Principles of Optimal Probabilistic Decision Tree Construction

Māris Ozols, Laura Mančinska, Ilze Dzelme-Bērziņa[1], Rubens Agadžanjans[1], and Ansis Rosmanis
Institute of Mathematics and Computer Science,
University of Latvia, Raiņa bulv. 29, Rīga, LV-1459, Latvia.
marozols@yahoo.com, violeta@navigator.lv, ilze.dzelme@sets.lv,
ruben.agadzanyan@gmail.com, ansis.rosmanis@gmail.com

***Abstract*— Probabilistic (or randomized) decision trees can be used to compute Boolean functions. We consider two types of probabilistic decision trees - one has a certain probability to give correct answer (but can also give nothing at all) and is not allowed to give wrong answers, but other always manages to give correct answer, but may require more computation. We provide a method, which can be used to construct the optimal probabilistic decision tree for a given Boolean function. The proposed method is based on optimization. To reduce the number of parameters in optimization problem, we take into account the symmetries of given function. As an example we consider a very symmetric Boolean function ("Fano plane function") with 7 arguments and construct the optimal decision tree for it using our method. The first kind of tree will guess the value of this function with probability 17/28 in the worst case, but the second will ask in average about 5 questions even if always the worst-case input is supplied.**

***Index Terms*— Probabilistic decision tree, computation of Boolean functions, symmetry, optimization, query algorithm.**

I. Introduction

The most simple model how to compute Boolean functions ($f : \{0,1\}^n \to \{0,1\}$) is decision tree model [2]. In this model the Boolean function is known, but arguments are kept in "black box". The aim is to compute the value of function by querying as few arguments as possible.

A *deterministic decision tree* is a rooted ordered binary tree T. Each internal node of T is labeled with a variable x_i and each leaf is labeled with a value 0 or 1. For given input $x \in \{0,1\}^n$ the evaluation starts at the root. If the current node is a leaf then the evaluation stops. Otherwise the variable x_i that labels the current node is queried. If $x_i = 0$, then left subtree will be recursively evaluated, if $x_i = 1$ then the right one. The output of the tree is the value (0 or 1) of the leaf that is eventually reached. A deterministic decision tree computes f if its output equals $f(x)$, for every $x \in \{0,1\}^n$. The complexity of the decision tree is its depth, i.e., the number of queries made on the worst-case input.

Definition 1: Boolean function f has *degenerated argument* x_i, if whatever values other arguments take, the value of f does not depend on x_i. Such functions we call *degenerated.*

Fact 1: Complexity of deterministic decision tree which computes function f coincides with the number of non-degenerated arguments of function f.

[1]This research was supported by European Social Fund.

It is also possible to use probabilistic (randomized) decision trees for Boolean function computing. *Probabilistic decision tree* is a tree with two types of internal nodes: query nodes and coin-flipping nodes. A coin-flipping node may have arbitrary finite number of subtrees and each subtree has a certain probability associated with it. When probabilistic decision tree is evaluated, at coin-flipping nodes it randomly chooses one of subtrees with appropriate probability. When a leaf is reached, the tree must output the answer. But we do not allow to give wrong answers. Thus there are two types of probabilistic decision trees possible.

Type 1: Which never asks more than some fixed number of questions. In case of bad luck it is allowed to say: "Sorry, I don't know".

Type 2: Which must always output the value of function, but in case of bad luck it may even require to ask all arguments of function.
The complexity of *Type 2* trees is the average number of queries required to compute the worst-case input. It is known that there are functions for which *Type 2* trees require less queries than deterministic trees [4], [3]. But we do not investigate the upper or lower bounds - we provide a method how to construct optimal tree for given function.

In this paper we will mainly discuss *Type 1* trees, but our method can be applied to *Type 2* trees as well. For *Type 1* trees with fixed number of queries one needs to know how often it will say "Sorry..." – let "?" denote this output.

Definition 2: $T(x)$ denotes the *outcome* of probabilistic decision tree T given input x. Depending on x, it is either probability distribution over $\{0,?\}$ or $\{1,?\}$.

Definition 3: The probability of correct answer of decision tree T for given function f and input x is denoted by $T_f(x) = P\big(T(x) = f(x)\big) = 1 - P\big(T(x) = ?\big)$.

Definition 4: We say that probabilistic decision tree T_f *computes* function f if $T_f(x) > 1/2$ for all inputs $x \in \{0,1\}^n$.

II. Primitive Reduction

One decision tree with slight modifications can be used to compute a vast range of Boolean functions. For example, if we know the best tree for some function f, we can build one also for $\neg f$, where "$\neg$" denotes negation. Even more - if we know optimal tree for $f(x_1, x_2, x_3)$, we can easily modify it and compute let's say $f(x_3, x_2, x_1)$ or $f(\neg x_1, x_2, x_3)$.

Definition 5: There are three *trivial reductions:*

1) *bit swapping:*
$$\mathrm{SWAP}_{ij}(x_1, x_2, \ldots, x_i, \ldots, x_j, \ldots, x_n) = (x_1, x_2, \ldots, x_j, \ldots, x_i, \ldots, x_n) \quad (1)$$
2) *bit inversion:*
$$\mathrm{NOT}_i(x_1, x_2, \ldots, x_i, \ldots, x_n) = (x_1, x_2, \ldots, \neg x_i, \ldots, x_n) \quad (2)$$
3) *result inversion:*
$$\mathrm{NOT} f(x) = \neg f(x) \quad (3)$$

Definition 6: Primitive reduction is any sequence of trivial reductions - bit swapping (1) and bit inversion (2), which can be followed by result inversion (3).

Example 1: Function $f(x_1, x_2) = x_1 \& x_2$ can be primitively reduced to $g(x_1, x_2) = x_2 \vee x_1$, because
$$f = \mathrm{NOT} \circ g \circ \mathrm{SWAP}_{12} \circ \mathrm{NOT}_2 \circ \mathrm{NOT}_1$$

Definition 7: Distance of Boolean function f is
$$D_f = \Big| \big|\{x \mid f(x) = 0\}\big| - \big|\{x \mid f(x) = 1\}\big| \Big| \quad (4)$$

It is easy to see that primitive reduction preserves distance (because each trivial reduction does). If we consider two-argument Boolean functions, we see that the only possible distances are 0, 2, and 4. It means that among 16 different two-argument Boolean functions there are at least three different up to primitive reduction. In fact there are four:

$f(x_1, x_2) = 0$	$D_f = 4$	(degenerated)
$f(x_1, x_2) = x_1$	$D_f = 0$	(degenerated)
$f(x_1, x_2) = x_1 \vee x_2$	$D_f = 2$	
$f(x_1, x_2) = x_1 \oplus x_2$	$D_f = 0$	

(here "$\oplus$" denotes logical XOR) and only two of them are non-degenerated. If we neglect them, we could say that there are only two essentially different two-argument Boolean functions.

Now we will show, how primitive reduction can be used to transform decision trees. It allows to obtain optimal decision tree for one function, if optimal tree for some other function is known.

Theorem 1: Let f and g be Boolean functions and π be primitive reduction such that $f = \pi(g)$. If T_g is optimal decision tree for g then $T_f = \pi(T_g)$ is optimal for f.

Proof: To understand how $\pi(T_g)$ is constructed, it suffices to consider only trivial reductions:

1) if $\pi(g) = g \circ \mathrm{SWAP}_{ij}$ then $\pi(T_g)$ is obtained by relabeling the query nodes of T_g, namely x_i should be replaced by x_j and vice versa,
2) if $\pi(g) = g \circ \mathrm{NOT}_i$ then $\pi(T_g)$ is obtained by exchanging subtrees of T_g corresponding to outcomes 0 and 1 for all nodes where x_i is queried,
3) if $\pi(g) = \mathrm{NOT} \circ g$ then $\pi(T_g)$ is obtained by exchanging 0 and 1 in decision tree output (in T_g leaves).

In all three cases $T_f = \pi(T_g)$ will be optimal for f. Otherwise there would be some T'_f which is better than T_f for f. But then $\pi^{-1}(T'_f)$ would be better than T_g for g which is a contradiction (note that $\pi^{-1} = \pi$ if π is trivial). ■

III. Symmetries of Hypergraphs

For the sake of simplicity from now on we will consider only monotone Boolean functions. This restriction is arbitrary and our method can be generalized for non-monotone Boolean functions as well, but then we would have to introduce more complicated notions of symmetry. The brightest side of this restriction is that actually we will be dealing with a larger class, i.e. the class of those Boolean functions which are primitively reducible to monotone ones.

Definition 8: Boolean function f is *monotone* IFF
$$\forall x, y \in \{0,1\}^n : (x \Rightarrow y) \Rightarrow \big(f(x) \Rightarrow f(y)\big) \quad (5)$$
where $x \Rightarrow y$ stands for bitwise implication.

Each monotone Boolean function can be written in disjunctive normal form (DNF), i.e. as a disjunction of terms where each term is a conjunction of variables. Note that DNF of monotone Boolean function does not contain negations.

Example 2: A monotone Boolean function in DNF:
$$\begin{aligned} f(x_1, x_2, x_3, x_4, x_5) = &(x_1 \& x_2 \& x_3 \& x_4) \\ &\vee (x_2 \& x_5) \\ &\vee (x_3 \& x_5) \end{aligned}$$

DNF of monotone Boolean function can be considered as a finite incidence structure where terms stand for lines and variables stand for points.

Definition 9: Finite incidence structure is a formal triple $(\mathcal{V}, \mathcal{B}, \mathcal{I})$, where $\mathcal{V}$ is a finite set of *points*, $\mathcal{B}$ is a finite set of *lines* and $\mathcal{I}$ is *incidence relation* between them: $(v, b) \in \mathcal{I}$ means point v is on line b.

Each incidence structure can be thought as a *hypergraph*, i.e. a graph with generalized edges (called *hyperedges*) connecting more than two vertices. For given incidence structure $(\mathcal{V}, \mathcal{B}, \mathcal{I})$ one can construct a hypergraph $G = (V, B)$ by setting $V = \mathcal{V}$ and $B = \{\{v \mid (v, b) \in \mathcal{I}\} \mid b \in \mathcal{B}\}$ – each hyperedge corresponds to a set of vertices.

Note that in DNF one term cannot be a subterm of other, because in that case the largest term can be eliminated using absorption rule. It means that in corresponding hypergraph one hyperedge cannot be a subset of other.

Definition 10: Hypergraph corresponding to DNF of Boolean function f is denoted by G_f.

Example 3: See Fig. 1 for hypergraph example.

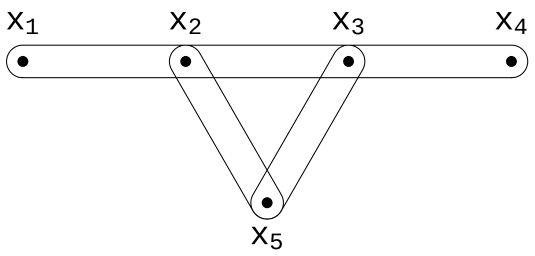

Fig. 1. A hypergraph for function (6) from previous example

It is not hard to notice that hypergraph in Fig. 1 possesses some symmetry. Let us introduce appropriate definitions.

Definition 11: Two hypergraphs $G_1 = (V_1, B_1)$ and $G_2 = (V_2, B_2)$ are said to be *isomorphic* ($G_1 \cong G_2$) IFF there exists a bijection $\varphi : V_1 \to V_2$ such that a pair of vertices $u, v \in V_1$ is adjacent in G_1 IFF $\varphi(u)$ and $\varphi(v)$ are adjacent in G_2.

Definition 12: A bijection $\varphi : V \to V$ is an *automorphism* of $G = (V, B)$ IFF it is an isomorphism between G and itself.

Example 4: Hypergraph in Fig. 1 possesses 4 automorphisms: trivial, $(x_1 \leftrightarrow x_4)$, $(x_2 \leftrightarrow x_3)$, $(x_1 \leftrightarrow x_4, x_2 \leftrightarrow x_3)$.

Fact 2: Automorphisms of G_f form its *automorphism group* $Aut(G_f)$.

Each automorphism $\varphi \in Aut(G_f)$ can be thought as a permutation of vertices of G_f that preserves hyperedges. It means that DNF of corresponding Boolean function is also preserved and thereby the function itself.

Corollary 1: Each automorphism $\varphi \in Aut(G_f)$ preserves the corresponding Boolean function f:

$$\forall x \in \{0,1\}^n : f(\varphi x) = f(x_{\varphi(1)}, x_{\varphi(2)}, \ldots, x_{\varphi(n)}) = f(x)$$

(here φx is a permutation of arguments).

This allows us to introduce the notion of symmetric vertices.

Definition 13: Vertices u and v of hypergraph G_f are *symmetric* ($u \overset{f}{\sim} v$) IFF there exists an automorphism $\varphi \in Aut(G_f)$ for which $\varphi(u) = v$.

Fact 3: Relation "$\overset{f}{\sim}$" divides the set of vertices of hypergraph G_f into equivalence classes. This applies also to arguments of corresponding function f.

Example 5: There are 3 equivalence classes of vertices of hypergraph in Fig. 1: $\{x_1, x_4\}$, $\{x_2, x_3\}$, $\{x_5\}$. The same stands for arguments of function (6).

If we consider an arbitrary input (sequence of variables) for given function f, we can investigate to which other inputs it can be mapped via an automorphism $\varphi \in Aut(G_f)$. Hence we can introduce the notion of symmetric inputs for given Boolean function f.

Definition 14: Inputs $x, y \in \{0,1\}^n$ of Boolean function f are said to be *symmetric* ($x \overset{f}{\approx} y$) IFF there exists an automorphism $\varphi \in Aut(G_f)$ such that $\varphi x = y$.

Example 6: A class of symmetric inputs for function (6) is for example:

$$\{(0,0,1,1,1), (0,1,0,1,1), (1,0,1,0,1), (1,1,0,0,1)\}$$

Fact 4: Relation "$\overset{f}{\approx}$" divides all inputs of f into equivalence classes.

Definition 15: $[x]_f = \{b \mid b \overset{f}{\approx} x\}$ is *equivalence class* of input x of function f.

If we take an arbitrary input x and repeatedly apply automorphisms from $Aut(G_f)$ to it, we will span the entire class $[x]_f$. Now we have introduced all the required notions to proceed to decision tree construction.

IV. Principles of Probabilistic Decision Tree Construction

Principle 1: If $x \overset{f}{\approx} y$ are inputs of Boolean function f, then probabilistic decision tree T_f should give correct answer with equal probabilities for both inputs.

Theorem 2: If probabilistic decision tree T_f does not satisfy *Principle 1*, it can be modified to satisfy it without decreasing the worst-case probability for any of the equivalence classes.

Proof: We will give an explicit description how to construct improved decision tree T'_f. Let $Aut(G_f) = \{\varphi_1, \varphi_2, \ldots, \varphi_N\}$ and $N = |Aut(G_f)|$ be the size of $Aut(G_f)$. Let the root of

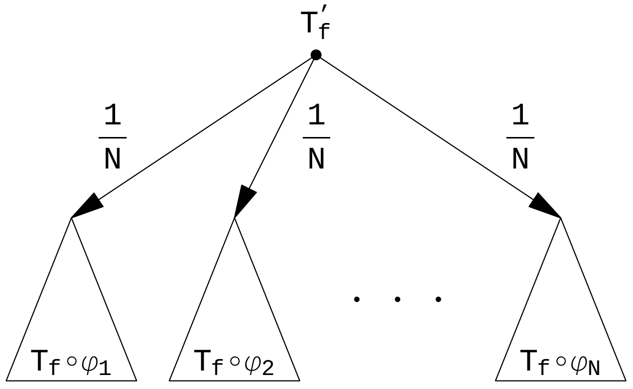

Fig. 2. Improved decision tree T'_f

new decision tree T'_f be a coin-flipping node with N uniformly distributed outcomes and let us assign to i-th outcome a copy of initial tree T_f with inputs permuted by automorphism φ_i (this subtree will be denoted by $T_f \circ \varphi_i$). Resulting tree is shown in Fig. 2. Because all symmetric inputs are treated equally, it is clear that T'_f satisfies

$$\forall x, y \in \{0,1\}^n : (x \overset{f}{\approx} y) \Rightarrow \left(T'_f(x) = T'_f(y)\right) \qquad (6)$$

and therefore *Principle 1*. Now we only have to prove, that we have not decreased the worst-case probability for any of the equivalence classes. Let $[e]_f$ be an equivalence class, where all members have the same probability for T_f, namely $T_f([e]_f)$. Due to symmetry it is clear that they will have the same probability also for T'_f, i.e. $T'_f([e]_f) = T_f([e]_f)$. But if there is a class $[u]_f$ where all members does not have equal probabilities for T_f and m is the worst-case input for this class, then

$$\begin{aligned} \forall x \in [u]_f : T'_f(x) =& \frac{1}{N}\sum_{i=1}^{N} T_f(\varphi_i(x)) \qquad (7) \\ >& \frac{1}{N}\sum_{i=1}^{N} T_f(m) = T_f(m) \end{aligned}$$

because average value of all probabilities will be greater than the smallest one. ∎

Corollary 2: If T_f satisfies *Principle 1* and x is the worst-case input, then all class $[x]_f$ consists of worst-case inputs.

Principle 2: If $x_i \overset{f}{\sim} x_j$ are arguments of Boolean function f, then probabilistic decision tree T_f in first query should ask both arguments with equal probabilities.

From programmer's point of view each decision tree T_f for f can be considered as a function which does the following: generates a random number R, then issues a query for an argument x_i where i depends on R, and finally calls a subroutine which is another probabilistic decision tree calculating (n-1)-argument function $f'(x_1,,\ldots,x_{i-1},x_{i+1},\ldots,x_n) = f(x_1,\ldots,x_{i-1},q,x_{i+1},\ldots,x_n)$ where q is the result of query. We will call f' a subfunction.

Definition 16: Boolean functions f and g are said to be *isomorphic* ($f \cong g$) IFF they have isomorphic hypergraphs: $G_f \cong G_g$ (see *Def. 11*).

Definition 17: Decision trees T_f and T_g are said to be *isomorphic* ($T_f \cong T_g$) IFF there exists a permutation of arguments φ such that $\varphi(T_f) = T_g$.

Note that the notion of isomorphic functions and isomorphic trees can be extended using primitive reduction (i.e., functions or trees are isomorphic IFF there exists a primitive reduction from one to other).

Principle 3: Probabilistic decision tree for function f should compute isomorphic subfunctions of f with isomorphic subtrees.

We want to emphasize that here with isomorphic subfunctions we also mean that the same number of questions has been asked. It is possible that we arrive to the same function, but we have asked different number of questions. Then almost definitely will have to use different strategies for evaluating these functions.

Although *Principle 3* looks very reasonable, we were not able to prove the following conjecture.

Conjecture 1: If probabilistic decision tree does not satisfy *Principle 3*, it can be modified to satisfy it without decreasing the worst-case probability.

Difficulties in proving *Conjecture 1* arises due to the fact that non-isomorphic subtrees for isomorphic subfunctions could possibly lead to overall symmetry of entire decision tree. It could happen that attempts to improve the non-symmetric subtrees using symmetrization techniques similar to one used in proof of *Theorem 2* could disturb the overall symmetry and therefore decrease the probability of correct answer for entire tree. But this is not the case for the next theorem.

Theorem 3: If probabilistic decision tree T_f satisfies *Principle 3* but not *Principle 2*, it can be improved without decreasing the worst-case probability.

Proof: Let M be an arbitrary coin-flipping node of T_f where *Principle 2* is not satisfied. Let function computed by subtree with root M be f'. Node M has sets of isomorphic subtrees $S_i = \{T_1^i,\ldots,T_{k_i}^i\}$ attached to it. It means root nodes of subtrees $\{T_1^i,\ldots,T_{k_i}^i\}$ are query nodes asking symmetric arguments for function f'. If M is the root node (and therefore $f' = f$), we can safely average the probabilities of evaluating $\{T_1^i,\ldots,T_{k_i}^i\}$ for each subset S_i, without decreasing the worst-case probability for T_f. After performing this step *Principle 2* will be satisfied for M and *Principle 1* will be satisfied for the whole T_f. Now we can repeat this procedure to all nodes M of the next level of T_f and so on. In this manner we will make all nodes of T_f satisfy *Principle 2* and the whole tree will satisfy *Principle 1*. ■

Corollary 3: If probabilistic decision tree satisfies *Principle 3* and *Principle 2*, it also satisfies *Principle 1*.

V. Demonstration of Method

A. Fano Plane Function

Lets now consider a particular function, which we will use to demonstrate our method. We will call it *Fano plane function*:

$$f_F(x_1,x_2,x_3,x_4,x_5,x_6,x_7) = 1 \text{ if } \begin{cases} x_1 = x_2 = x_4 = 1 \text{ or} \\ x_2 = x_3 = x_5 = 1 \text{ or} \\ x_3 = x_4 = x_6 = 1 \text{ or} \\ x_4 = x_5 = x_7 = 1 \text{ or} \\ x_5 = x_6 = x_1 = 1 \text{ or} \\ x_6 = x_7 = x_2 = 1 \text{ or} \\ x_7 = x_1 = x_3 = 1 \end{cases} \tag{8}$$

$$f_F(x_1,x_2,x_3,x_4,x_5,x_6,x_7) = 0 \text{ if } \begin{cases} x_1 = x_2 = x_4 = 0 \text{ or} \\ x_2 = x_3 = x_5 = 0 \text{ or} \\ x_3 = x_4 = x_6 = 0 \text{ or} \\ x_4 = x_5 = x_7 = 0 \text{ or} \\ x_5 = x_6 = x_1 = 0 \text{ or} \\ x_6 = x_7 = x_2 = 0 \text{ or} \\ x_7 = x_1 = x_3 = 0 \end{cases} \tag{9}$$

This function may look similar to definition of Hamming codes but in fact they have almost nothing in common. We will show that conditions in equations (8) and (9) does not contradict and function f_F is defined for all inputs. In order to do this, we have to introduce the notion of *projective plane*.

Definition 18: A *projective plane* is a finite set of *points* and a set of subsets of point set (called *lines*) which satisfies the following conditions:

1) every two points lie on exactly one line;
2) every two lines intersect in exactly one point;
3) there are four points with no three collinear.

The projective plane of order n has n^2+n+1 points and n^2+n+1 lines. Each point lies on exactly $n+1$ lines and every line contains exactly $n+1$ points. Fano plane is projective plane of order $n = 2$. It is possible to assign arguments of function f_F to points of Fano plane (as depicted in Fig. 3) in such manner, that each triplet of arguments from conditions (8) and (9) lies on a line (circle x_2, x_5, x_3 also will be called "line").

Lemma 1: Conditions (8) and (9) does not contradict and function f_F is defined for all inputs $\{0,1\}^7$.

Proof: We can reformulate conditions (8) and (9) for Fano plane. Let white dot "○" denote 0 and black dot "●" denote 1 in Fano plane. We will call line in Fano plane white (black) if all three points on it are white (black). Then (8) says: "$f_F = 1$ if there is a black line in Fano plane" and (9) says: "$f_F = 0$ if there is a white line in Fano plane". It is clear that these two cannot contradict, because each two lines in projective plane have a common point and this point is either white or black. Now we have to show that f_F is defined for all inputs. In Fano plane terms it means "if points of Fano plane are colored with

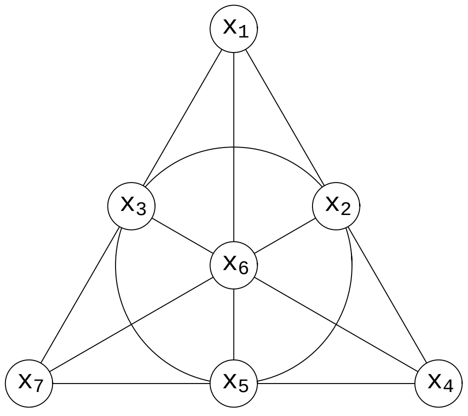

Fig. 3. Correspondence between points of Fano plane and arguments of function f_F from (8) and (9)

two colors (lets say black and white), there will always be at least one line with all three points of same color". If we note the symmetry of colors from (8) and (9) then there are only five non-isomorphic colorings of Fano plane (see Fig. 4). It is easy to verify that in each case there is a line of same color (e.g., the bottom line). ■

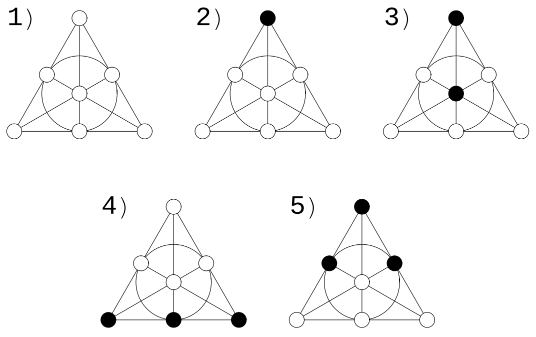

Fig. 4. Non-isomorphic colorings of Fano plane

We can use (8) to write down the DNF of function f_F:

$$\begin{aligned} f_F(x_1,x_2,x_3,x_4,x_5,x_6,x_7) \quad = \quad & (x_1 \& x_2 \& x_4) \vee \\ & (x_2 \& x_3 \& x_5) \vee \\ & (x_3 \& x_4 \& x_6) \vee \\ & (x_4 \& x_5 \& x_7) \vee \\ & (x_5 \& x_6 \& x_1) \vee \\ & (x_6 \& x_7 \& x_2) \vee \\ & (x_7 \& x_1 \& x_3) \qquad (10) \end{aligned}$$

It says: "$f_F(x) = 1$ if all three points on first line are black or all three points on second line are black… etc." From (10) one can see that Fano plane is indeed useful for visualizing f_F, because the incidence structure corresponding to DNF of f_F coincides with the projective plane of order 2.

It is easy to see that non-deterministic complexity of f_F is 3, because for all inputs one needs to know only three arguments to show what value the function admits.

B. General Decision Tree for Fano Plane Function

Now we will use the results from *Sect. IV* to construct a probabilistic decision tree for f_F (according to *Theorem 3*, it must satisfy *Principles 2* and *3*). According to *Principle 3*, we have to consider all subfunctions of f_F (cases when some of arguments are known) and divide all unknown arguments into equivalence classes (they are denoted with gray polygons in Fig. 5). According to *Principle 2* decision tree has to ask with equal probabilities for all arguments in the same equivalence class. Thus we only have to assign some unknown probabilities to equivalence classes (denoted with p_1, p_2, … in Fig. 5). And for each equivalence class we have to consider all possible outcomes of queries asking arguments from this class. In this manner we have to proceed till some depth of our decision tree (i.e. number of arguments asked) – in our case it is five. At the end when we have left only one question, there is no sense to ask for such arguments which can give no information about the value of f_F (see three rightmost cases in last level of Fig. 5).

We want to emphasize that due to symmetry of function f_F colors used in Fig. 5 are relative (they can be exchanged), because $f_F(\neg x) = \neg f_F(x)$. You should pay attention only to the fact - are the points of same color or not. And please note that every Fano plane in Fig. 5 corresponds to a whole class of isomorphic subfunctions.

C. Calculating Probabilities

According to *Corollary 3*, our tree will satisfy *Principle 1.* It means, in order to find the worst-case probability, we have to consider only five non-isomorphic inputs depicted in Fig. 4. We used *Mathematica 5.2* to calculate the probability of correct answer for these inputs. We got five multilinear polynomials of unknown probabilities:

$$P_1 = 1 \tag{11}$$

$$P_2 = 1 - \frac{1}{7}p_1(1-p_3) \tag{12}$$

$$P_3 = \frac{5}{7} + \frac{1}{21}\big(2p_2 - 2p_1(1-p_3) - p_4(p_1+2p_2)\big) \tag{13}$$

$$P_4 = \frac{4}{7} + \frac{1}{7}\big(2p_2 - p_4(p_1+2p_2)\big) \tag{14}$$

$$P_5 = \frac{13}{28} + \frac{1}{28}\big(4p_2(1-p_4) - p_1(1-p_3+2p_4)\big) \tag{15}$$

Note that p_5 does not appear - it does not affect the result.

D. Optimizing Worst-case Probability

The worst-case probability for our decision tree is the minimum probability of correct answer

$$P_w(p_1,p_2,p_3,p_4) = \min\{P_1,P_2,P_3,P_4,P_5\} \tag{16}$$

We have to maximize the worst-case probability to get the best possible tree:

$$P = \max_{0\le p_i\le 1} P_w(p_1,p_2,p_3,p_4) = \frac{17}{28} \tag{17}$$

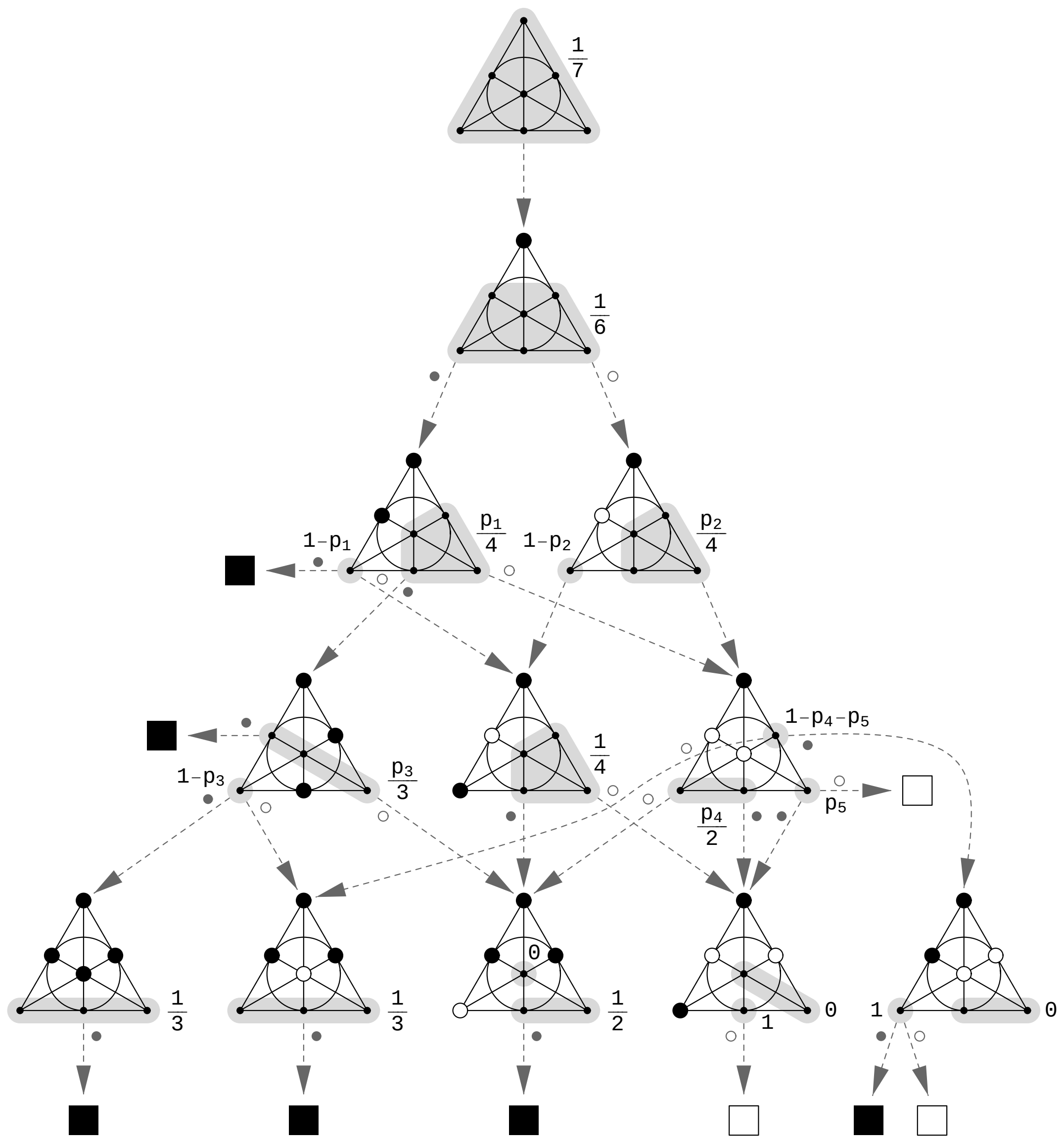

Fig. 5. General probabilistic decision tree for f_F which satisfies *Principles 2* and *3*

which holds for all p_1, p_2, p_3, p_4 satisfying:

$$4p_2 + p_1p_3 = 4 + p_1 + 2p_1p_4 + 4p_2p_4 \tag{18}$$

But we must also take into account that $p_4 + p_5 \leq 1$. Then (18) reduces to:

$$p_2 = 1 \tag{19}$$

$$p_4 = 0 \tag{20}$$

$$0 \leq p_5 \leq 1 \tag{21}$$

$$p_1p_3 = p_1 \tag{22}$$

where (22) means $p_1 = 0$ and $0 \leq p_3 \leq 1$ or $p_3 = 1$ and $0 \leq p_1 \leq 1$. We can see that worst input class is case 5 from Fig. 4. Members of this class will have probability $17/28$ for all probabilities satisfying our solution. In general there are several decision trees with the same worst-case probability, but they may have different probabilities of correct answer for classes 2, 3, and 4.

We constructed the tree depicted in Fig. 5 for the case of five questions, because for less than five questions it is not possible to have the worst-case probability greater than $1/2$ (using the same method we showed that for four questions it cannot exceed $5/14$).

To calculate the average number of questions for optimal *Type 2* tree for Fano plane function, we constructed the tree depicted in Fig. 5 till the last level (where all arguments are known). Then we minimized the maximal number of questions required to get the answer (cases shown in Fig. 4). We were not able to find global minimum analytically, but with numerical optimization we were able to obtain solution with average number of questions equal to 5.107. This is considerably less than 7 - the number of questions required for deterministic tree in worst case.

VI. Conclusions and Future Work

It is algorithmically possible to construct the best or "almost best" (using numerical optimization) possible probabilistic decision tree for given Boolean function. This can be used in compilers to perform randomized optimizations of programs. We can consider a program which needs to evaluate some compound "IF" condition frequently. If it contains a lot of time-costly subconditions, it is important to find the value of "IF" by evaluating as few subconditions as possible. Application of our method to *Type 2* decision trees could decrease the average number of subconditions evaluated.

Another direction where to extend our results is quantum query algorithms [1], because *Type 1* tree model is very close to the quantum query model. It is well known that each *Type 1* decision tree can be effectively simulated by quantum query algorithm [2]. Quantum computation heavily exploits unitary matrices. It is well known that each unitary matrix can be decomposed into simple 2×2 Hadamard matrices and CNOT gates [5]. This allows to specify $n \times n$ unitary matrix in general form. If we could transfer principles (notions of symmetric arguments and symmetric subtrees) to unitary matrices, it could be possible to apply similar optimization technique as in probabilistic case.

References

[1] Ambainis, A., Quantum lower bounds by quantum arguments. quant-ph/0002066 (2000)

[2] Buhrman, H., de Wolf, R.: Complexity measures and decision tree complexity: a survey. (2002)

[3] Saks, M., Wigderson, A.: Probabilistic Boolean decision trees and the complexity of evaluating game trees. In Proceedings of 27th FOCS (1986) 29–38

[4] Snir, M.: Lower bounds for probabilistic linear decision trees. Theoretical Computer Science **38** (1985) 69–82

[5] Vartiainen, J.J., Möttönen, M., Salomaa, M.M.: Efficient decomposition of quantum gates. Phys. Rev. Lett. (2004) vol. 92, 177902