Journal of Universal Computer Science, vol. 17, no. 6 (2011), 859-873
submitted: 30/7/10, accepted: 17/2/11, appeared: 28/3/11 © J.UCS

Nondeterministic Query Algorithms

Alina Vasilieva
(Faculty of Computing, University of Latvia, Riga, Latvia
Alina.Vasilieva@gmail.com)

Rūsiņš Freivalds
(Institute of Mathematics and Computer Science, University of Latvia
Riga, Latvia
Rusins.Freivalds@mii.lu.lv)

Abstract: Query algorithms are used to compute Boolean functions. The definition of the function is known, but input is hidden in a black box. The aim is to compute the function value using as few queries to the black box as possible. As in other computational models, different kinds of query algorithms are possible: deterministic, probabilistic, as well as nondeterministic. In this paper, we present a new alternative definition of nondeterministic query algorithms and study algorithm complexity in this model. We demonstrate the power of our model with an example of computing the Fano plane Boolean function. We show that for this function the difference between deterministic and nondeterministic query complexity is 7^N versus $O(3^N)$.

Key Words: Algorithm complexity, nondeterministic query algorithm, decision tree, Boolean function
Category: F.1.1, F.1.3, F.2.3

1 Introduction

Let $f(x_1, x_2, ..., x_n) : \{0,1\}^n \to \{0,1\}$ be a Boolean function. We consider the query model, where the definition of the function is known, but a black box contains the input $(x_1, x_2, ..., x_n)$, which can be accessed only by querying x_i values. The goal is to compute the value of the function for arbitrary input. The complexity of a query algorithm is measured by the number of questions it asks. The classical version of this model is known as *decision trees* [Buhrman, de Wolf 2002].

In this paper, we are considering nondeterministic algorithms. A nondeterministic finite automata, as introduced in [Rabin, Scott 1959], is a machine with many choices in its moves. At each stage such machine may choose one of several next internal states. Nondeterministic machine accepts a tape if there is at least one winning combination of choices of states leading to a designated final state. This is a traditional point of view on nondeterminism. In [Floyd 1967], nondeterministic algorithms are considered to be conceptual devices to simplify the design of backtracking algorithms. There is a point of view presented that algorithms are nondeterministic not in the sense of being random, but in the sense of having *free will*. The detailed definitions of nondeterministic finite automata, pushdown automata, Turing machine, and related results in complexity theory are presented and discussed in [Hopcroft, Ullman 1969].

In this paper, we are investigating the nature of the above-mentioned *nondeterministic free will*. We provide a way to measure the amount of nondeterminism in an algorithm. In the traditional nondeterministic query model, the power of nondeterminism comes at no cost. Our idea is that we must pay with additional queries for the nondeterministic help. We introduce an alternative definition of the nondeterministic query model, which incorporates behavior described above.

The main field of our research interest is quantum computing. In recent years, we have been actively studying quantum query complexity in different models - exact [Vasilieva 2009], exact with a promise [Freivalds, Iwama 2009], bounded-error [Vasilieva, et. al., 2010], postselection [Scegulnaja-Dubrovska, et. al., 2010] and also nondeterministic [Vasilieva (Dubrovska) 2007], as well as non-standard classical models of computation, for instance, non-constructive methods for finite probabilistic automata [Freivalds 2008]. The definition of the nondeterministic quantum query algorithms, as first introduced in [de Wolf 2003] and further investigated in [Vasilieva (Dubrovska) 2007], seems to us a bit counterintuitive. It was another motivation for us to introduce a new approach for nondeterminism in query algorithms. Based on our knowledge, we believe that this kind of a nondeterministic query model is going to be more natural for quantum computing. However, in this paper we limit ourselves to considering classical model only.

This paper is organized as follows. In Section 2, we provide a brief overview of the traditional query model. In Section 3, we introduce our alternative model for nondeterministic query computation. Finally, in Section 4, we demonstrate an example of computing the Fano plane Boolean function in our model.

2 Query Algorithms

In this section, we give a brief overview of the classical query algorithms and provide all necessary definitions.

Definition 1 *Boolean function is* total *if it is defined for all inputs. If for some subset of inputs function value is not defined, function is called* partial.

The classical version of the query model is known as *decision trees*, for details see the survey [Buhrman, de Wolf 2002]. The definition of the Boolean function is known to everybody, but the input $(x_1, x_2, ..., x_n)$ is hidden in a black box, and can be accessed by querying x_i values. The algorithm must be able to determine the value of a function correctly for arbitrary input. The complexity of the algorithm is measured by the number of queries on the worst-case input.

A deterministic decision tree is an ordered tree with internal nodes labeled with variables x_i, arrows exiting internal nodes labeled with possible variable values and leaves labeled with function values. Deterministic decision tree always follows the same path for each input and produces result with probability $p = 1$.

Definition 2 *[Buhrman, de Wolf 2002] The* deterministic complexity *of a function f, denoted by D(f), is the maximum number of questions that must be asked on a worst-case input by an optimal deterministic algorithm for f.*

The traditional nondeterministic decision tree differs from the deterministic one with an additional possibility that there can be more than one arrow labeled with the same value exiting each tree vertex.

Definition 3 *The nondeterministic decision tree computes Boolean function f(X), if for an arbitrary input X it is true that:*

- *if f(X)=1, then a legal path exists from the root to the leaf with result 1;*
- *if f(X)=0, then a legal path exists from the root to the leaf with result 0;*
- *there is no path from the tree root to the leaf with incorrect function value.*

3 Alternative Nondeterministic Query Model

In this paper, we propose an alternative definition of a nondeterministic query model. This model is different from the traditional one and allows to get interesting behavior. The main idea and difference is that in this variation we do not receive the power of nondeterminism free of charge, but have to spend additional queries to obtain nondeterministic help. In some sense this is a way to measure the amount of nondeterminism in an algorithm.

Suppose that we need to compute some arbitrary Boolean function *F(X)* in a nondeterministic query model [1]. Then, the first step is to define a *nondeterministic helper function* *G(X,Y)*. This function has to satisfy definite conditions, we will specify them precisely a little bit later. The second step is to design a deterministic query algorithm for the function *G(X,Y)* . Finally, the *nondeterministic query complexity* of the function *F(X)* is equal to the complexity of the deterministic query algorithm for a nondeterministic helper function *G(X,Y)*.

Now, we will give formal definitions for the computational model informally described above.

[1] Here and later on in this paper, when we are talking about *nondeterministic* model we mean our alternative definition, not the traditional one.

Definition 4 *The* nondeterministic helper function *G(X,Y) for the Boolean function F(X) is a partial Boolean function, which satisfies the following conditions:*

1. $\forall x_1, ..., x_n, \exists y_1, ..., y_k,$ *such that* $G(x_1, ..., x_n, y_1, ..., y_k) = F(x_1, ..., x_n)$;
2. $\forall x_1, ..., x_n, \neg\exists y_1, ..., y_k,$ *such that* $G(x_1, ..., x_n, y_1, ..., y_k) \neq F(x_1, ..., x_n)$.

When computing *G(X,Y)* deterministically we will get either an answer that $G(X,Y) = b$ ($b \in \{0,1\}$) or will receive indefinite answer *"don't know"*. If some Boolean value b is retrieved during calculation, it implies that $F(X) = b$.

Definition 5 *The nondeterministic* query complexity of the function $F(X)$ with the fixed helper function $G(X,Y)$ *is denoted with* $ND_G(F)$ *and is equal to the deterministic complexity of the G(X,Y):* $ND_G(F) = D(G)$.

An additional restriction on the deterministic query algorithm for the helper function *G(X,Y)* is that after computing this function deterministically we should be able to re-calculate or verify the value of *F(X)* independently, using variable values extracted from the black box during the calculation of *G(X,Y)*.

Definition 6 *The* nondeterministic query complexity of the function $F(X)$ *is denoted with* $ND(F)$ *and is equal to the minimal nondeterministic query complexity of the function* $F(X)$ *over all possible fixed helper functions* $G(X,Y)$*:* $ND(F) = \min_{G(X,Y)} ND_G(F)$.

4 Computing the Fano Plane Function

In this section, we demonstrate our alternative nondeterministic query model in action. We demonstrate that a gap between deterministic and nondeterministic query complexity for a certain Boolean function can be large. We define a function based on the Fano plane and proposed nondeterministic query model appears to be well-suited for computing this kind of function.

4.1 Definition of the Fano Plane Boolean Function

The *Fano plane* is the two-dimensional finite projective plane with the least number of points and lines [Weisstein]. This plane has seven points and seven lines with three points on every line. Fano plane has many applications including factoring integers via quadratic forms [Lehmer, D.H., Lehmer, E. 1974]. We define a 7-variable Boolean function based on the structure of the Fano plane. We label each vertex of the Fano plane with a variable number x_i. [Fig. 1] represents a variant of variables assignment and we will use this fixed definition in the sequel of this paper.

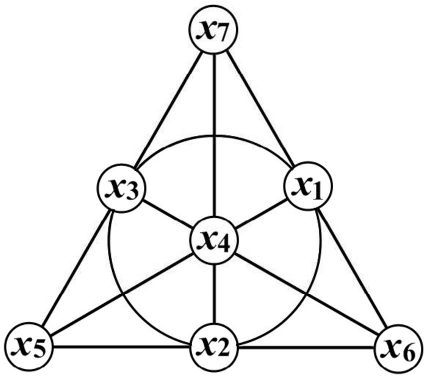

Figure 1: Fano plane with vertices labeled with function *FANO(X)* variables

Definition 7 *A line of the Fano plane with Boolean values assigned to vertices is called* constant *if all vertices in a line have the same Boolean value assigned.*

There are two important properties of the Fano plane with Boolean values assigned to vertices. For any variable values assignment ($x_i \in \{0, 1\}$) there always is a constant line. Second property is that for any variable values assignment there cannot be two constant lines assigned with the different Boolean value at the same time. These two properties allow us to define a Boolean function based on the Fano plane.

Definition 8 *Boolean function $FANO(x_1, ..., x_7)$ is defined as follows. For an arbitrary input $X = (x_1, ..., x_7)$ find a constant line in the Fano plane. Value of the $FANO(x_1, ..., x_7)$ function equals Boolean value assigned to vertices in that constant line.*

An example of *FANO(X)* function value assignment is illustrated in [Fig. 2].

The Fano plane Boolean function can be represented also with a logical formula:

$FANO(x_1, ..., x_7) = (x_1 \wedge x_2 \wedge x_3) \vee (x_5 \wedge x_3 \wedge x_7) \vee (x_7 \wedge x_1 \wedge x_6) \vee$
$\vee (x_6 \wedge x_2 \wedge x_5) \vee (x_3 \wedge x_4 \wedge x_6) \vee (x_1 \wedge x_4 \wedge x_5) \vee (x_2 \wedge x_4 \wedge x_7).$

4.2 Deterministic Complexity of the Fano Plane Boolean Function

To determine *FANO(X)* function value using the deterministic decision tree we need to query all variables.

Theorem 1 *Deterministic complexity of the Fano plane Boolean function is equal to the number of function variables: D(FANO)=7 .*

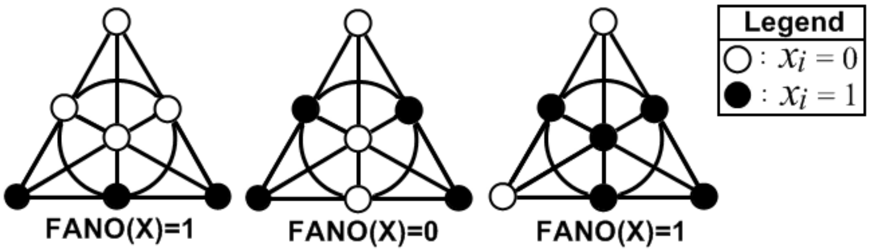

Figure 2: Illustration of *FANO(X)* Boolean function value assignment

Proof. To prove this lower bound we use a kind of *adversary* method. We view the computation as a game between arbitrary algorithm and adversary, which is playing against that algorithm. Algorithm is querying variables in order to determine function value, but adversary is providing variable values trying to use a strategy that will force algorithm to query all variables. We consider all possible scenarios and show that for any deterministic algorithm always exists such adversary strategy that forces to query all variables. In other words, it means that for any fixed algorithm such input X always exists on which all variables must be queried.

First of all, we define a *winning game state* for an algorithm. In such a state next query will give function value for sure, either 0 or 1. We say that variable is *open* if its value is already known to an algorithm, otherwise variable is *closed*. State is *winning* if there are two crossing lines, where crossing point is closed, two points on one line are open as “0”, but two points on other line are open as “1”. Query about crossing point surely will be the last. See [Fig. 3] for an example of winning state.

Now let us examine all possible cases. Because of the symmetry of the Fano plane, number of cases to consider is rather small. After first three queries, only two different in essence states are possible: (1) three open points are located on one line, (2) three open points form a triangle. In both cases adversary strategy is to open two “0” and one “1”. See example in [Fig. 4].

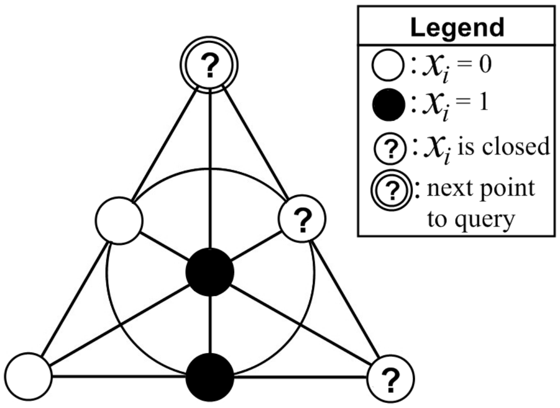

Figure 3: Example of winning game state for an algorithm

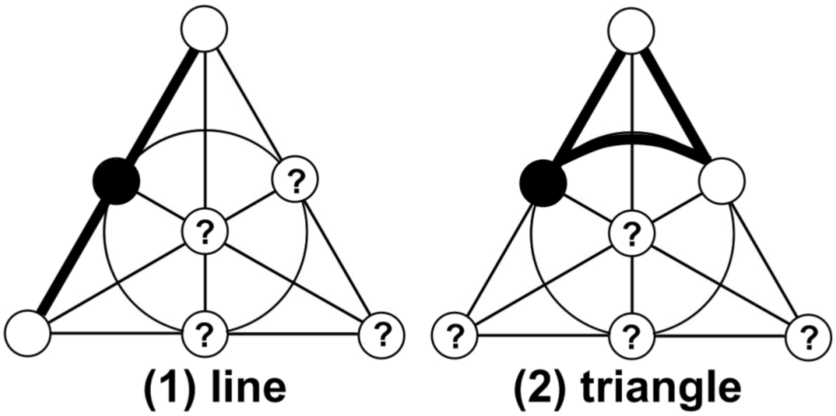

Figure 4: Example of two possible in essence distinguishable states after three queries

Case 1 (line). All four choices of variable for the fourth query are equivalent because of the symmetry. Adversary strategy in any case is to open “1”. After such fourth query there remains four potential constant lines:

– L1, with two “1” already open;

– L2, with one “0” already open;

– L3, with one “0” already open;

– L4, with one “1” already open.

See example in [Fig. 5].

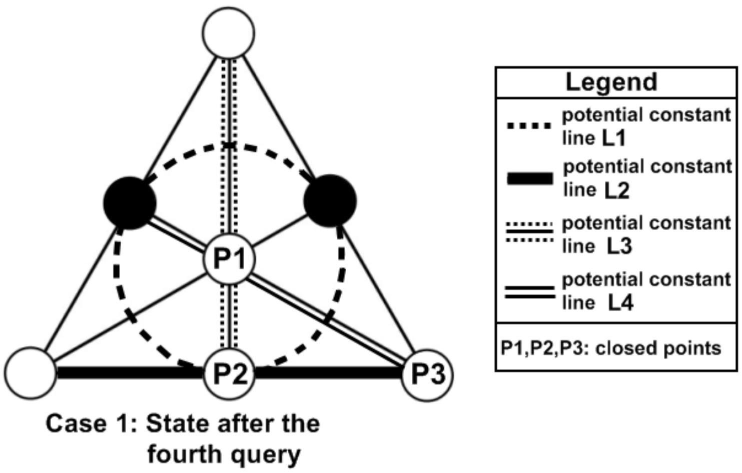

Figure 5: Case 1: illustration of potential constant lines and closed points after the fourth query

At the same time there remains three closed points:

- P1 $\in$ L2, L4;
- P2 $\in$ L1, L2, L3;
- P3 $\in$ L3, L4.

We consider three possible cases for an algorithm to choose variable for the fifth query.

Case 1.1. If algorithms chooses point P1 for the fifth query, adversary strategy is to open "1". Result - there is no winning state and thus adversary is able to force algorithm to query all seven variables. See [Fig. 6.A].

Case 1.2. If algorithms chooses point P2 for the fifth query, adversary strategy is to open "0". There again is no winning state. So, adversary will be able to give non-finishing variable value to any next query and seventh query will be required. See [Fig. 6.B].

Case 1.3. For the last remaining option, point P3, adversary strategy is to open "0". No winning state, so adversary again is able to force algorithm to query all seven variables. See [Fig. 6.C].

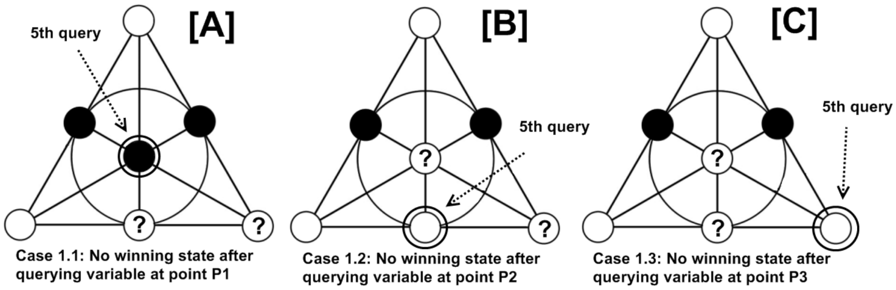

Figure 6: Cases 1.1, 1.2, 1.3: illustration of possible states after fifth query

Case 2 (triangle). Four closed points that remain after third query can be divided to two sets. We consider two different cases (2.1 and 2.2) based on this separation.

Case 2.1. There are three points (e.g. lower line in [Fig. 4.2]), opening of which after fourth query will bring a game to the state equivalent to that described in Case 1 [2]. Adversary strategy for such fourth query is to open "1". As shown above, in such situation adversary is able to force querying all seven variables.

Case 2.2. There is one point (e.g. central point in [Fig. 4.2]), after opening of which one line of the Fano plane will remain still fully closed. Adversary strategy is to open "0" for such fourth query. Otherwise, winning state would appear. Next, for any algorithm choice for fifth query adversary strategy is to open "1". There will be no winning state, so, adversary is able to force algorithm to query all seven variables. See example in [Fig. 7].

We analyzed all possible cases and demonstrated that for any fixed deterministic algorithm such input X always exists on which all seven variables must be queried.

[2] The roles of "0" and "1" are interchanged in some cases.

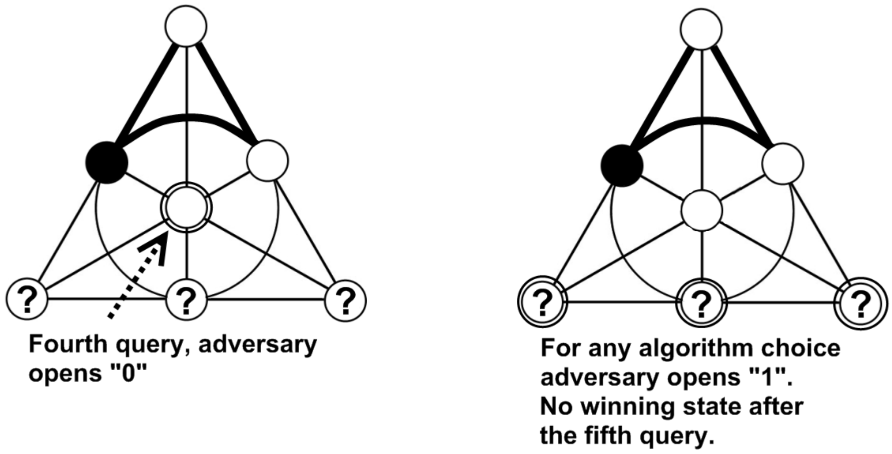

Figure 7: Case 2.2: illustration of possible states after fourth and fifth queries

4.3 Nondeterministic Algorithm for the Fano Plane Boolean Function

In this subsection, we demonstrate the computation of the Fano plane Boolean function in proposed alternative nondeterministic query model.

The first step is to define helper function $G_{FANO}(X, Y)$. We add three helper variables, so in total there will be ten variables:

$$G_{FANO}(x_1, ..., x_7, y_1, y_2, y_3).$$

We assign a binary sequence number to each line of the Fano plane. This assignment can be arbitrary, we will use variant presented in [Tab. 1].

Line variables	Line number	Line variables	Line number
x_1, x_2, x_3	000	x_3, x_4, x_6	100
x_5, x_3, x_7	001	x_1, x_4, x_5	101
x_7, x_1, x_6	010	x_2, x_4, x_7	110
x_6, x_2, x_5	011		

Table 1: Binary numbering of the Fano plane lines

Variables of the partial helper function $G_{FANO}(X, Y)$ are divided to two subsets. Variables of the X subset represent variable assignment of original $FANO(X)$ Boolean function. Variables of the Y subset represent the Fano plane line binary number. Since there is no line numbered with "111", function G_{FANO} is not defined for all inputs where $y_1 = y_2 = y_3 = 1$.

Definition 9 *Partial Boolean function $G_{FANO}(X, Y)$ is defined as:*

- *$G_{FANO}(X, y_1, y_2, y_3) = 1$, if the Fano plane line numbered with $y_1y_2y_3$ is constant and variables on that line are assigned Boolean value 1;*
- *$G_{FANO}(X, y_1, y_2, y_3) = 0$, if the Fano plane line numbered with $y_1y_2y_3$ is constant and variables on that line are assigned Boolean value 0;*
- *Otherwise, function value is not defined.*

For the illustration purpose partial truth table for inputs X=0000001 and X=0110110 is given in [Tab. 2].

$X = (x_1, x_2, ..., x_7)$	$Y = (y_1, y_2, y_3)$	$FANO(X)$	$G_{FANO}(X, Y)$
0000001	000	0	0
	001		not defined
	010		not defined
	011		0
	100		0
	101		0
	110		not defined
	111		not defined
...	...	...	...
0110110	000	1	not defined
	001		not defined
	010		not defined
	011		1
	100		not defined
	101		not defined
	110		not defined
	111		not defined

Table 2: Partial truth table for $G_{FANO}(X, Y)$

Now, when we have the definition of helper function $G_{FANO}(X, Y)$, the next step if to design an algorithm for computing this function. We would like to remind that this algorithm has to be deterministic and variable values retrieved during computation process shall give us enough information to verify the value of $FANO(X)$ function independently.

The deterministic decision tree algorithm for computing $G_{FANO}(X,Y)$ consists of the following steps:

1. Sequentially query variables y_1, y_2 and y_3.
2. Find the Fano plane line numbered with $y_1y_2y_3$ and sequentially query all three variables composing this line.
3. If all variable values retrieved in the second step are equal - output this Boolean value. Otherwise, output "*not defined*".

All three variables composing the line must be queried in the step 2 because of the restriction that retrieved information should allow us to verify the value of $FANO(X)$ function. To perform such verification we simply substitute retrieved variable values to the logical formula of $FANO(X)$ and ensure that it evaluates to the correct value.

Theorem 2 *Nondeterministic query complexity of the Fano plane Boolean function FANO(X) with the fixed helper function $G_{FANO}(X,Y)$ is*

$$ND_{G_{FANO}}(FANO) = 6.$$

Proof. Algorithm described above performs three queries to the black box in the first step and next three queries to the black box in the second step. □

4.4 Complexity of the Recursive Fano Plane Function

Finally, we show that a gap between deterministic and nondeterministic query complexity can be asymptotically large.

Definition of the Fano plane function can be applied recursively.

Definition 10 *Recursive Boolen function $FANO^i$ is defined as follows:*

– $FANO^1(X^1) = FANO(x_1, x_2, ..., x_7)$;
– $FANO^i(X^i) = FANO^1(FANO^{i-1}(X_1^{i-1}), ..., FANO^{i-1}(X_7^{i-1}))$, *where* $X^i = X_1^{i-1}X_2^{i-1}...X_7^{i-1}$

Recursively defined Boolean function $FANO^N(X)$ has 7^N variables.

Theorem 3 *Deterministic decision tree complexity of the recursive Fano plane Boolean function $FANO^N$ is $D(FANO^N) = 7^N$.*

Proof. Since $D(FANO^1) = 7$, on each recursion level we need to know values of all seven sub-functions. So, on the last level, when we come down to variables, the total number of variables to be queried is equal to 7^N. □

In a nondeterministic helper function $G_{FANO^N}(X, Y)$ for a recursive Fano plane function $FANO^N$ three additional helper variables are defined for each recursion level. These helper variables indicate which three sub-functions need to be computed in order to determine function value. The total complexity of the deterministic algorithm for the helper function evaluates to $O(3^N)$.

Theorem 4 *Nondeterministic query complexity of the recursive Fano plane Boolean function* $FANO^N$ *with the fixed helper function is*

$$ND_G(FANO^N) = O(3^N).$$

Proof. For each recursion level, first, we query three helper variables to determine three sub-functions that compose a line. Next, we go one level deeper and calculate value of each sub-function. Calculation of number of queries[3] is presented in [Tab. 3].

Recursion level	Number of queries
$i = 1$	$ND_G(FANO^1) = 3_y + 3_x = 6$
$i = 2$	$ND_G(FANO^2) = 3_y + 3(3_y + 3_x) = 3_y + 9_y + 9_x = 21$
$i = 3$	$ND_G(FANO^3) = 3_y + 3(3_y + 3(3_y + 3_x)) =$ $= 3_y + 9_y + 27_y + 27_y = 66$
...	...
$i = N$	$ND_G(FANO^N) = 3_y + 3(ND_G(FANO^{N-1})) =$ $= (\sum_{i=1}^{N} 3_y^i) + 3_x^N$

Table 3: Calculation of nondeterministic query complexity for different recursion levels

Finally, we use partial sum formula for the term $\sum_{i=1}^{N} 3^i$ and derive the complexity estimation for the case of recursion level $i = N$:

$$ND_G(FANO^N) = \left(\sum_{i=1}^{N} 3^i\right) + 3^N = \tfrac{3}{2}(3^N - 1) + 3^N = \tfrac{5}{2} \cdot 3^N - \tfrac{3}{2} = O(3^N)$$

□

[3] Subscript s near each number i_s (e.g. 3_x or 3_y) indicates that variable from subset S (X or Y) is queried.

5 Conclusion

In this paper, we studied nondeterministic query algorithms for computing Boolean functions. We presented a new vision of nondeterminism in query algorithms and introduced an alternative definition of a nondeterministic query model. The main difference from traditional nondeterministic model is that nondeterministic behavior is not obtained free of charge, but additional queries must be spent to obtain nondeterministic help. In some sense, this is a way to measure the amount of nondeterminism in an algorithm. We demonstrated our model using an example of computing the Fano plane Boolean function. Proposed alternative nondeterministic query model appeared to be well-suited for computing this kind of function. We demonstrated that when the definition of the function is applied recursively, a gap between deterministic and nondeterministic query complexity is 7^N versus $O(3^N)$.

Future work includes developing and improving the nondeterministic query model introduced in this paper. The main field of our interest is quantum algorithms, so next we are going to extend the model to a quantum case. The scope of further investigation is very wide, from designing algorithms for certain problems in this model to performing a detailed complexity analysis and comparison to other computational models. Considering Boolean function based on projective finite geometries, similar to the Fano plane function, seems to be a promising direction for searching for interesting examples.

Acknowledgements

This research was supported by Grant No. 09.1570 from the Latvian Council of Science and by the European Social Fund within the project "Support for Doctoral Studies at the University of Latvia".

References

[Buhrman, de Wolf 2002] Buhrman, H., de Wolf, R.: "Complexity Measures and Decision Tree Complexity: A Survey"; Theoretical Computer Science, 288, 1 (2002), 21-43.

[Floyd 1967] Floyd, R. W.: "Nondeterministic Algorithms"; Journal of the ACM (JACM), v.14 n.4, (1967), 636-644.

[Freivalds 2008] Freivalds, R.: "Non-constructive methods for finite probabilistic automata"; Int. J. Found. Comput. Sci. 19(3), (2008), 565-580.

[Freivalds, Iwama 2009] Freivalds, R., Iwama, K.: "Quantum Queries on Permutations with a Promise"; Implementation and Application of Automata, Lecture Notes in Computer Science, Volume 5642/2009, (2009), 208-216.

[Hopcroft, Ullman 1969] Hopcroft, J. E., Ullman J. D.: "Formal Languages and their Relation to Automata"; Addison-Wesley, Reading, MA, (1969).

[Lehmer, D.H., Lehmer, E. 1974] Lehmer, D.H., Lehmer, E.: "A New Factorization Technique Using Quadratic Forms"; Mathematics of Computation, 28, 126 (Apr., 1974), 625-635.

[Rabin, Scott 1959] Rabin, M. O., D. Scott, D.: "Finite automata and their decision problems"; IBM. Journal of Research and Development, 3:2, (1959), 114-125.

[Scegulnaja-Dubrovska, et. al., 2010] Scegulnaja-Dubrovska, O., Lace, L., Freivalds, R.: "Postselection Finite Quantum Automata"; UC 2010, Lecture Notes in Computer Science, Volume 6079/2010, (2010), 115-126

[Vasilieva (Dubrovska) 2007] Vasilieva (Dubrovska), A.: "Properties and Application of Nondeterministic Quantum Query Algorithms"; Proc. SOFSEM 2007, volume II, MatFyz Press (2007), 37-49.

[Vasilieva 2009] Vasilieva, A.: "Exact Quantum Query Algorithm for Error Detection Code Verification"; Fifth Doctoral Workshop on Mathematical and Engineering Methods in Computer Science, MEMICS 2009, Proceedings, ISBN 978-80-87342-04-6, (2009), 200-207.

[Vasilieva, et. al., 2010] Vasilieva, A., Mischenko-Slatenkova, T.: "Quantum Query Algorithms for Conjunctions"; UC 2010, Lecture Notes in Computer Science, Volume 6079/2010, (2010), 140-151.

[Weisstein] Weisstein, E. W.: "Fano Plane"; From MathWorld - A Wolfram Web Resource, http://mathworld.wolfram.com/FanoPlane.html.

[de Wolf 2003] de Wolf, R.: "Nondeterministic Quantum Query and Quantum Communication Complexities"; SIAM Journal on Computing, 32, 3 (2003), 681-699.