. The competitive ratio of an algorithm for evaluating f is the maximum, over all inputs x, of the ratio between the cost incurred by the algorithm in determining $f(x)$, and the cost of the min-cost certificate of f contained in x. Charikar et al. also considered (cost-independent) algorithms for minimizing *extremal* competitive ratio, which is the maximum of the competitive ratio over all possible cost vectors. Significant further results on these problems were subsequently obtained by Cicalese and Laber [7] and Cicalese et al. [6]. The modified round-robin approach that we use in our approximation algorithms is similar to the Balance algorithm of Charikar et al. [4].

In this paper we consider the ratio betwen the *average* cost of an algorithm on all inputs x and the *average* cost of the min-cost certificates for these inputs. This ratio is clearly upper bounded by the competitive ratio; however, it can be significantly lower. For example, in the evaluation of an OR formula with unit costs, the competitive ratio

is n, whereas the ratio we consider is $\frac{2(2^n-1)}{2^n+n-1}$, which is <2. Because the numerator in our ratio is an average over all inputs x, we can amortize the costs incurred on different assignments. In the analysis of our algorithms, we follow an approach of Kaplan et al., charging costs incurred on one assignment to costs incurred on other assignments.

Table 1 summarizes the best results known for the SBFE problem for classes of DNF formulas and for monotone versions of those classes.

3 Preliminaries

3.1 Definitions

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 assignment satisfies it. The *size* of a term or clause is the number of literals it contains.

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 in which each term has size at most k. The *size* of a DNF (CNF) formula is the number of terms (clauses) it contains; 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 in which each variable appears at most once.

Given a Boolean function $f : \{0,1\}^n \to \{0,1\}$, 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 $a \in \{0,1\}^n$ 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 $\sum_{j:b_j \neq *} c_j$. The *variables in a certificate* b are all x_i such that $b_i \neq *$. We say that $x \in \{0,1\}^n$ *contains* certificate b if $x_i = b_i$ on the variables of b. A certificate for f on x is a certificate b of f that is contained in x. If x contains b and S is a superset of the variables in b, then we say that S *contains* b.

The *expected certificate cost* of f, with respect to cost vector $c = (c_1, \ldots, c_n)$ and distribution D on $\{0,1\}^n$, is the expected cost of the minimum cost certificate of f contained in x, when $x \sim D$.

The SET COVER 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 $\bigcup_{S_i \in \mathcal{S}'} S_i = A$ 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.

Table 1 Complexity of the SBFE problem for DNF formulas

DNF formula	Monotone or non-monotone	Monotone
Read-once DNF	$O(n \log n)$-time algorithm [3,16,20]	$O(n \log n)$-time algorithm [16,20]
k-DNF	Inapproximable even under *ud* (Sect. 4)	NP-hard, even with *uc* (Sect. 4)
		Poly-time $\left(\frac{4}{\rho^k}\right)$-approx. alg. (Sect. 5.1)
k-term DNF	Poly-time (in n^k) $O(k \log n)$-approx. alg. [12]	$O(n^{2^k})$-time alg. (Sect. 6)
		$O\left(n2^{2^k}\right)$-time alg. for *uc/ud* case (Sect. 6)
		Poly-time max $\left\{2k, \frac{2}{\rho}(1+\ln k)\right\}$-approx. (Sect. 5.2)
CDNF	Poly-time $O(\log(kd))$-approx. wrt optimal [12]	Known upper bounds same as for non-monotone case
	Poly-time $O\left(\frac{1}{\rho}\log(kd)\right)$-approx. [11]	
General DNF	Inapproximable even under *ud* [16]	NP-hard, even with *uc* (Sect. 4)
		Inapproximable within a factor of $c \ln n$ for a constant c (Sect. 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. Approximation results for non-monotone DNF also apply to monotone DNF. Except where noted, approximation factors are with respect to the expected certificate cost, which lower bounds the cost of the optimal solution. The result for k-term DNF for the non-monotone case follows from the result for CDNF formulas, using the fact that given a DNF formula with at most k terms, an equivalent CNF formula with at most n^k clauses can be computed in time polynomial in n^k

Greiner et al. [16] showed that the SBFE problem for CNF formulas is NP-hard as follows. If a CNF formula is unsatisfiable, then no testing is necessary to determine its value on an assignment x. If there were a polynomial-time algorithm solving the SBFE problem for CNF formulas, 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 simply observe whether the algorithm chooses 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, it is co-NP-hard on DNF formulas.

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

The SBFE problem for DNF is NP-hard even when the DNF is monotone. To show this, we use an argument similar to one used by Cox et al. [9] in proving NP-hardness of linear threshold function 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^*. Then the idea 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^*. We also show hardness of approximation.

For this reduction, it will be useful to demonstrate that for a particular distribution (namely one with small p_i) , the optimal decision tree over this distribution is also optimal for the specific case where x is the all-zero vector. We obtain a reduction from vertex cover by fixing the value of the input to be the all-zero assignment and then running an SBFE algorithm over the input. It is therefore necessary to demonstrate that there exists a distribution for which the optimal decision tree with respect to expected cost is also optimal on this fixed input.

Lemma 1 *Let $\mathcal{T}$ be the set of decision trees computing $f(x_1, \ldots, x_n)$. Define D_γ to be the distribution on $x_1, \ldots, x_n$ where each $q_i = 1 - p_i = \left(\frac{\gamma n + 0.5}{\gamma n + 1}\right)^{1/n}$ for some parameter $\gamma \geq 1$. Denote by $cost_T(x)$ the cost incurred by tree T on assignment x. Let $cost^* = \min_{T \in \mathcal{T}} \mathop{\mathbf{E}}_{x \sim D_\gamma}[cost_T(x)]$ and let $\mathcal{T}^* = \{T \in \mathcal{T} : \mathop{\mathbf{E}}_{x \sim D_\gamma}[cost_T(x)] = cost^*\}$. ($\mathcal{T}^*$ is the set of all decision trees computing f that are optimal with respect to D_γ.) Denote by $\mathbf{0}$ the all-zero vector. Let Z be the set of all minimum-cost certificates contained in $\mathbf{0}$ and let ζ denote the number of variables set to 0 in an element of Z. Then the following hold:*

[1] We note that Kaplan et al. actually define the problem slightly differently. They require the evaluation strategy to output a *proof* of the function value upon termination. In the case of a tautological DNF formula, they would require testing of the variables in one term of the DNF in order to output that term as a certificate.

1. *If* $\gamma = 1$, *then for every* $T^* \in \mathcal{T}^*$, *the variables tested by* T^* *on assignment* $\mathbf{0}$ *are exactly the variables set to zero in some element of* Z.
2. *For every* $T \in \mathcal{T}$, *if* $\mathop{\mathbf{E}}_{x \sim D_\gamma}[cost_T(x)] \leq \gamma\, cost^*$, *then* $cost_T(\mathbf{0}) \leq 2\gamma\zeta$.

Proof We begin by proving the first statement. Suppose for sake of contradiction that there exists some T^* in $\mathcal{T}^*$ such that the variables tested on assignment $\mathbf{0}$ do not correspond to an element of Z. Then $\text{cost}_{T^*}(\mathbf{0}) \geq \zeta + 1$ and it follows that $\text{cost}^* \geq (\zeta + 1)q^n$.

Let $T^0 \in \mathcal{T}$ be such that $\text{cost}_{T^0}(\mathbf{0}) = \zeta$. ($T^0$ tests exactly the variables in some element of Z on the all-zeros assignment). Then $\mathbf{E}[\text{cost}_{T^0}] \leq \zeta q^n + n(1 - q^n)$ and we have the following:

$$\begin{aligned}
\mathbf{E}[\text{cost}_{T^0}(x)] &\leq \zeta q^n + n(1 - q^n) \\
&= \left(\tfrac{n+0.5}{n+1}\right)\zeta + \left(\tfrac{0.5n}{n+1}\right) \\
&< \left(\tfrac{n+0.5}{n+1}\right)\zeta + \left(\tfrac{n+0.5}{n+1}\right) \\
&= (\zeta + 1)q^n \\
&\leq \text{cost}^*,
\end{aligned}$$

which contradicts the optimality of T^*.

The proof of the second statement is similar. Suppose that there exists $T \in \mathcal{T}$ such that $\mathbf{E}_{x \sim D_\gamma}[\text{cost}_T(x)] \leq \gamma\ \text{cost}^*$ and $\text{cost}_T(\mathbf{0}) > 2\gamma\zeta$. Then

$$q^n(2\gamma\zeta + 1) \leq \mathop{\mathbf{E}}_{x \sim D_\gamma}[\text{cost}_T(x)].$$

By assumption,

$$\text{cost}^* \geq \tfrac{1}{\gamma}\left(\mathop{\mathbf{E}}_{x \sim D_\gamma}[\text{cost}_T(x)]\right) \geq q^n\left(2\zeta + \tfrac{1}{\gamma}\right) \tag{1}$$

Let T' be the tree obtained by the following algorithm: the algorithm tests the variables in minimum-cost 0-certificate. If all the variables in this minimum-cost 0-certificate are 0, the algorithm outputs 0. If any of these variables is 1, then the algorithm tests the variables in any order and outputs 0 and 1, as appropriate. Then we have,

$$\text{cost}^* \leq \mathop{\mathbf{E}}_{x \sim D_\gamma}[\text{cost}_{T'}(x)] \leq q^n\zeta + (1 - q^n)n \tag{2}$$

By using (1), we have that,

$$\begin{aligned}
\text{cost}^* &\leq q^n\zeta + (1 - q^n)n \\
&= q^n\left(\zeta + \tfrac{0.5n}{\gamma n + 0.5}\right)
\end{aligned}$$

$$< q^n \left(\zeta + \frac{0.5n}{\gamma n}\right)$$
$$= \tfrac{1}{2} q^n \left(2\zeta + \frac{1}{\gamma}\right)$$

and we obtain a contradiction with the lower bound of (1).

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* $c' > 0$.

Proof Suppose there is a polynomial-time algorithm ALG for the SBFE problem for monotone 2-DNF, with unit costs and arbitrary probabilities. We show this algorithm could be used to solve the VERTEX COVER problem: Given a graph $G(V, E)$, find a minimum-size vertex cover for G, i.e., a minimum-size set of vertices $V' \subseteq V$ such that for each edge $(v_j, v_k) \in E$, $\{v_j, v_k\} \bigcap V' \neq \emptyset$.

The reduction is as follows. On graph $G(V, E)$, construct a monotone 2-DNF formula ϕ whose variables x_j correspond to the vertices $v_j \in V$, and whose terms $x_j x_k$ correspond to the edges $e = (v_j, v_k)$ in E. Consider the all 0's assignment $\mathbf{0}$. Since a 0-certificate for ϕ must set each term of ϕ to 0, any min-cost certificate for ϕ contained in $\mathbf{0}$ must also be a minimum-size vertex cover for G. Thus by the previous lemma, one can find a minimum-size vertex cover for G by using ALG to evaluate ϕ on input $\mathbf{0}$ with unit costs and the probabilities p_i given in Lemma 1, and observing which variables are tested.

We can also reduce from the SET COVER problem to the SBFE problem on DNF formulas with terms of arbitrary lengths as follows: We construct a DNF formula ϕ whose variables correspond to the subsets s_i in the SET COVER instance S. We include in ϕ a term t_j for each ground element e_j such that $t_j = \bigwedge_{i|e_j \in s_i} x_i$. Now consider the all 0's assignment $\mathbf{0}$. The size of the min-cost certificate for ϕ must also be a minimum-size cover for S.

Suppose there is a polynomial-time algorithm ALG for the SBFE problem on monotone DNF, with unit costs and arbitrary probabilities. Then we could use ALG to evaluate ϕ on input $\mathbf{0}$ with unit costs and the probabilities given in Lemma 1, observing which variables are tested. The inapproximability bound in the theorem follows from the $c \ln n$ inapproximability result for set cover [13], where $c = 2c'$. □

Given the difficulty of exactly solving the SBFE problem for montone DNF formulas, we will instead consider approximation algorithms.

5 Approximation Algorithms for the Evaluation of Monotone k-DNF and k-term DNF

5.1 Monotone k-DNF Formulas

In this section, we 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 x_i. It would be possible for the 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 assume the algorithms do not share test outcomes. 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. [20] 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 along with with a strict round-robin policy that alternated between performing 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 suitable only for unit costs, whereas our algorithm handles arbitrary costs by modifying the round-robin protocol. 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 if $f(x) = 0$, Alg0 will succeed in outputting a 0-certificate of f contained in x. Similarly, if $f(x) = 1$, Alg1 will successfully output a 1-certificate contained in x. Since x cannot contain both a 0-certificate and a 1-certificate, if Alg0 and Alg1 are both run on the same input x, only one can succeed in outputting the type of certificate it is trying to find.

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 x_{j_1} denote the next test of Alg1 and let x_{j_0} denote the next test of Alg0, and let c_{j_1} and c_{j_0} denote the costs of the next test of Alg1 and Alg0 respectively. To choose which test to perform, the protocol uses the following rule: *if* $K_0 + c_{j_0} \leq K_1 + c_{j_1}$ *it performs test* x_{j_0}; *otherwise it performs test* x_{j_1}. We present the pseudocode for this procedure in Algorithm 1.

The result of the test is combined with the results of the other tests performed by the algorithm. If these results constitute a certificate of the "correct" type (a 1-certificate for Alg1, or a 0-certificate for Alg0), then the algorithm successfully terminates and the protocol ends. If the results constitute a certificate of the "wrong" type (a 0-certificate for Alg1, or a 1-certificate for Alg0), then the algorithm has failed; the other algorithm is run to successful termination and the protocol ends. Otherwise, the algorithm has not found a certificate and it must make at least one more test. In this case, the protocol again chooses between the next test of Alg0 and Alg1 using the rule above.

We now show that the following invariant holds at the end of each step of the protocol, provided that neither Alg0 nor Alg1 failed in that iteration.

Lemma 2 *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.*

```
Initialize execution of Alg1 and Alg0
K_0 ← 0, K_1 ← 0
repeat
  if {x_i | x_i was tested by Alg1} contains a 0-certificate then
    Let Alg0 run to successful termination //Alg1 has failed, x does not have a 1-certificate
  else if {x_i | x_i was tested by Alg0} contains a 1-certificate then
    Let Alg1 run to successful termination // Alg0 has failed, x does not have a 0-certificate
  else
    x_{j_0} ← the next test of Alg0, x_{j_1} ← the next test of Alg1
    if K_0 + c_{j_0} ≤ K_1 + c_{j_1} then
      test x_{j_0} //perform the next step of Alg0
      K_0 = K_0 + c_{j_0}
    else
      test x_{j_1} //perform the next step of Alg1
      K_1 = K_1 + c_{j_1}
    end if
  end if
until Alg0 terminates successfully OR Alg1 terminates successfully
```

Algorithm 1: Modified round robin protocol

Proof The invariant clearly holds after the first step. Suppose it is true at the end of the kth step, and without loss of generality assume that x_{j_1} was tested during that step. Thus $K_1 - c_{j_0} \leq K_0 \leq K_1$ at the end of the kth step.

Consider the $(k+1)$st step. Note that x_{j_0}, the variable to be tested next by Alg0 is the same in this step as in the previous one, because in the previous step, we did not execute the next step of Alg0. There are 2 cases, depending on which if-condition is satisfied when the rule is applied in this step, $K_0 + c_{j_0} \leq K_1 + c_{j_1}$ or $K_0 + c_{j_0} > K_1 + c_{j_1}$.

Case 1 $K_0 + c_{j_0} \leq K_1 + c_{j_1}$ is satisfied.

Then x_{j_0} is tested in this step and K_0 increases by c_{j_0}. We show that $K_0 - c_{j_1} < K_1$ and $K_1 \leq K_0$ at the end of the step, which is what we need. At the start of the step, $K_0 + c_{j_0} \leq K_1 + c_{j_1}$ and at the end, K_0 is augmented by c_{j_0}, so $K_0 \leq K_1 + c_{j_1}$. Consequently, $K_0 - c_{j_1} \leq K_1$. Further, by assumption, $K_1 - c_{j_0} \leq K_0$ at the start of the step, and hence at the end, $K_1 \leq K_0$.

Case 2 $K_0 + c_{j_0} > K_1 + c_{j_1}$ is satisfied.

By assumption, therefore, $K_1 - c_{j_0} \leq K_0 \leq K_1$ at the start. Then x_{j_1} is tested in this step, and K_1 increases by c_{j_1}. We show that $K_1 - c_{j_0} \leq K_0$ and $K_0 \leq K_1$ at the end of the step. By the condition in the case, $K_0 + c_{j_0} > K_1 + c_{j_1}$ at the start of the step, so at the end, $K_0 + c_{j_0} > K_1$, and hence $K_1 - c_{j_0} < K_0$. Further, by assumption, $K_0 \leq K_1$ at the start, and since only K_1 was increased, this also holds at the end. □

We can now prove the following lemma:

Lemma 3 *If $f(x) = 1$, then at the end of the modified round-robin protocol, $K_1 \geq K_0$. Symmetrically, if $f(x) = 0$, then at the end of the modified round-robin protocol, $K_0 \geq K_1$.*

Proof There are two ways for the protocol to terminate. Either Alg0 or Alg1 is detected to have succeeded at the start of the repeat loop, or one of the two algorithms fails and the other is run to successful termination.

Suppose the former, and without loss of generality suppose it is Alg0 that succeeded. It follows that it was x_{j_0} that was tested at the end of the previous step (unless this is the first step, which would be an easy case), because otherwise, the success of Alg0 would have been detected in an earlier step.

Thus at the end of the last step, by Lemma 2, $K_1 \leq K_0$.

Suppose instead that one algorithm fails and without loss of generality, suppose it was Alg0 and thus we ran Alg1 to termination. Since Alg0 did not fail in a prior step, it follows that in the previous step, x_{j_0} was tested (unless this is the first step, which would be an easy case). Thus at the end of the previous step, by the invariant, $K_0 - c_{j_1} \leq K_1$ and so $K_0 \leq K_1 + c_{j_1}$. We have to run at least one step of Alg1 when we run it to termination. Thus running Alg1 to termination augments K_1 by c_{j_1}, and so at the end of the algorithm, we have $K_0 \leq K_1$. □

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 was 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 [18]. This algorithm selects items to place in the cover, one by one, based on a certain greedy choice rule. Briefly, Hochbaum's algorithm is based on the dual of the standard linear program for SET COVER. The variables of the dual correspond to the elements that need to be covered, and the constraints correspond to the subsets that can be used to form the cover. The algorithm chooses subsets to place in the cover, one by one until all elements are covered. At each iteration, the algorithm greedily chooses a dual variable and assigns it a non-zero value, causing a new constraint of the dual to become tight without violating any constraints. The subset corresponding to the newly tight constraint is then added to the cover.

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 order of increasing cost 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 prove 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 whose expected cost is within a factor of* $\frac{4}{\rho^k}$ *of the expected certificate cost. Here* ρ *is either the minimum* p_i *value or the minimum* $1 - p_i$ *value, whichever is smaller.*

Proof Let f be the input monotone k-DNF, defined on $x \in \{0, 1\}^n$. We will also use f to denote the function computed by this formula.

Let Alg be the algorithm for evaluating f that alternates between the Alg0 and Alg1 algorithms just described, using the modified round-robin protocol.

Let $S_1 = \{x | f(x) = 1\}$ and $S_0 = \{x | f(x) = 0\}$. Let $\mathbf{E}[\text{cost}_f(x)]$ denote the expected cost incurred by the round-robin algorithm in evaluating f on random x. Let $\text{cost}_f(x)$ denote the cost incurred by running the algorithm on x. Let $\mathbf{Pr}[x]$ denote the probability that x is realized with respect to the product distribution D_p. Thus $\mathbf{E}[\text{cost}_f(x)]$ is equal to $\sum_{x \in S_1} \mathbf{Pr}[x] \text{cost}_f(x) + \sum_{x \in S_0} \mathbf{Pr}[x] \text{cost}_f(x)$. Similarly, let $\text{cert}_f(x)$ denote the cost of the minimum cost certificate of f contained in x, and let $\mathbf{E}[\text{cert}_f(x)] = \sum_x \mathbf{Pr}[x] \text{cert}_f(x)$. We need to show that the ratio between $\mathbf{E}[\text{cost}_f(x)]$ and $\mathbf{E}[\text{cert}_f(x)]$ is at most $\frac{4}{\rho^k}$.

We consider first the costs incurred by Alg0 on inputs $x \in S_0$. Following the approach of Kaplan et al., we divide the tests performed by Alg0 into two categories, which we call useful and useless, and amortize the cost of the useless tests by charging them to the useful tests. More formally, we say that a test on variable x_i is useful to Alg0 if $x_i = 0$ (x_i is added to the 0-certificate in this case) and useless if $x_i = 1$. The number of useful tests on x is equal to the size of the certificate output by Alg0, and thus the total cost of the useful tests Alg0 performs on x is at most $k(\text{cert}_f(x))$.

Let $\text{cost}^0_f(x)$ denote the cost incurred by Alg0 alone when running Alg to evaluate f on x, and let $\text{cost}^1_f(x)$ denote the cost incurred by Alg1 alone. Suppose Alg0 performs a useless test on an $x \in S_0$, finding that $x_i = 1$. Let x' be the assignment produced from x by setting x_i to 0. Because $f(x) = 0$ and f is monotone, $f(x') = 0$ too. Because x and x' differ in only one bit, if Alg0 tests x_i on assignment x, it will test x'_i on x', and that test will be useful. Thus each useless test performed by Alg0 on $x \in S_0$ corresponds to a distinct useful test performed on an $x' \in S_0$. When x_i is tested, the probability that it is 1 is p_i, and the probability that it is 0 is $1 - p_i$. Each useless test contributes $c_i p_i$ to the expected cost, whereas each useful test contributes $(1 - p_i)c_i$. If we multiply the contribution of the useful test by $1/(1 - p_i)$, we get the contribution of both a useful and a useless test, namely c_i. To charge the cost of a useless test to its corresponding useful test, we can therefore multiply the cost of the useful test by $1/(1-p_i)$ (so that if, for example, $p_i = 1/2$, we charge double for the useful test). Because $1/(1 - p_i) \leq \frac{1}{\rho}$ for all i, it follows that $\sum_{x \in S_0} \mathbf{Pr}[x]\, \text{cost}^0_f(x) \leq \frac{1}{\rho} \sum_{x \in S_0} \mathbf{Pr}[x](k(\text{cert}_f(x)))$. Hence,

$$\frac{\sum_{x \in S_0} \mathbf{Pr}[x]\text{cost}^0_f(x)}{\sum_{x \in S_0} \mathbf{Pr}[x]\text{cert}_f(x)} \leq \frac{k}{\rho}. \tag{3}$$

We will now show, by induction on the number of terms of f, that $\mathbf{E}\left[\text{cost}^1_f(x)\right] / \mathbf{E}[\text{cert}_f(x)] \leq \frac{1}{\rho^k}$.

If f has only one term, it has at most k variables. In this case, Alg1 is just using the naïve algorithm which tests the variables in order of increasing cost until the function value is determined. Since the cost of using the naïve algorithm on x in this case is at most k times $\text{cert}_f(x)$, $\rho \leq 1/2$, and $k \leq 2^k$ for all $k \geq 1$, it follows that $\mathbf{E}\left[\text{cost}^1_f(x)\right] / \mathbf{E}[\text{cert}_f(x)] \leq k \leq 2^k \leq \frac{1}{\rho^k}$. Thus we have the base case.

Assume for the purpose of induction that $\mathbf{E}\left[\text{cost}^1_f(x)\right] / \mathbf{E}[\text{cert}_f(x)] \leq \frac{1}{\rho^k}$ holds for f having at most m terms. Suppose f has $m + 1$ terms. Let t denote the min-cost term. Let C denote the sum of the costs of literals in t, and k' the number of variables in t, so $k' \leq k$. If x does not satisfy term t, then after Alg1 evaluates term t on x, the results of the tests performed in the evaluation correspond to a partial assignment a to the variables in t. More particularly, if Alg1 tested exactly z variables of t, the test results correspond to the partial assignment a setting the $z - 1$ cheapest variables of t to 1 and the zth to 0, leaving all other variables in t unassigned. Thus there are k' possible values for a. Let T denote this set of partial assignments a.

For $a \in T$, let $f[a]$ denote the formula obtained from f by replacing any occurrences of variables in t by their assigned values in a (if a variable in t is not assigned in a, then occurrences of those variables are left unchanged). Let $T_0 = \{a \in T \mid f[a]$ is identically 0 $\}$, and let $T_* = T - T_0$. For any x, the cost incurred by Alg1 in evaluating t on x is at most C. For $x \in T_0$, Alg1 only evaluates t, so its total cost on x is at most C. Let $\mathbf{Pr}[a]$ denote the joint probability of obtaining the observed values of those variables tested in t. More formally, if W is the set of variables tested in t, $\mathbf{Pr}[a] = \prod_{i:x_i \in W \wedge x_i = 1} p_i \prod_{i:x_i \in W \wedge x_i = 0} (1 - p_i)$. We thus have the following recursive expression:

$$\mathbf{E}\left[\text{cost}^1_f(x)\right] \leq C + \sum_{a \in T_*} \mathbf{Pr}[a]\, \mathbf{E}\left[\text{cost}^1_{f[a]}(x[a])\right]$$

where $x[a]$ is a random assignment to the variables of f not assigned values in a, chosen independently according to the relevant parameters of D_p.

For any x satisfying t, since t is min-cost and f is monotone, $\text{cert}_f(x) = C$. Let $x \in T_*$, and let a_x be the partial assignment representing the results of the tests Alg1 performed in evaluating t on x. Let $\hat{x}$ be the restriction of x to the variables of f not assigned values by a_x. Any certificate for f that is contained in x can be converted into a certificate for $f[a_x]$, contained in $\hat{x}$, by simply removing the variables assigned values by a_x. It follows that $\text{cert}_f(x) \geq \text{cert}_{f[a_x]}(\hat{x})$.

Since $k' \leq k$, the probability that x satisfies the first term is at least ρ^k. By ignoring the $x \in T_0$ we get

$$\mathbf{E}[\text{cert}_f(x)] \geq \rho^k C + \sum_{a \in T_*} \mathbf{Pr}[a]\, \mathbf{E}[\text{cert}_{f[a]}(x[a])]$$

The ratio between the first term in the expression bounding $\mathbf{E}\left[\text{cost}^1_f(x)\right]$ to the first term in the expression bounding $\mathbf{E}[\text{cert}_f(x)]$ is equal to $1/\rho^k$. By induction, for each $a \in T_*$,

$$\mathbf{E}\left[\text{cost}^1_{f[a]}(x[a])\right] / \mathbf{E}[\text{cert}_{f[a]}(x[a])] \leq \frac{1}{\rho^k}.$$

Thus

$$\mathbf{E}\left[\text{cost}^1_f(x)\right] / \mathbf{E}[\text{cert}_f(x)] \leq \frac{1}{\rho^k}. \tag{4}$$

Setting $\beta = \sum_{x \in S_1} \mathbf{Pr}[x]\text{cost}_f(x) + \sum_{x \in S_0} \mathbf{Pr}[x]\text{cost}^1_f(x)$, we have

$$\mathbf{E}[\text{cost}_f(x)] = \beta + \sum_{x \in S_0} \mathbf{Pr}[x]\text{cost}^0_f(x).$$

By Lemma 3, the cost incurred by Alg on any $x \in S_1$ is at most twice the cost incurred by Alg1 alone on that x. Thus

$$\begin{aligned}\beta &\leq \sum_{x \in S_1} 2\mathbf{Pr}[x]\text{cost}^1_f(x) + \sum_{x \in S_0} \mathbf{Pr}[x]\text{cost}^1_f(x) \leq \sum_{x \in S_0 \bigcup S_1} 2\mathbf{Pr}[x]\text{cost}^1_f(x) \\ &= 2\,\mathbf{E}\left[\text{cost}^1_f(x)\right].\end{aligned}$$

It follows that

$$\mathbf{E}[\text{cost}_f(x)] \leq 2\,\mathbf{E}\left[\text{cost}^1_f(x)\right] + \sum_{x \in S_0} \mathbf{Pr}[x]\text{cost}^0_f(x). \tag{5}$$

Further, since $\mathbf{E}[\text{cert}_f(x)] \geq \sum_{x \in S_0} \mathbf{Pr}[x]\,\text{cert}_f(x)$,

$$\mathbf{E}[\text{cert}_f(x)] \geq \frac{1}{2}(\mathbf{E}[\text{cert}_f(x)] + \sum_{x \in S_0} \mathbf{Pr}[x]\text{cert}_f(x)). \tag{6}$$

It follows from the above numbered lines that $\mathbf{E}[\text{cost}_f(x)]/\,\mathbf{E}[\text{cert}_f(x)]$ is at most $\max\left\{\frac{4}{\rho^k}, \frac{2k}{\rho}\right\} = \frac{4}{\rho^k}$, since $k \geq 1$ and $\rho \leq \frac{1}{2}$. □

5.2 Monotone k-term DNF Formulas

We can use techniques from the previous subsection to obtain results for the class of monotone k-term DNF formulas as well. In Sect. 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 whose expected cost is within a factor of* $\max\left\{2k, \frac{2}{\rho}(1 + \ln k)\right\}$ *of the expected certificate cost. Here ρ is either the minimum p_i value or the minimum $1 - p_i$ value, whichever is smaller.*

Proof Let f be the input monotone k-term DNF, defined on $x \in \{0, 1\}^n$.

Just as in the proof of Theorem 2, we will use a modified round robin protocol that alternates between one algorithm for finding a 0-certificate (Alg0) and one for finding a 1-certificate (Alg1). Again, let $S_1 = \{x | f(x) = 1\}$ and $S_0 = \{x | f(x) = 0\}$.

However, in this case Alg0 will use GREEDY, Chvátal's well-known greedy algorithm for weighted SET COVER [5] instead of the Dual Greedy algorithm of Hochbaum. GREEDY simply maximizes, at each iteration, "bang for the buck" by selecting the subset that maximizes the ratio between the number of uncovered elements covered by the subset, and the cost of the subset. GREEDY yields an $H(m)$ approximation, where m is the number of ground elements in the SET COVER instance and $H(m)$ is the m^{th} harmonic number, which is upper bounded by $1 + \ln m$. Once again, we will view the terms as ground elements and the variables that evaluate to 0 as the subsets. Since f has at most k terms, there are at most k ground elements. On any $x \in S_0$, GREEDY will yield a certificate whose cost is within a factor of $1 + \ln k$ of the min-cost 0-certificate $\text{cert}_f(x)$, and thus the cost incurred by the useful tests on x (tests on x_i where $x_i = 0$) is at most $\text{cert}_f(x)(1 + \ln k)$. By multiplying by $1/\rho$ the charge to the variables that evaluate to 0, to account for the useless tests, we get that the expected cost incurred by Alg0 on x, for $x \in S_0$, is at most $\frac{1}{\rho}\,\text{cert}_f(x)(1 + \ln k)$.

Alg1 in this case simply evaluates f term by term, each time choosing the remaining term of minimum cost and evaluating all of the variables in it. Without loss of generality, let t_1 be the first (cheapest) term evaluated by Alg1 and let t_i be the i^{th} term evaluated. Suppose $x \in S_1$. If x falsifies terms t_1 through t_{i-1} and then satisfies t_i, $\text{cert}_f(x)$ is precisely the cost of t_i and Alg1 terminates after evaluating t_i. Since none of the costs of the first $i - 1$ terms exceeds the cost of t_i, the total cost of evaluating f is at most k times the cost of t_i. Hence, Alg1 incurs a cost of at most $k(\text{cert}_f(x))$.

By executing the two algorithms according to the modified round robin protocol, we can solve the problem of evaluating monotone k-term DNF with cost no more than double the cost incurred by Alg1, when $x \in S_1$, and no more than double the cost incurred by Alg0, when $x \in S_0$. Hence the total expected cost of the algorithm is within a factor of $\max\left\{2k, \frac{2}{\rho}(1+\ln k)\right\}$ of the expected certificate cost. □

We now prove that the problem of exactly evaluating monotone k-term DNF can be solved in polynomial time for constant k.

6 Minimum Cost 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. [16] to show some properties of optimal strategies for monotone DNF formulas. Then we will use these properties to compute an optimal strategy for monotone k-term DNF formulas. Greiner et al. [16] 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 variable 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, as indicated by the label of the associated internal node in the and-or tree. 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 $R(x_i) = p_i/c_i$. Further, variables 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*. Let a variable be considered *redundant* if testing it is unnecessary to determine the correct value of the function. Let a strategy be considered *redundant* if it tests redundant variables. A strategy S is considered *contiguous* with respect to a subset C of variables if, along all paths of S, whenever a variable of C is tested, S continues to test all of the other variables in C (provided that they do not become redundant as the results of the tests are obtained), until either C is exhausted or the path of S is terminated (reaches a leaf node). Greiner et al. show that, for any and-or tree, (without loss of generality, they assume that leaf-parents are OR nodes), there is an optimal strategy S that satisfies the following conditions:

(a) For any sibling variables x and y such that $R(y) > R(x)$, x is not tested before y on any root-to leaf path of S.
(b) For any *R-class* W, S is contiguous with respect to W.

We show 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 monotone DNF formula. Here maximal means that no proper super-set of a sibling class is a

sibling class. We will refer to the elements of a sibling class as siblings or sibling variables. For instance, all variables are siblings for an AND function, whereas no two variables are siblings in an OR function. All the other definitions can easily be adapted accordingly. In this case, the ratio of a variable x_i is $R(x_i) = \frac{1-p_i}{c_i}$, since each term of the monotone DNF formula is a conjunction.

It is possible to adapt the proof of Theorem 20 in [16] 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. The proofs of Lemma 4 and Theorem 4 are very similar to the proofs of Theorem 20 and Observation 26 in [16]. We present the proofs here for the sake of completeness.

Lemma 4 *Let x and y be sibling variables in a monotone k-term DNF. Let S be a non-redundant strategy that tests x first and if x is 1 then y is tested next. (Note that if x is 0, then y becomes redundant.) Let S' be obtained from S by exchanging x and y, which is also a non-redundant strategy. Then*

(a) if $R(y) > R(x)$ then S' has lower expected cost then S.
(b) if $R(y) = R(x)$ then S' has the same expected cost as S.

Proof The strategies S and S' are depicted in Fig. 1. $S^{+(-)}$ is the sub-strategy followed when the variable in the root is 1(0). Note that the subtrees S^-, rooted at x, and S^-, rooted at y, are identical since x and y are siblings. By definition of siblings, the function obtained by fixing x at 0 and the function obtained by fixing x at 1 and y at 0 are the sam. Since x and y are sibling variables, it is clear that S' is a non-redundant valid strategy. Let $C(S)$ denote the expected cost of strategy S.

$$C(S) = c_x + p_x c_y + (q_x + p_x q_y)C(S^-) + p_x p_y C(S^+)$$

and

$$C(S') = c_y + p_y c_x + (q_y + p_y q_x)C(S^-) + p_x p_y C(S^+).$$

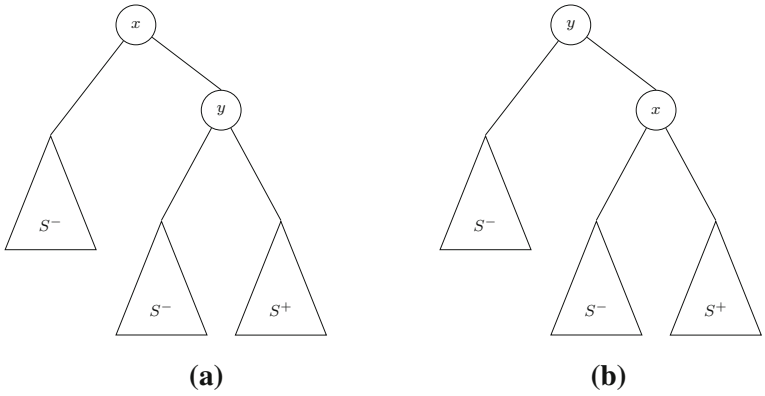

Fig. 1 Modification of S to obtain S'

Then we find that $C(S) - C(S') = q_y c_x - q_x c_y$. The result follows immediately. □

Theorem 4 *For any monotone* DNF*, there exists an optimal testing strategy S that satisfies both of the following conditions.*

(a) For any sibling variables x and y such that $R(y) > R(x)$, x is not tested before y on any root-to-leaf path of S.
(b) For any sibling class where each element of the sibling class has the same ratio, S is contiguous with respect to the sibling class.

Proof We prove the theorem by induction on the number of variables. The theorem holds for functions with a single variable. Suppose the theorem holds for any function that has at most $n-1$ variables. Now let us consider a function with n variables and an optimal strategy S for evaluating this function. Let x be the first variable of S, at the root node. Let S_x^+ and S_x^- be the sub-strategies of S when x is 1 and 0, respectively. Let us note that since S is optimal so are S_x^+ and S_x^- for the function restricted to $x = 1$ and $x = 0$ respectively. By induction, we can assume that S_x^+ and S_x^- satisfy the conditions of the theorem. So on any path from the root to the leaves in S_x^+ and S_x^-, the sibling appears before its other siblings that have smaller (worse) ratio and the siblings with the same ratio come one after another. Now let us assume that S does not satisfy the theorem. That means that x has at least one sibling that has the same or higher ratio that appears on some leaf to root path(s) of S. We will show, in this case, that there exists another optimal strategy that satisfies the conditions of the theorem. We will construct this strategy starting from S and by changing the order of sub-strategies of S. Let Y be the set of sibling variables of x with an equal or higher ratio than x. Let y be the variable with the minimum ratio among tests in Y. Since all sibling variables with the same ratio are tested one after another in S_x^+, we may assume that y is tested the last among all variables in Y by the sub-strategy S_x^+. Let M be the number of nodes of S_x^+ labeled by the variable y, and let $S_{y_1}, S_{y_2}, ..., S_{y_M}$ be the sub-trees of S_x^+ for which the roots are labeled by y. So y appears M times in S_x^+. For $k = 1, ..., M$, let $S_{y_k}^+$ and $S_{y_k}^-$ be the sub-strategies of Sy_k when y is true or false, respectively. Let S_r denote the part of S_x^+ that contains nodes that are not in $S_{y_1}, S_{y_2}, ..., S_{y_k}$. Now let us construct another strategy, S', where S' is constructed by taking S_r (so the root of S_r is the root of S') and changing S_{y_k} as follows for each $k = 1, ..., M$. We test x just before y. If x is 0, we continue with strategy $S_{y_k}^-$. If x is 1, we test y next. Depending on the value of y, we continue with strategy $S_{y_k}^+$ or $S_{y_k}^-$. Strategies S and S' are depicted in Fig. 2. First, we need to show that S' is indeed a valid strategy. In order to show this, we should show that for all paths in S', we correctly evaluate the original function. There are three possible cases with regard to the paths in S'.

(a) Both x and y are tested along the path. These paths differ only in the ordering of the variables from S. So they should evaluate the function correctly.
(b) x is tested along the path but not y. Since x and y are sibling variables, their values affect the function in the same way. When we compare these paths with the corresponding paths in S, the only difference is that x is tested instead of y. So these paths should evaluate the function correctly.

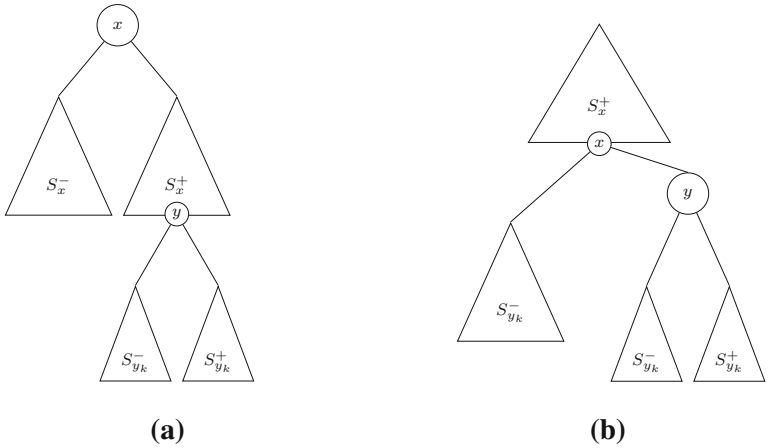

Fig. 2 Creating S' from S

(c) Neither x nor y is tested along the path. If we consider the corresponding paths in S, we see that in these paths y is not tested and x is 1. If x is 1 and y is not tested in S, that means that the results of the other variables provide a certificate and value of x is irrelevant, given the values of other variables. In S', these paths are the same as S except that x is not tested. Since x is irrelevant given the values of other variables on these paths, the function is evaluated correctly on these paths.

Now let S'' be obtained from S' by switching the labels of x and y. By Lemma 4, S'' is a valid strategy for evaluating the function and expected cost of S'' is not higher than the expected cost of S'. Since S'' satisfies the conditions of the theorem, it suffices to show that S' is optimal. Before showing that we note the following: if x has other siblings with the same ratio, then y should be one of those siblings. Hence S'' is contiguous with respect to these variables. We also observe that S'' is contiguous on any other set of tests with the same ratio that does not include x but include y since y is tested as the last test from Y. Also since x is tested right after y, when y is false, S'' preserves the right order of sibling variables.

Now we show that S' is not worse than S. We compare $C(S')$ with $C(S)$. Let us first consider the strategy S. In S, let C^r be the expected cost associated with nodes where x is 1 and y is not tested at all. That is, $C^r = \sum_{v \in S^r} p_v c(x_v)$ where the sum is over all nodes of S^r, p_v is the probability of the path from the root of S_x^+ to node v and $c(x_v)$ is the cost of the variable of node v. For any $k = 1, ..., M$, let p_{y_k} be the probability of the path from the root of S_x^+ to the root of S_{y_k}.

Expected cost of strategy S can be written in the following way:

$$C(S) = c_x + p_x C(S_x^+) + q_x C(S_x^-)$$

where

$$C(S_x^+) = C^r + \sum_{k=1}^{M} p_{yk} \left[c_y + p_y C(S_{y_k}^+) + q_y C\left(S_{y_k}^-\right) \right]$$

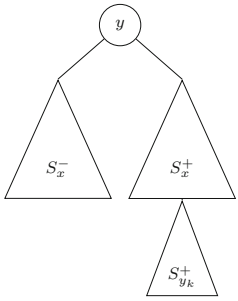

Fig. 3 Strategy $S_y^{x=1}$

whereas for the expected cost of S' we have

$$C(S') = C^r + \sum_{k=1}^{M} p_{y_k} \left\{ \left[c_x + q_x C\left(S_{y_k}^-\right) \right] + p_x \left[c_y + p_y C\left(S_{y_k}^+\right) + q_y C\left(S_{y_k}^-\right) \right] \right\}$$

By contradiction, let us assume that the expected cost of S' is higher than S. For notational convenience, let $D = C^r + \sum_{k=1}^{M} p_{y_k} C\left(S_{y_k}^-\right) - C\left(S_x^-\right)$ and $P^r = 1 - \sum_{k=1}^{M} p_{y_k}$, then we obtain

$$q_x D > P^r c_x.$$

Since $\sum_{k=1}^{M} p_{y_k}$ is the probability of reaching any node labeled by y in the strategy $S-$, $P^r \geq 0$. Consequently, $D > 0$ and so

$$\frac{q_x}{c_x} > \frac{P^r}{D}. \tag{7}$$

We will show that, in this case, we can replace the sub-strategy S_x^+ of S by a sub-strategy with strictly lower expected cost, contradicting the optimality of S_x^+.

Consider the strategy in Fig. 3, that starts evaluation by testing y first, if y is 1, it continues with S^r until it hits any node labeled as y. In this case since we already know that y is 1, we replace this with sub-strategy $S_{y_k}^+$. If y is 0 it continues with S_x^-. This strategy is a valid and non-redundant strategy when x is 1. Let us refer to this strategy as $S_y^{x=1}$ The expected cost of this strategy can be written as:

$$C\left(S_y^{x=1}\right) = c_y + p_y \left(C^r + \sum_{k=1}^{M} p_{y_k} C\left(S_{y_k}^+\right) + q_y C\left(S_x^-\right) \right).$$

Then we obtain

$$C\left(S_x^+\right) - C\left(S_y^{x=1}\right) = q_y D - P^r c_y.$$

But then from 7 and the fact that

$$\frac{q_y}{c_y} \geq \frac{q_x}{c_x}$$

it follows that $C\left(S_x^+\right) > C\left(S_y^{x=1}\right)$ which is a contradiction. □

In other words, there exists an optimal strategy such that on any path from the root to the leaf, sibling tests appear in non-increasing 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 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. Next, we provide a dynamic programming based method to find an optimal strategy.

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.*

Proof We will use a dynamic programming method similar to that used by Guijarro et al. in [17] for building decision trees for functions defined by truth tables. We will exploit condition (a) above.

Let f be the function that is defined by ϕ. We will construct a table P indexed by partial assignments to the $t = 2^k$ sibling classes. By Theorem 4, there is an optimal evaluation order of the variables within each sibling class. Let $1 \leq j \leq t$ index the sibling classes in arbitrary order. For each sibling class s^j, let us rename the variables contained in it x_i^j, where $1 \leq i \leq \ell^j$ refers to the position of the variable in the testing order according to their ratios $R(i) = q_i/c_i$, and where ℓ^j refers to the number of variables within the class s^j. Hence, for each class we will have $\ell^j + 2$ states in P: not evaluated, variable x_1^j evaluated to 1, variable x_2^j evaluated to 1, …variable $x_{\ell^j}^j$ evaluated to 1, any variable evaluated to 0. (Due to monotonicity, the evaluation of any variable to 0 ends the evaluation of the entire class.) Given the optimal ordering, the knowledge of which variable i of a sibling class was evaluated last is sufficient to determine which variable $i + 1$ should be evaluated next within that class. Given a partial assignment α that is being evaluated under an optimal testing strategy, let s_i^j denote the variable x_i^j that will be evaluated next for each class s^j (that is, that the values of variables x_h^j for all $h < i$ have already been revealed).

At each position α in the table, we will place the decision tree with the minimum expected cost that computes the function f_α, where f_α is the function f defined by ϕ projected over the partial assignment α. Then, once the full table P has been constructed, $P[*^n]$ (the value for the empty assignment) will provide the minimum expected cost decision tree for f.

For any Boolean function g, let $|g|$ denote the expected cost of the minimum expected cost decision tree consistent with g. For any partial assignment α and any

variable v not assigned a value in α, let $\alpha \cup v \leftarrow b$ denote the partial assignment created by assigning the value $b \in \{0, 1\}$ to v to extend α. Let c_i^j denote the cost of evaluating x_i^j, let p_i^j denote the probability that $x_i^j = 1$, and let q_i^j denote the probability that $x_i^j = 0$.

We can construct the table P using dynamic programming by following these rules:

1. For any complete assignment α, the minimum size decision tree has a cost of 0, since no variables need to be evaluated to determine its value. Hence, the value $P[\alpha] = 0$.
2. For any partial assignment α such that there exists a variable v that has not yet been evaluated and $f_{\alpha \cup v \leftarrow 0} = f_{\alpha \cup v \leftarrow 1}$, then $f_\alpha = f_{\alpha \cup v \leftarrow 0}$ and the entry $P[\alpha] = P[\alpha \cup v \leftarrow 0]$.
3. For any partial assignment α that does not meet conditions 1 or 2, then

$$|f_\alpha| = \min_{j,i} \left\{ c_i^j + p_i^j \left| f_{\alpha \cup x_i^j \leftarrow 1} \right| + q_i^j \left| f_{\alpha \cup x_i^j \leftarrow 0} \right| \right\}$$

Then we can fill in the entry for $P[\alpha]$ by finding the next variable s_i^j that has the minimum expected cost testing strategy, placing it at the root of a tree and creating left and right subtrees accordingly.

Let ℓ denote the size of the largest sibling class. Then, since there are t sibling classes, P will have size at most $t\ell$. Clearly, $\ell \leq n$, so the size of P is at most tn, and we can construct P in time $O(tn^t)$. Since $t = 2^k$ and k is constant, the dynamic program will run in time $O(n^{2^k})$. □

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 $k = O(\log \log n)$.*

Proof Under the uniformity assumption the ratios are the same for all variables. Hence, each sibling class will be evaluated as a single block and tested in an arbitrary order until either a variable evaluates to 0 or a term evaluates to 1, or until the sibling class is exhausted. Since we will evaluate each sibling class together, we can view each class as a single variable. Then we have a k-term DNF defined over 2^k variables. Let V be the set of the new variables. For each $v \in V$, let v_ℓ denote the number of "real" variables in v.

We can then find the optimal strategy using a dynamic programming method as before. The first two rules are as in the previous table formulation method. We will modify the third rule as follows:

For any partial assignment α that does not meet the first two conditions, then

$$|f_\alpha| = \min_{v \in V} \left\{ \sum_{i=1}^{v_\ell} \left[\left(\frac{1}{2} \right)^i |f_{\alpha \cup v \leftarrow 0}| \right] + \left(\frac{1}{2} \right)^{v_\ell} |f_{\alpha \cup v \leftarrow 1}| + 1 \right\}.$$

which follows directly from the unit costs and uniform probabilities.

The size of the table will be only 3^t; hence we can determine the optimal testing strategy over the sibling classes in time $O\left(2^{2^k}\right)$. The evaluation of the original n variables can hence be done in time $O\left(n2^{2^k}\right)$. □

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 the NYU School of Engineering and Sarah Allen and Devorah Kletenik were students there. We thank anonymous referees for their helpful suggestions.

References

1. Bar-Noy, A., Bellare, M., Halldórsson, M., Shachnai, H., Tamir, T.: On chromatic sums and distributed resource allocation. Inform. Comput. **140**(2), 183–202 (1998)
2. Bellala, G., Bhavnani, S.K., Scott, C.: Group-based active query selection for rapid diagnosis in time-critical situations. Inst. Elect. Electron. Eng. Trans. Inform. Theory **58**(1), 459–478 (2012)
3. Boros, E., Ünlüyurt, T.: Computing tools for modeling, optimization and simulation, operations research/computer science interfaces series. In: Sequential Testing of Series-Parallel Systems of Small Depth, vol. 12, pp. 39–73. Springer, Heidelberg (2000)
4. Charikar, M., Fagin, R., Guruswami, V., Kleinberg, J., Raghavan, P., Sahai, A.: Query strategies for priced information. J. Comput. Syst. Sci. **64**(4), 785–819 (2002)
5. Chvátal, V.: A greedy heuristic for the set-covering problem. Math. Oper. Res. **4**(3), 233–235 (1979)
6. Cicalese, F., Gagie, T., Laber, E., Milanič, M.: Competitive boolean function evaluation: beyond monotonicity, and the symmetric case. Discret. Appl. Math. **159**(11), 1070–1078 (2011)
7. Cicalese, F., Laber, E.: On the competitive ratio of evaluating priced functions. J. ACM **58**(3), 9:1–9:40 (2011)
8. Cicalese, F., Laber, E., Saettler, A.M.: Diagnosis determination: decision trees optimizing simultaneously worst and expected testing cost. In: Proceedings of the 31st International Conference on Machine Learning, pp. 414–422 (2014)
9. Cox, L.A., Qiu, Y., Kuehner, W.: Heuristic least-cost computation of discrete classification functions with uncertain argument values. Ann. Oper. Res. **21**(1–4), 1–29 (1989). Linkages with artificial intelligence
10. Deshpande, A., Hellerstein, L.: Flow algorithms for parallel query optimization. In: IEEE 24th International Conference on Data Endineering, pp. 754–763. IEEE (2008)
11. Deshpande, A., Hellerstein, L., Kletenik, D.: Approximation algorithms for stochastic boolean function evaluation and stochastic submodular set cover (2013). arXiv:1303.0726
12. Deshpande, A., Hellerstein, L., Kletenik, D.: Approximation algorithms for stochastic Boolean function evaluation and stochastic submodular set cover. In: Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 1453–1467 (2014)
13. Feige, U.: A threshold of $\ln n$ for approximating set cover. J. ACM **45**, 314–318 (1998)
14. Golovin, D., Krause, A.: Adaptive submodularity: theory and applications in active learning and stochastic optimization. J. Artif. Intell. Res. **42**, 427–486 (2011)
15. Golovin, D., Krause, A., Ray, D.: Near-optimal Bayesian active learning with noisy observations. In: Conference proceeding in the 23rd annual NIPS. Advances in Neural Information Processing Systems, pp. 766–774 (2010)
16. Greiner, R., Hayward, R., Jankowska, M., Molloy, M.: Finding optimal satisficing strategies for AND-OR trees. Artif. Intell. **170**(1), 19–58 (2006)
17. Guijarro, D., Lavín, V., Raghavan, V.: Exact learning when irrelevant variables abound. Inform. Process. Lett. **70**(5), 233–239 (1999)
18. Hochbaum, D.S.: Approximation algorithms for the set covering and vertex cover problems. SIAM J. Comput. **11**(3), 555–556 (1982)
19. Ibaraki, T., Kameda, T.: On the optimal nesting order for computing N-relational joins. ACM Trans. Database Syst. **9**(3), 482–502 (1984)

20. Kaplan, H., Kushilevitz, E., Mansour, Y.: Learning with attribute costs. In: Proceedings of the 37th Annual ACM Symposium on Theory of Computing, pp. 356–365 (2005)
21. Krishnamurthy, R., Boral, H., Zaniolo, C.: Optimization of nonrecursive queries. In: Proceedings of the 12th International Conference on Very Large Data Bases, vol. 86, pp. 128–137 (1986)
22. Moshkov, M., Chikalov, I.: Bounds on average weighted depth of decision trees. Fundam. Inform. **31**(2), 145–156 (1997)
23. Moshkov, M.J.: Approximate algorithm for minimization of decision tree depth. In: Wang, G., Liu, Q., Yao, Y., Skowron, A., (eds.) Rough Sets, Fuzzy Sets, Data Mining, and Granular Computing, pp. 611–614. Springer, Heidelberg (2003)
24. Munagala, K., Babu, S., Motwani, R., Widom, J.: The pipelined set cover problem. In: Proceedings of the 10th International Conference on Database Theory, pp. 83–98. Springer (2005)
25. Skutella, M., Williamson, D.P.: A note on the generalized min-sum set cover problem. Oper. Res. Lett. **39**(6), 433–436 (2011)
26. Srivastava, U., Munagala, K., Widom, J., Motwani, R.: Query optimization over web services. In: Proceedings of the 32nd international conference on Very Large Data Bases, pp. 355–366. VLDB Endowment (2006)
27. Ünlüyurt, T.: Sequential testing of complex systems: a review. Discret. Appl. Math. **142**(1–3), 189–205 (2004)