

# The Hardness of the Expected Decision Depth Problem

In memory of the second author's loving father, Yaakov (Yasha) Rosenfeld

Dana Ron <sup>\*,1</sup>

*Department of EE-Systems, Tel Aviv University, Ramat Aviv, Israel*

Amir Rosenfeld <sup>2</sup>

*Altair Semiconductor, Hod Hasharon, Israel.*

Salil Vadhan <sup>3</sup>

*Division of Engineering and Applied Sciences, Harvard University, Cambridge, MA, USA*

---

## Abstract

Given a function  $f$  over  $n$  binary variables, and an ordering of the  $n$  variables, we consider the *Expected Decision Depth* problem. Namely, what is the expected number of bits that need to be observed until the value of the function is determined, when bits of the input are observed according to the given order. Our main finding is that this problem is  $\#P$ -hard. In particular, hardness holds when the function  $f$  is represented as a decision tree.

Added  
abstract

*Key words:* Computational Complexity, Decision Trees  
*1991 MSC:* 68Q17

---

<sup>\*</sup> Corresponding author.

*Email addresses:* `danar@eng.tau.ac.il` (Dana Ron),  
`amirrosenfeld@yahoo.com` (Amir Rosenfeld), `salil@eecs.harvard.edu` (Salil Vadhan).

<sup>1</sup> Work partially done while a fellow at the Radcliffe Institute for Advanced Study, Harvard University. Also supported by the Israel Science Foundation.

<sup>2</sup> This work was part of the author's MsC thesis at the Department of EE – Systems, Tel Aviv University

<sup>3</sup> Work done while a fellow at the Radcliffe Institute for Advanced Study, Harvard University. Also supported by a Sloan Research Fellowship and NSF grant CCF-0133096.

# 1 Introduction

Added  
motivating  
scenario

Consider the following communication problem between two types of servers. One type of server is a *function* server – it has some function  $f$  over  $n$  binary variables. The other type of server is an *input* server – it has some input  $\alpha \in \{0, 1\}^n$ . For a fixed ordering of the variables, the input server sends, one by one, the corresponding bits of the input it holds to the function server. Once the value  $f(\alpha)$  is determined given the bits that were sent, the function server notifies the input server and the input server sends no more bits.

One interesting problem in this context is to find an ordering that (approximately) minimizes the number of bits sent until the value of the function is determined, when averaging over all inputs. Here we consider the more basic problem of computing this average number, when given a particular ordering of the inputs. Solving this, seemingly simpler, computational problem, could possibly serve as a building block for finding an optimal order. However, we show that computing this average number is hard.

## 1.1 Formal Problem Definition

Let  $f : \{0, 1\}^n \rightarrow \{0, 1\}$  be a Boolean function and let  $\pi : [n] \rightarrow [n]$  (where  $[n] \stackrel{\text{def}}{=} \{1, \dots, n\}$ ) be an ordering on the variables of the function. For a fixed choice of  $f$  and  $\pi$ , the *decision depth* of  $\alpha \in \{0, 1\}^n$  with respect to  $f$  and  $\pi$ , denoted  $\text{DD}_{f,\pi}(\alpha)$ , is the minimum number  $j$  such that the value of  $f(\alpha)$  is determined given  $\alpha_{\pi(1)}, \dots, \alpha_{\pi(j)}$ . That is,

$$\text{DD}_{f,\pi}(\alpha) \stackrel{\text{def}}{=} \min_{0 \leq j \leq n} \left\{ \forall \beta \in \{0, 1\}^n \text{ s.t. } \beta_{\pi(i)} = \alpha_{\pi(i)} \forall i \leq j, f(\beta) = f(\alpha) \right\} .$$

Note that if  $\text{DD}_{f,\pi}(\alpha') = 0$  for some  $\alpha'$ , then  $\text{DD}_{f,\pi}(\alpha) = 0$  for every  $\alpha$ , and this holds if and only if  $f$  is a constant function (i.e., either the all 0 function or the all 1 function). On the other hand observe that there exist functions, such as the parity function, for which  $\text{DD}_{f,\pi}(\alpha) = n$  for every  $\pi$  and  $\alpha$ .

The *expected decision depth* of  $f$  with respect to  $\pi$ , denoted  $\text{EDD}(f, \pi)$ , is simply the expected value of  $\text{DD}_{f,\pi}(\alpha)$ , where the expectation is taken over the uniform choice of  $\alpha$ . Namely,

$$\text{EDD}(f, \pi) \stackrel{\text{def}}{=} \text{Exp}_{\alpha \in \{0,1\}^n} [\text{DD}_{f,\pi}(\alpha)] . \tag{1}$$

In this paper we consider the problem of computing  $\text{EDD}(f, \pi)$  for  $f$  that is given as a decision tree, and refer to this problem as EDD. We note that computing  $\text{DD}_{f,\pi}(\alpha)$  for any given  $\alpha$  when  $f$  is given in the form of a decision

Removed  
poly-size

tree can be done in time polynomial in the size of the tree.<sup>4</sup> Observe that in contrast, if  $f$  is given in the form of a general CNF (or DNF) function, then computing  $DD_{f,\pi}(\alpha)$  is NP-hard. This is true since an algorithm for computing  $DD_{f,\pi}$  in this case can be used to decide satisfiability of (a CNF function)  $f$ . This follows from the aforementioned fact that  $DD_{f,\pi}(\alpha) = 0$  (for some  $\alpha$  and hence for every  $\alpha$ ) if and only if  $f$  is either a tautology or is unsatisfiable.

For the rest of this paper, we consider only functions represented as decision trees, and refer to EDD as the problem restricted to this representation.

## 1.2 Our Result

**Theorem 1** *EDD is computationally equivalent to #P. That is, EDD reduces to some problem in #P and every problem in #P reduces to EDD.*

Added  
explanation

Observe that (for technical reason) we do not show that EDD is in #P, but rather that it reduces to a problem in #P. In particular, EDD cannot belong to #P since it typically does not have an integer value. However, from a complexity point of view, this technicality is not of much importance.

The main part of the proof of Theorem 1, which is proving #P-hardness, is done by a reduction from #SAT to EDD. We note that the reduction is a Cook reduction (i.e., polynomial-time oracle reduction), and leave open the question of the existence of a Karp reduction (i.e., polynomial-time mapping reduction). We also note that this hardness result should be contrasted with the fact that obtaining an *approximation* of EDD (whether additive or multiplicative<sup>5</sup>) can be done efficiently by sampling, where the sample and time complexity of the algorithm are polynomial in  $n$  and  $1/\epsilon$ , where  $\epsilon$  is a given approximation parameter.

Finally we consider a variant of EDD in which the cost does not take into account variables that do not influence the value of the function given the assignment to previous variables in the order  $\pi$  (as considered in [6], discussed below). That is, the order is fixed, but a variable  $v$  may be “skipped” if, for the given assignment to the previous variables, for every assignment to the remaining variables, the value of the function is the same when  $v = 0$  and when  $v = 1$ . This corresponds to a scenario in which there is direct access to

---

<sup>4</sup> For each variable (according to the order defined by  $\pi$ ), once the value of the variable is determined, we can efficiently modify the decision tree by restricting all appearances of the variable in the tree to this value. Once all leaves of the decision tree have the same label, the output of the function is decided.

<sup>5</sup> Note that unless  $f$  is a constant function, which can be determined efficiently when  $f$  is a decision tree,  $1 \leq \text{EDD}(f, \pi) \leq n$  for every  $\pi$ .

the input, rather than the communication scenario described at the start of the introduction. This variant is computationally equivalent to #P as well.

### 1.3 Related Work

Variants of the EDD problem have been considered in several papers (e.g., [2,3,6], and more remotely related [1,4,5]). The most closely related work is the paper of Kaplan, Kushilevitz and Mansour [6]. They consider a more general optimization problem in which each variable,  $x_i$ , has an associated cost,  $c_i$ , and they seek an ordering of the variables that (approximately) minimizes the expected cost with respect to a known distribution  $D$  over  $\{0,1\}^n$ .<sup>6</sup> They show that if  $f$  is a single clause (disjunction of literals) and  $D$  is a product distribution then the optimal ordering can be found in polynomial time. When  $D$  is a general distribution (but  $f$  is still a single clause) then the problem is NP-hard, but there is a polynomial time 4-approximation algorithm. This approximation holds more generally for read-once DNF. Finally they consider the case where  $f$  is a monotone function whose DNF representation has at most  $m$  terms and its CNF representation has at most  $m$  clauses. They show that if every variable has unit cost and the distribution over inputs is uniform (as in our case), then there is an algorithm that evaluates  $f$  whose expected cost is  $O(\log m)$  times the optimum expected cost.

### 1.4 Open Problems

As noted at the start of the introduction, an interesting open problem is *finding* a permutation  $\pi$  for which  $\text{EDD}(f, \pi)$  is approximately minimized when  $f$  is given as a polynomial size decision tree. Another problem is to improve/extend the results in [6] to non-monotone functions. Recall that for monotone functions with small CNF and DNF representation, Kushilevitz et. al. [6] do not find a permutation whose expected cost is some factor larger than the minimum expected cost but rather describe an adaptive algorithm with such a cost for computing the function.

---

<sup>6</sup> To be precise, they assume that they have access to an oracle that given any assignment to a subset of the variables, returns the conditional probability that the function  $f$  is 1 (so that in particular, satisfiability is no longer an issue). Their cost is defined slightly differently from ours, where the cost takes into account only variables that influence the value of the function given the setting of the previous variables queried.

## 2 Preliminaries

Added  
definitions

For every binary relation  $R \subseteq \Sigma^* \times \Sigma^*$ , where  $\Sigma$  is some finite alphabet, the *counting function*  $\#R : \Sigma^* \rightarrow \mathbb{N}$  is defined by  $\#R(x) \stackrel{\text{def}}{=} |\{y : (x, y) \in R\}|$ . The class  $\#P$  is the class of all counting functions  $\#R$  where  $R$  is an NP relation. That is,  $R$  is polynomial-time decidable, and there exists a polynomial  $p(\cdot)$  such that for every  $(x, y) \in R$ , it holds that  $|y| \leq p(|x|)$ . In particular, let  $R$  be the SAT relation, that is, SAT consists of all pairs  $(\phi, \tau)$  such that  $\phi$  is a CNF function, and  $\tau$  is an assignment that satisfies  $\phi$ . For any CNF function  $\phi$ , we have that  $c_{\text{SAT}}(\phi)$  is the number of satisfying assignments of  $\phi$ . Since SAT is an NP relation, the function  $c_{\text{SAT}}$  is in  $\#P$ , and furthermore, it is  $\#P$ -complete.

Recall that a decision tree  $T$  over a set of variables  $V$  is a (binary) tree whose internal nodes are associated with variables in  $V$ , and whose leaves have labels in  $\{0, 1\}$ . For each internal node, one of its outgoing edges is labeled 0 and the other is labeled 1. The value of the function  $f_T$  computed by the decision tree  $T$  for a particular assignment  $\alpha : V \rightarrow \{0, 1\}$ , is determined as follows. Starting from the root of the tree, and until a leaf is reached, at each step we consider the variable  $v$  associated with the current node, and take the outgoing edge labeled  $\alpha(v)$ . The value of  $f_T(\alpha)$  is the value at the final leaf. For simplicity, we shall equate the tree  $T$  with the function  $f_T$ . In all that follows we consider trees whose size (number of nodes) is polynomial in  $n$ .

## 3 Proof of Theorem 1

In the next two subsections we prove the two directions of the equivalence stated in Theorem 1. We start with the easy direction.

### 3.1 EDD is reducible to $\#P$

Modified  
subsection

Let TDD (which stands for *total* decision depth), be defined as follows:  $\text{TDD}(T, \pi) = \sum_{\alpha \in \{0,1\}^n} \text{DD}_{T,\pi}(\alpha)$ . Thus,  $\text{EDD}(T, \pi) = 2^{-n} \cdot \text{TDD}(T, \pi)$ . Since EDD clearly reduces to TDD, it suffices to show that  $\text{TDD} \in \#P$ . To this end we need to define an NP relation  $R$  such that  $\text{TDD} = \#R$  (where the counting function  $\#R$  is as defined in Section 2). Specifically, we define

$$R = \left\{ ((T, \pi), (\alpha, k)) : \text{DD}_{T,\pi}(\alpha) \geq k \right\} \quad (2)$$

where  $\alpha$  ranges over all assignments to the variables in  $T$  and  $k$  ranges over all positive integers between 1 and the number of variables. Clearly,  $(\alpha, k)$  is of size linear in  $(T, \pi)$ , and since  $\text{DD}_{T,\pi}(\alpha)$  is computable in polynomial time given  $T$  and  $\pi$ , it follows that  $R$  is polynomial-time recognizable. Therefore,  $R$  is an NP relation, as required.

For any given choice of  $(T, \pi)$  let  $R(T, \pi) = \{(\alpha, k) : ((T, \pi), (\alpha, k)) \in R\}$ . By definition of the counting function  $\#R$ , we have that  $\#R(T, \pi) = |R(T, \pi)|$ . By definition of  $R$ , if  $\text{DD}_{T,\pi}(\alpha) = t$ , then  $(\alpha, k) \in R(T, \pi)$  for every  $k \leq t$  and  $(\alpha, k) \notin R(T, \pi)$  for every  $k > t$ . In other words, for every  $\alpha$ , the number of integers  $k$  such that  $(\alpha, k) \in R(T, \pi)$  is  $\text{DD}_{T,\pi}(\alpha)$ . Hence,

$$\text{TDD}(T, \pi) = \sum_{\alpha} \text{DD}_{T,\pi}(\alpha) = |R(T, \pi)| = \#R(T, \pi) \quad (3)$$

### 3.2 $\#P$ is reducible to EDD

In this subsection we prove:

**Lemma 2**  $\#SAT$  reduces to EDD.

As noted in the introduction, we show a Cook reduction and leave open the question of the existence of a Karp reduction. Specifically, let  $\phi : \{0, 1\}^n \rightarrow \{0, 1\}$  be a 3-CNF function with  $m$  clauses over the set of variables  $X = \{x_1, \dots, x_n\}$ . We assume, without loss of generality, that  $\phi$  is a function of all  $n$  variables, so that, in particular the size of  $\phi$  is greater than  $n$ . We shall construct two decision trees,  $T_1$  and  $T_2$  over  $n' = O(n)$  variables, and an ordering  $\pi$  over  $[n']$ , such that the number of satisfying assignments of  $\phi$  can be computed exactly given the difference between the expected decisions depths of the two trees, that is, given  $\text{EDD}(T_2, \pi) - \text{EDD}(T_1, \pi)$ .

#### 3.2.1 The high-level idea

We start by giving the high level idea of the construction. Both trees will be defined over the set of variables  $X$  of  $\phi$  as well as several  $(\log m + O(1))$  auxiliary variables. Let the union of  $X$  with the auxiliary variables be denoted by  $V$ . The two trees and the ordering will be such that the following holds. For every assignment  $\alpha$  to the tree variables (which, in particular is an assignment to the set of variables,  $X$ , of  $\phi$ ), if  $\alpha$  does not satisfy  $\phi$ , then  $\text{DD}_{T_2,\pi}(\alpha) - \text{DD}_{T_1,\pi}(\alpha) = 0$ , while if  $\alpha$  satisfies  $\phi$ , then  $\text{DD}_{T_2,\pi}(\alpha) - \text{DD}_{T_1,\pi}(\alpha) = 1$ . This will allow for computing the number of satisfying assignments of  $\phi$  (among all assignments to  $X$ ) from  $\text{EDD}(T_2, \pi) - \text{EDD}(T_1, \pi)$  as we show in detail subsequently.

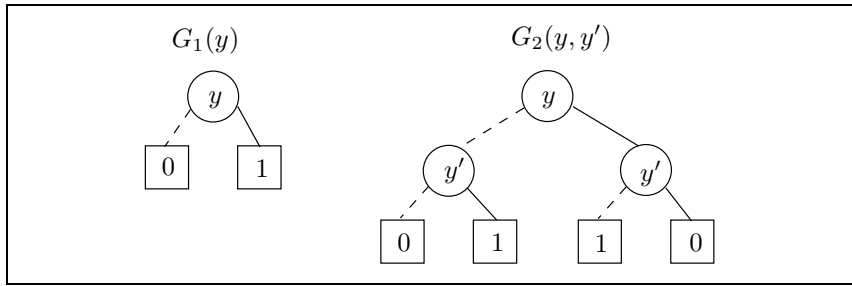


Fig. 1. The two types of “waiting” gadgets used to construct  $T_1$  and  $T_2$ . On the left is a single-variable gadget  $G_1(y)$ , and on the right, a two-variable gadget  $G_2(y, y')$ . The solid edges are labeled 1 and the dashed ones are labeled 0.

The two trees are constructed based on the clauses of the CNF function  $\phi$ . Namely, for each clause  $C$  of  $\phi$  there is a subtree in each of the two trees, which contains the variables of the clause and auxiliary variables; there is a unique path from the root of the tree to this clause-subtree (using auxiliary variables). The difference between the trees is in the construction of the clause-subtrees. The structure of these subtrees together with the ordering  $\pi$  will ensure the desired property regarding satisfying and non-satisfying assignments for  $\phi$ . Specifically, in the case of a satisfying assignment for  $\phi$ , the ordering  $\pi$  will “force” the computation on the tree  $T_2$  to read one more (auxiliary) variable as compared to the computation on  $T_1$ , while in the case of a non-satisfying assignment the computations on the two trees are essentially the same.

### 3.2.2 Details of the construction

At the heart of the construction are two simple gadgets. For a variable  $y$ , the *single-variable* gadget  $G_1(y)$  is a decision tree that outputs the assignment to the variable  $y$ . For a pair of variables  $y$  and  $y'$ , the *two-variable* gadget  $G_2(y, y')$  is a tree that computes the function  $y \oplus y'$ . For an illustration see Figure 1. Thus the output value of  $G_1(y)$  is determined once the variable  $y$  is read, and the output value of  $G_2(y, y')$  is determined only once both  $y$  and  $y'$  are read. We think of these gadgets as *waiting* gadgets, since they force the decision to wait until certain variables are read.

Using the above types of gadgets, the decision tree  $T_1$  is constructed as follows over the set of variables  $V$  which includes all variables in  $X$  (the variables of  $\phi$ ), as well as auxiliary variables  $y_0, y_1, y'_1$  and  $v_1, \dots, v_k$  where  $k = \lceil \log m \rceil$ . For each clause  $C_j$  in the function  $\phi$ , there is a subtree  $t_j$  in  $T_1$ . The subtree computes the function “if  $C_j$  then  $G_1(y_1)$  else  $G_1(y_0)$ ”. For an illustration see Figure 2. We “put together” the different subtrees  $t_1, \dots, t_m$  by using the variables  $v_1, \dots, v_k$ . Namely, we construct a binary tree, where nodes in level  $i$  are associated with  $v_i$ , and where each node at level  $k$  has two children that are roots of two different clause-subtrees that were defined above. If  $m$  is not a power of 2 then some of the clause-subtrees will be children of nodes at level

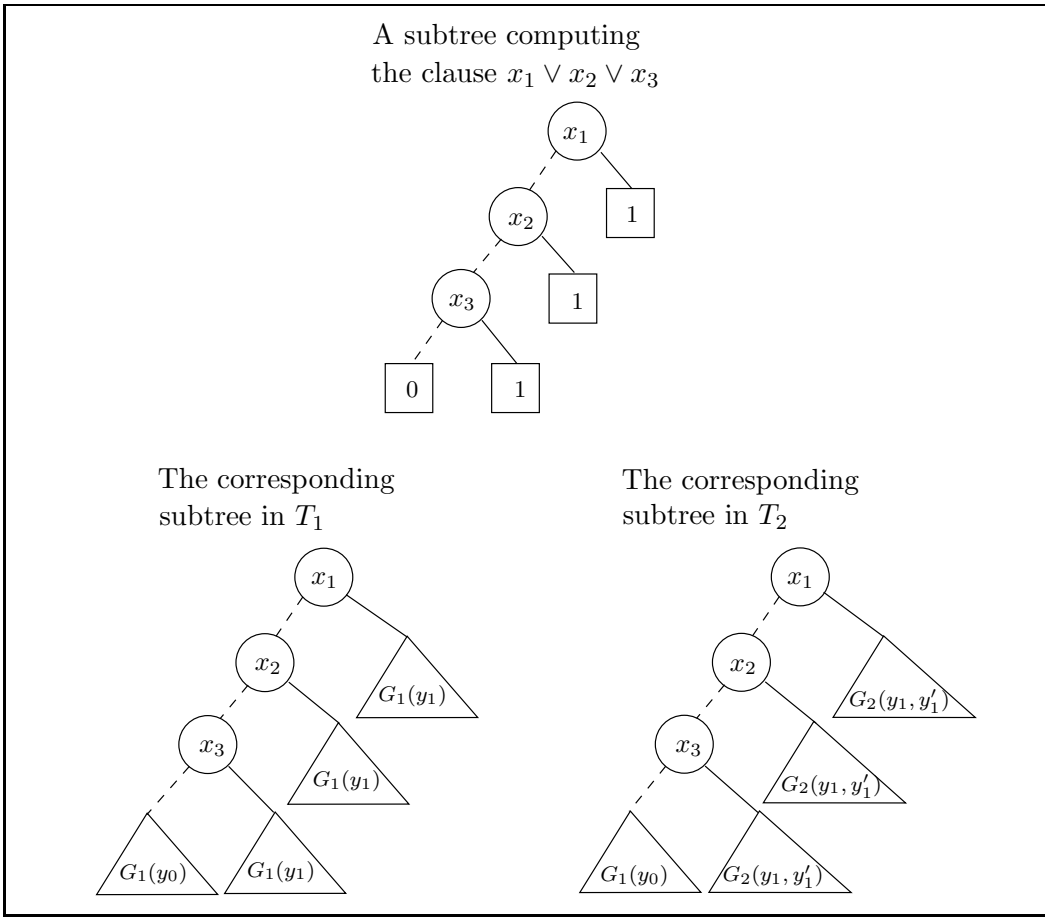


Fig. 2. On the top is the CNF clause  $(x_1 \vee x_2 \vee x_3)$ , computed by a decision tree over the three variables. On the bottom left is the corresponding subtree in  $T_1$ , and on the bottom right is the corresponding subtree in  $T_2$ .

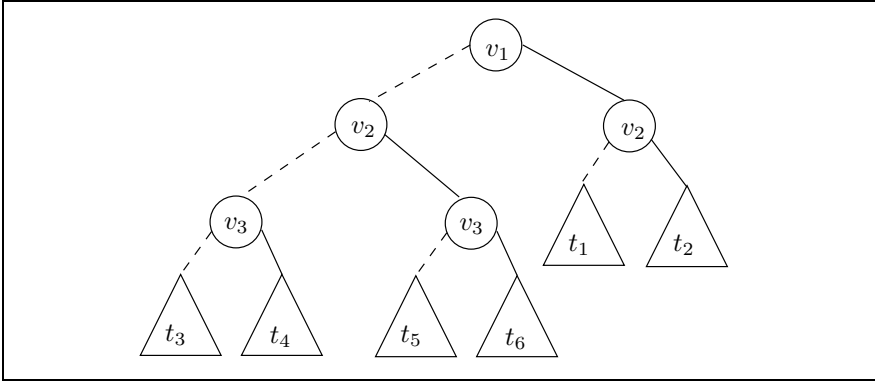


Fig. 3. The structure of decision trees  $T_1$  and  $T_2$  when the number of clauses in  $\phi$  is 6.

$k - 1$ . For an illustration see Figure 3.

The decision tree  $T_2$  is constructed exactly as  $T_1$  with the following important exception: instead of each single-variable gadget  $G_1(y_1)$  in  $T_1$ , we put a two-variable gadget  $G_2(y_1, y_1')$ . That is, the subtree  $t_j$  computes the function: “If



$C_j$  then  $G_2(y_1, y'_1)$  else  $G_1(y_0)$ ".<sup>7</sup> As we shall see momentarily, this, together with the choice of ordering of the variables, will "force" the decision depth to be larger by 1 in  $T_2$  for every assignment that satisfies  $\phi$ , while forcing equal decision depth for non-satisfying assignments of  $\phi$ . With slight abuse of notation we may sometimes refer to clauses that belong to a certain subtree  $T'$  (where we mean clauses that correspond to subtrees within  $T'$ ).

The ordering,  $\pi$ , of the variables is defined as follows. First appear the  $n$  variables  $x_1, \dots, x_n$ , then the variables  $y_1$  and  $y'_1$ , after them  $v_1, \dots, v_k$ , and finally  $y_0$ . We shall show that  $\phi$  is satisfiable if and only if  $\text{EDD}(T_1, \pi) < \text{EDD}(T_2, \pi)$  and furthermore, the number of satisfying assignments of  $\phi$  among all assignments to  $X$ , equals  $2^n \cdot (\text{EDD}(T_2, \pi) - \text{EDD}(T_1, \pi))$ . For the sake of succinctness, we continue with the convention used in the high level discussion by which we say that an assignment  $\alpha : V \rightarrow \{0, 1\}$  satisfies (does not satisfy)  $\phi$ , when its restriction to  $X$  satisfies (respectively, does not satisfy)  $\phi$ .

**Lemma 3** *Let  $\alpha$  be any fixed assignment  $\alpha : V \rightarrow \{0, 1\}$ .*

- (1) *If  $\alpha$  satisfies  $\phi$ , then  $\text{DD}_{T_2, \pi}(\alpha) - \text{DD}_{T_1, \pi}(\alpha) = 1$ .*
- (2) *If  $\alpha$  does not satisfy  $\phi$  then  $\text{DD}_{T_2, \pi}(\alpha) - \text{DD}_{T_1, \pi}(\alpha) = 0$ .*

We prove Lemma 3 momentarily, but first prove Lemma 2 (i.e., #P reduces to EDD) based on Lemma 3.

**Proof of Lemma 2:** Given a 3-CNF function  $\phi$  over  $n$  variables, we construct the trees  $T_1$  and  $T_2$  and the ordering  $\pi$  as described in the preceding discussion. Clearly this can be done in time polynomial in the size of  $\phi$ . Assuming we have an oracle for computing EDD, we compute and output  $\hat{s}(\phi) = 2^n \cdot (\text{EDD}(T_2, \pi) - \text{EDD}(T_1, \pi))$ . By Lemma 3 and the fact that for each assignment  $\tau : X \rightarrow \{0, 1\}$  that satisfies  $\phi$  there are  $2^{|V|-n}$  assignments  $\alpha : V \rightarrow \{0, 1\}$  that extend  $\tau$  to  $V$ ,

Added  
elaboration

$$\hat{s}(\phi) = 2^n \cdot 2^{-|V|} \cdot \sum_{\alpha \in \{0,1\}^{|V|}} (\text{DD}_{T_2, \pi}(\alpha) - \text{DD}_{T_1, \pi}(\alpha)) \quad (4)$$

$$= 2^{n-|V|} \cdot |\{\alpha \in \{0, 1\}^{|V|} : \alpha \text{ satisfies } \phi\}| \quad (5)$$

$$= |\{\tau \in \{0, 1\}^n : \tau \text{ satisfies } \phi\}|. \quad (6)$$

Therefore,  $\hat{s}(\phi)$  equals the number of satisfying assignments of  $\phi$  (among all  $2^n$  assignments to  $X$ ).  $\square$

**Proof of Lemma 3:** Let  $\alpha : V \rightarrow \{0, 1\}$ . Suppose  $\alpha$  satisfies  $\phi$ . Since  $\phi$  is a

<sup>7</sup> We note that the variable  $y'_1$  does not appear in the tree  $T_1$ . This was done for the sake of simplicity, at the cost of elegance. This issue is addressed (in a more general context) in Subection 3.3.

conjunction of its clauses, this means that  $\alpha$  satisfies each clause. Namely, for each clause  $C$ , at least one of its literals is assigned 1 by  $\alpha$ . This implies that *no matter what is the assignment* to  $v_1, \dots, v_k$ , the computation path on  $T_1$  reaches a  $G_1(y_1)$  gadget, and the computation path on  $T_2$  reaches a  $G_2(y_1, y_2)$  gadget. Namely, the output value of  $T_1$  is determined once the assignment to the variable  $y_1$  is determined but not before that, while the output value of  $T_2$  is determined once the assignment to both  $y_1$  and  $y'_1$  is determined (and not before that). Given the ordering  $\pi$  (by which  $x_1, \dots, x_n$  appear first and after them  $y_1$  and then  $y'_1$ ), this implies that  $\text{DD}_{T_1, \pi}(\alpha) = n + 1$  while  $\text{DD}_{T_2, \pi}(\alpha) = n + 2$ , and so  $\text{DD}_{T_2, \pi}(\alpha) - \text{DD}_{T_1, \pi}(\alpha) = 1$ , as claimed in the first item of the lemma.

Turning to Item 2, let  $\alpha : V \rightarrow \{0, 1\}$  be a non-satisfying assignment for  $\phi$ . This means that there exists some clause  $C$  in  $\phi$  that is not satisfied by  $\alpha$ . Hence, neither the value of  $T_1$  on  $\alpha$  nor the value of  $T_2$  on  $\alpha$  can be determined before at least some non-empty prefix of the variables  $v_1, \dots, v_k$  (and possibly  $y_0$ ) is determined. There are now two cases. If, for some  $\ell \leq k$ , the path determined by  $\alpha(v_1), \dots, \alpha(v_\ell)$  reaches a node such that all clauses in the subtree rooted at that node are satisfied by  $\alpha$ , then the output value of both  $T_1$  and  $T_2$  is determined (since  $y_1$  and  $y'_1$  were already determined). In the other case,  $\alpha(v_1), \dots, \alpha(v_k)$  reaches a clause that is not satisfied by  $\alpha$ . This means that the computation path (in both trees) ends at a  $G_1(y_0)$  gadget. This in turn implies that for both trees, the output value of the tree is determined by  $y_0$ , which is the last variable in the order  $\pi$ . Therefore, in both cases  $\text{DD}_{T_1, \pi}(\alpha) = \text{DD}_{T_2, \pi}(\alpha)$  (though the decision depth may vary according to  $\alpha$ ).  $\square$

### 3.3 A Note on the cost measure

Recall that according to our definition of the decision depth, we “pay” for every variable in the order determined by  $\pi$ , until the value of the function can be determined. An alternative definition would allow “skipping” variables that do not influence the value of the function given the assignment to all variables already read (indeed as defined in [6]). Namely, for a function  $f$  and ordering  $\pi$  of the variables  $V$  of  $f$ , a variable  $v \in V$  can be skipped given an assignment  $\alpha'$  to a subset  $V' \subset V$  of variables that appear before  $v$  in the ordering  $\pi$ , if and only if for every assignment to  $V \setminus \{V' \cup \{v\}\}$  that extends  $\alpha'$ , the value of  $f$  is the same when  $v = 1$  and when  $v = 0$ . We denote the analogues of DD and EDD according to this cost measure by  $\text{DD}'$  and  $\text{EDD}'$ , respectively, and claim that  $\text{EDD}'$  is computationally equivalent to  $\#P$ .

Using a reduction as in Subsection 3.1, one can show that  $\text{EDD}'$  reduces to  $\#P$  (where DD is replaced by  $\text{DD}'$  in the definition of the relation  $R$ ). The only thing that needs to be verified is that  $\text{DD}'_{T, \pi}(\alpha)$  is computable in polynomial time (implying that the relation is recognizable in polynomial time). This holds

because deciding whether a variable can be skipped given an assignment to a subset of the variables translates to checking whether two decision trees are equivalent, which can be done in time polynomial in the sizes of the decision trees [7].

In order to prove that #SAT (and hence #P) reduces to EDD', we slightly modify our construction of the trees  $T_1$  and  $T_2$ . To see why this is necessary, consider an assignment  $\alpha$  that does not satisfy  $\phi$ . Our reduction from #SAT to EDD used the fact that  $DD_{T_1, \pi}(\alpha) = DD_{T_2, \pi}(\alpha)$ . However, for the new cost measure the corresponding equality does not hold. The reason is that since no node in  $T_1$  is associated with  $y'_1$ , this variable can be skipped when evaluating  $T_1(\alpha)$  while it must be read when evaluating  $T_2(\alpha)$ . In order to address this issue we modify  $T_1$  and  $T_2$  so that each  $G_1(y_0)$  gadget is replaced by a  $G_2(y'_1, y_0)$  gadget. Let the modified trees be denoted  $T'_1$  and  $T'_2$  respectively. Clearly now  $T'_1$  depends on  $y'_1$ , and  $y'_1$  cannot be skipped when computing  $T'_1(\alpha)$  for  $\alpha$  that does not satisfy  $\phi$ .

It remains to verify that this modification suffices to prove a lemma analogous to Lemma 3. Namely, that for every  $\alpha : V \rightarrow \{0, 1\}$ , if  $\alpha$  satisfies  $\phi$  then  $DD'_{T'_2, \pi}(\alpha) - DD'_{T'_1, \pi}(\alpha) = 1$  while if  $\alpha$  does not satisfy  $\phi$  then  $DD'_{T'_2, \pi}(\alpha) - DD'_{T'_1, \pi}(\alpha) = 0$  (from which the proof that #SAT reduces to EDD' follows). We first observe that a variable  $x_i$  can be skipped given the restriction of  $\alpha$  to a subset  $X_i \subset \{x_1, \dots, x_{i-1}\}$  if and only if all clauses in which  $x_i$  or  $\bar{x}_i$  appear are already satisfied given the restriction of  $\alpha$  to  $X_i$ . This observation holds for computations on both  $T'_1$  and  $T'_2$ . The claim that  $DD'_{T'_2, \pi}(\alpha) - DD'_{T'_1, \pi}(\alpha) = 1$  when  $\alpha$  satisfies  $\phi$  now follows as in the proof of Lemma 3. When  $\alpha$  does not satisfy  $\phi$  then both  $y_1$  and  $y'_1$  must be read in the computation on both trees (unless all clauses are unsatisfied by  $\alpha$  in which case for both trees  $y_1$  can be skipped and  $y'_1$  must be read). It is easy to verify that from this point on, every variable that can be skipped in the computation on one tree can be skipped in the computation on the other tree, and so  $DD'_{T'_2, \pi}(\alpha) - DD'_{T'_1, \pi}(\alpha) = 0$  in this case, as claimed.

## References

- [1] S. Babu, R. Motwani, K. Munagala, I. Nishizawa, and J. Widom. Adaptive ordering of pipelines stream filters. In *Proceedings of the ACM International Conference on Management of Data*, pages 407–418, 2004.
- [2] M. Charikar, R. Fagin, V. Guruswami, J. Kleinberg, P. Raghavan, and A. Sahai. Query strategies for priced information. *Journal of Computer and Systems Sciences*, 64(4):785–819, 2002.
- [3] F. Cicalese and E.S. Laber. A new strategy for querying priced information.

- In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing*, pages 674–683, 2005.
- [4] O. Etzioni, S. Hanks, T. Jiang, R.M. Karp, O. Madani, and O. Wartz. Efficient information gathering on the internet. In *Proceedings of the 37th Annual Symposium on Foundations of Computer Science*, pages 234–243, 1996.
- [5] O. Etzioni, S. Hanks, T. Jiang, and O. Madani. Optimal information gathering on the internet with time and cost constraints. *SIAM Journal on Computing*, 29(3):1596–1620, 2000.
- [6] H. Kaplan, E. Kushilevitz, and Y. Mansour. Learning with attribute costs. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing*, pages 356–365, 2005.
- [7] H. Zantema. Decision trees: Equivalence and propositional operations. In *Proceedings of the 10th Netherlands/Belgium Conference on Artificial Intelligence (NAIC'98)*, pages 157–166, 1998. Extended version appeared as report UU-CS1998–14, Utrecht University.