Quantum vs. deterministic queries on permutations

Alina Vasilieva, Taisia Mischenko-Slatenkova, Rūsiņš Freivalds and Ruslans Tarasovs

Department of Computer Science, University of Latvia, Raiņa bulvāris. 29, Riga, LV-1459, Latvia *
Alina.Vasiljeva@gmail.com, Taisia.Miscenko@gmail.com, Rusins.Freivalds@mii.lu.lv, Ruslan@tarasovs.com

Abstract. K.Iwama and R.Freivalds [12] considered query algorithms where the black box contains a permutation. Since then several authors have compared quantum and deterministic query algorithms for permutations. It turns out that the case of n-permutations where n is an odd number is difficult. There was no example of permutation problem where quantization can save half of the queries for $(2m+1)$-permutations if $m \geq 2$. Even for $(2m)$-permutations with $m \geq 2$ the best proved advantage of quantum query algorithms is the result in [12] where the quantum query complexity is m but the deterministic query complexity is $(2m-1)$. We present a group of 5-permutations such that the deterministic query complexity is 4 and the quantum query complexity is 2.

1 Introduction

Many papers on query algorithms consider computation of Boolean functions. The input of the query algorithm is a black box oracle containing the values of the variables $x_1 = a_1, x_2 = a_2, \cdots, x_n = a_n$ for an explicitly known Boolean function $f(x_1, \cdots, x_n)$. The result of the query algorithm is to be the value $f(a_1, \cdots, a_n)$. The query algorithm can ask for the values of the variables. The queries are asked individually, and the result of any query influences the next query to be asked or the result to be output.

The complexity of the query algorithm is defined as the number of the queries asked to the black box oracle. Deterministic query algorithms prescribe the next query uniquely depending only on the previously received answers from the black box oracle. Probabilistic query algorithms allow randomization of the process of computation. They sometimes allow to reduce the complexity of the algorithm dramatically [1].

Quantum query algorithms (see a formal definition in [6]) consists of a finite number of states in each of which they make a query to the black box oracle

* The research was supported by Grant No. 09.1570 from the Latvian Council of Science and by Project 2009/0216/1DP/1.1.1.2.0/09/IPIA/VIA/044 from the European Social Fund.

and determine how to change states. In fact they alternate *query operations* and *unitary transformations*. In the steps called *query operations* the states of the algorithm are divided into subsets corresponding to the allowed quantum-parallel queries. If each of states $q_{i_1}, \cdots, q_{i_m}$ asks a query "$x_i =?$" then for every possible answer "$x_i = j$" a unitary operation over the states $q_{i_1}, \cdots, q_{i_m}$ is pre-programmed. In the steps called *unitary transformations* the amplitudes of all states are transformed according a unitary matrix. All the sequence of the steps is ended in a special operation called *measurement* in which the amplitudes (being complex numbers) for all the states are substituted by real numbers called *probabilities* by the following rule. The complex number $a + bi$ is substituted by the real number $a^2 + b^2$. It follows from the unitarity of all the operations that the total of the probabilities of all states equals 1. Some states are defined to be accepting, and the other states are defined to be rejecting. This distinction is not seen before the measurement. After the measurement the probabilities of the accepting states are summed up and the result is called the accepting probability. We say that the quantum query algorithm is *exact* if the accepting probability is always either 1 or 0.

The notion of *promise* for quantum algorithm was introduced by Deutsch and Jozsa [10], and Simon [13]. In quantum query algorithms for problems under promise the domain of correctness of the algorithm is explicitly restricted. We are not interested in behavior of the algorithm outside this restriction. For instance, in this paper all the query algorithms are considered under a promise that the target function describes a permutation (in a way precisely stated below).

Recently there have been many papers studying query algorithms computing Boolean functions. Powerful methods to prove lower bounds of quantum query complexity were developed by A. Ambainis [2, 3]. A good reference is the survey by Buhrman and de Wolf [6].

We consider in this paper a more general class of functions $f(x_1, \cdots, x_n)$, namely, functions $\{0, 1, 2, \cdots, n-1\}^n \to \{0, 1\}$. The domain $\{0, 1, 2, \cdots, n\}^n$ includes a particularly interesting case - permutations. For instance,

$$x_1 = 4, x_2 = 3, x_3 = 2, x_4 = 1, x_5 = 0$$

can be considered as a permutation of 5 symbols {0,1,2,3,4} usually described as 43210. Under such a restriction the functions $f : \{0, 1, 2, \cdots, n-1\}^n \to \{0, 1\}$ can be considered as properties of permutations. For instance, the function

$$f(0, 1, 2) = 1, f(1, 2, 0) = 1, f(2, 0, 1) = 1, f(0, 2, 1) = 0, f(1, 0, 2) = 0, f(2, 1, 0) = 0$$

describes the property of 3-permutations to be *even* (as opposed to the property to be *odd*).

The property of a permutation to be even or odd can be defined in many equivalent ways. One of the most popular definitions used below is as follows. A permutation $x_1 = a_1, x_2 = a_2, \cdots, x_n = a_n$ is called even (odd) if it can be obtained from the identical permutation $x_1 = 0, x_2 = 1, \cdots, x_n = n - 1$ by an even (odd) number of transpositions, i.e. mutual changes of exactly two elements of the permutation: substituting $x_i = a_i$ and $x_j = a_j$ by $x_i = a_j$ and $x_j = a_i$. It

is a well-known fact that the property to be even or odd does not depend on the particular sequence of transpositions. Deciding whether a given permutation is even or odd is called deciding the parity of this permutation.

Our Contribution. It is easy to see that for every n-permutation $(n-1)$ queries uniquely determine the permutation. Hence for arbitrary permutation problem the deterministic query complexity never exceeds $(n-1)$. Theorem 1 below gives us a hope that sometimes half of the number of queries can be eliminated by quantization. The proof of Theorem 1 suggests that Fourier transform might be used again in counterparts of this theorem for larger values of n. However, it is far from obvious how to organize the pairs of the values of results of the queries into a linear string which is needed to apply the Fourier transform.

The paper [12] attempted to show that quantum query algorithms can use only about a half of the number of queries needed for deterministic query algorithms for deciding parity of permutations. Unfortunately, this attempt was only partially successful. It was proved that for $2m$-permutations it suffices to have m quantum queries but for $2m+1$-permutations it suffices to have $m+1$ quantum queries. In both cases it is more than the half of the deterministic query complexity.

In this paper we were not able to construct an effective quantum query algorithm to decide parity of 5-permutations. Instead we propose another permutation problem for 5-permutations where a quantum algorithm is indeed exactly twice as efficient as the best deterministic algorithm.

For every problem where a quantum algorithm is more efficient than any deterministic algorithm, it is crucially important to have a rich structure of symmetries. This is why we use a group of 5-permutations such that it has interesting automorphisms. We believe that our algorithm admits a generalization for larger values of n but up to now we have been able to use it only for 5-permutations.

2 First example

Theorem 1. *([12]) There is an exact quantum query algorithm deciding the parity of 3-permutations with one query.*

Proof. By the way of quantum parallelism, in the state q_1 we ask the query x_1 with an amplitude $\frac{1}{\sqrt{3}}$, in the state q_2 we ask the query x_2 with an amplitude $\frac{1}{\sqrt{3}}$, and in the state q_3 we ask the query x_3 with an amplitude $\frac{1}{\sqrt{3}}$.

If the answer from the black box to the query x_1 is 0, we do not change the amplitude of the state q_1. If the answer is 1, we multiply the existing amplitude to $e^{i\frac{2\pi}{3}}$. If the answer is 2, we multiply the existing amplitude to $e^{i\frac{4\pi}{3}}$.

If the answer from the black box to the query x_2 is 0, we multiply the amplitude of the state q_2 to $e^{i\frac{4\pi}{3}}$. If the answer is 1, we do not change the amplitude. If the answer is 2, we multiply the existing amplitude to $e^{i\frac{2\pi}{3}}$.

If the answer from the black box to the query x_3 is 0, we multiply the amplitude the state q_3 to $e^{i\frac{2\pi}{3}}$. If the answer is 1, we multiply the existing amplitude to $e^{i\frac{4\pi}{3}}$. If the answer is 2, we do not change the amplitude.

We process the obtained amplitudes of the states q_1, q_2, q_3 by a unitary transformation corresponding to the matrix

$$\begin{pmatrix} (\frac{1}{\sqrt{3}}) & (\frac{1}{\sqrt{3}}) & (\frac{1}{\sqrt{3}}) \\ (\frac{1}{\sqrt{3}}) & (\frac{1}{\sqrt{3}})e^{i\frac{2\pi}{3}} & (\frac{1}{\sqrt{3}})e^{i\frac{4\pi}{3}} \\ (\frac{1}{\sqrt{3}}) & (\frac{1}{\sqrt{3}})e^{i\frac{4\pi}{3}} & (\frac{1}{\sqrt{3}})e^{i\frac{2\pi}{3}} \end{pmatrix}$$

This transformation is a particular case of Fourier transform. If we are computing $f(0,1,2)$, $f(1,2,0)$ or $f(2,0,1)$ (these are all even 3-permutations) the amplitude of the state q_1 becomes, correspondingly, 1, $e^{i\frac{2\pi}{3}}$ or $e^{i\frac{4\pi}{3}}$. After measuring this state we get the probability 1. If we are computing $f(0,2,1)$ (which is an odd permutation) the amplitude of the state q_1 becomes 0 but the amplitude of the state q_2 becomes 1. If we are computing $f(1,0,2)$ or $f(2,1,0)$ (which are odd permutations) the amplitude of the state q_1 becomes 0 but the amplitude of the state q_3 becomes $e^{i\frac{4\pi}{3}}$ or $e^{i\frac{2\pi}{3}}$, correspondingly. □

3 Further results

We define a group GR of 5-permutations consisting of 20 permutations. These permutations are:

01234 12340 23401 34012 40123
02413 13024 24135 30241 41302
03142 14203 20314 31420 42031
04321 10432 21043 32104 43210

First of all, we note that the 5-permutations in GR can be represented as linear functions modulo 5.

$$\begin{array}{ccccc} x & x+1 & x+2 & x+3 & x+4 \\ 2x & 2x+1 & 2x+2 & 2x+3 & 2x+4 \\ 3x & 3x+1 & 3x+2 & 3x+3 & 3x+4 \\ 4x & 4x+1 & 4x+2 & 4x+3 & 4x+4 \end{array}$$

These permutations can be considered as group with a 2-argument algebraic operation "multiplication of permutations". The properties of this group GR are well-known already long ago. For instance, it is known that GR is not a commutative group and it is not a cyclic group.

In this section we prove that there is an exact quantum query algorithm deciding the membership in the group GR of 5-permutations with 2 queries. Obviously, 4 deterministic queries are needed for this problem because if a permutation

$$P = \{x_0 = a_0, x_1 = a_1, x_2 = a_2, x_3 = a_3, x_4 = a_4\}$$

is in GR and only 3 queries are asked, there remain two possibilities, namely, either $x_i = a_i, x_j = a_j$ and the permutation is in GR, or $x_i = a_j, x_j = a_i$ and the permutation is not in GR.

We wil prove below that there is an exact quantum query algorithm deciding the membership in the group GR of 5-permutations with two queries.

We need a numbering of 20 values of all possible pairs (a_i, a_j) where a_i and a_j values from the black box (i.e. the elements of the permutation in question). We might arrange these 20 values in accordance with the above-presented table but we prefer a slightly different layout.

$$\begin{array}{c} 01\ 12\ 23\ 34\ 40 \\ 02\ 24\ 41\ 13\ 30 \\ 04\ 43\ 32\ 21\ 10 \\ 03\ 31\ 14\ 42\ 20 \end{array}$$

These values correspond to the linear functions

$$\begin{array}{c} x \quad x+1 \quad x+2 \quad x+3 \quad x+4 \\ 2x\ 2x+1\ 2x+2\ 2x+3\ 2x+4 \\ 4x\ 4x+1\ 4x+2\ 4x+3\ 4x+4 \\ 3x\ 3x+1\ 3x+2\ 3x+3\ 3x+4 \end{array}$$

There is a certain regularity in this layout. Each row can be obtained from the preceding row multiplying it to 2 modulo 5. Each column can be obtained from the preceding column adding 1 modulo 5.

Now we introduce two distances among the values of these pairs. The distance $D_r[(u,v),(a,b)]$ between the pair (u,v) and the pair (a,b) is the the value (the number of the row of (a,b)) - (the number of the row of (u,v)) (mod 4). The distance $D_c[(u,v),(a,b)]$ between the pair (u,v) and the pair (a,b) is the the value (the number of the column of (a,b) - (the number of the column of (u,v)) (mod 5).

Our quantum query algorithm (in a way of quantum parallelism) enters (with equal amplitudes $\frac{1}{\sqrt{20}}$) 20 states. In each of these states the algorithm asks one of the 20 possible queries (x_i, x_j) where $x_i \in \{0,1,2,3,4\}, x_j \in \{0,1,2,3,4\}$, and $i \neq j$. In the result of these queries the amplitude is multiplied either to (-1) or to (+1). The following is a description of the value of these multipliers.

There are 20 values of all possible pairs (x_i, x_j) where $x_i \in \{0,1,2,3,4\}, x_j \in \{0,1,2,3,4\}$, and $x_i \neq x_j$. For every pair of queries (x_i, x_j) where $x_i \neq x_j$, the answer-pair (a_i, a_j) corresponds to the answer-pair (a_j, a_i) of the query (x_j, x_i). However, for the sake of symmetry, we have considered a quantum query algorithm with 20 possible pairs of queries.

Since the black box contains a permutation, the results of the query are also such that $a_i \in \{0,1,2,3,4\}, a_j \in \{0,1,2,3,4\}$, and $a_i \neq a_j$. Hence there are exactly 20 values of all possible answer-pairs (a_i, a_j).

For each of these 20 pairs (x_i, x_j) we get the corresponding pair of answers (a_i, a_j) and again $a_i \in \{0,1,2,3,4\}, a_j \in \{0,1,2,3,4\}$, and $a_i \neq a_j$. We can consider 20 distances $D_r[(x_i, x_j),(a_i, a_j)]$ and 20 distances $D_c[(x_i, x_j),(a_i, a_j)]$.

In the table below each of these pairs of distances is considered as a pair (w, z) where $w = D_r[(x_i, x_j),(a_i, a_j)]$ and $z = D_c[(x_i, x_j),(a_i, a_j)]$. Please observe that while asking the query (x_i, x_j) we automatically get an answer to the query

(x_j, x_i) as well but for this pair the distances may be different. Our quantum query algorithm uses this distinction essentially.

Suppose that the permutation in the black box is 03241 and we consider the pair of queries (x_2, x_4). Then the pair of answers (a_2, a_4) equals $(2, 1)$ and the pair of answers for (x_4, x_2) denoted as (a_4, a_2) equals $(1, 2)$. Then $D_r[(x_2, x_4), (a_2, a_4)]$ is equal to $D_r[(2, 4), (2, 1)] = 1$, $D_c[(2, 4), (2, 1)] = 2$, $D_r[(4, 2), (1, 2)] = 1$, and $D_c[(4, 2), (1, 2)] = 3$.

The following table describes in which cases the multiplier is $+1$ and in which cases it is -1. The first column corresponds to the pair

$$(w_1, z_1) = (D_r[(x_i, x_j), (a_i, a_j)], D_c[(x_i, x_j), (a_i, a_j)]),$$

the second column corresponds to

$$(w_2, z_2) = (D_r[(x_j, x_i), (a_j, a_i)]), D_c[(x_j, x_i), (a_j, a_i)]),$$

the last column corresponds to the multiplier $(+1)$ or (-1). For instance, the above-mentioned example is described in the table below as

$$(1, 2)\ (1, 3) \to +1$$

The table is as follows.

$$\begin{array}{l}
(0, 0)\ (0, 0) \to +1 \\
(0, 1)\ (0, 4) \to +1 \\
(0, 2)\ (0, 3) \to +1 \\
(0, 3)\ (0, 2) \to +1 \\
(0, 4)\ (0, 1) \to +1 \\
(1, 0)\ (1, 0) \to +1 \\
(1, 1)\ (1, 4) \to +1 \\
(1, 2)\ (1, 3) \to +1 \\
(1, 3)\ (1, 2) \to +1 \\
(1, 4)\ (1, 1) \to +1 \\
(2, 0)\ (2, 0) \to -1 \\
(2, 1)\ (2, 4) \to -1 \\
(2, 2)\ (2, 3) \to -1 \\
(2, 3)\ (2, 2) \to -1 \\
(2, 4)\ (2, 1) \to -1 \\
(3, 0)\ (3, 0) \to -1 \\
(3, 1)\ (3, 4) \to -1 \\
(3, 2)\ (3, 3) \to -1 \\
(3, 3)\ (3, 2) \to -1 \\
(3, 4)\ (3, 1) \to -1
\end{array}$$

Lemma 1. *If the permutation in the black box is from the group GR then all the 20 multipliers are equal.*

Proof. If the permutation corresponds to the function $ax + b$ where If $a = 3$ or $a = 4$ then the multiplier equals (-1). □

Lemma 2. *If the permutation in the black box is the following ones*

$$01243, 01342, 01423, 01324, 01432$$

then exactly 10 multipliers equal (-1) *and exactly 10 multipliers equal* $(+1)$*.*

Proof. By explicit counting. □

Lemma 3. *If the permutation in the black box* $f(x)$ *can be obtained from a permutation* $g(x)$ *in the set*

$$\{\ 01243,\ 01342,\ 01423,\ 01324,\ 01432\ \}$$

as $f(x) = ag(x) + b(mod 5)$ *then exactly 10 multipliers equal* (-1) *and exactly 10 multipliers equal* $(+1)$*.*

Proof. The definition of the values of multipliers depend only on the distances D_r but not on the distances D_c. Application of a linear function $at + b$ does not change the distance D_r. □

Lemma 4. *If the permutation in the black box is not from the group* GR *then exactly 10 multipliers equal* (-1) *and exactly 10 multipliers equal* $(+1)$*.*

Proof. The group G_5 of all 5-permutations consists of 120 elements. GR is a subgroup of G_5 consisting of 20 elements. Lagrange's theorem on finite groups shows that G_5 is subdivided into 6 cosets of equal size, one of the cosets being GR. The other 5 cosets GC_1, GC_2, GC_3, GC_4, GC_5 can be described as the set of all permutations $f(x)$ such that $f(x) = ag(x) + b(mod 5)$ and $g(x) \in GC_i$. It follows from Lemma 4 that exactly 10 multipliers equal (-1) and exactly 10 multipliers equal $(+1)$. □

Theorem 2. *There is an exact quantum query algorithm deciding the membership in the group* GR *of 5-permutations with two queries.*

Proof. Our quantum query algorithm (in a way of quantum parallelism) enters (with equal amplitudes $\frac{1}{\sqrt{20}}$) 20 states. In each of these states the algorithm asks one of the 20 possible queries (x_i, x_j) where $x_i \in \{0, 1, 2, 3, 4\}, x_j \in \{0, 1, 2, 3, 4\}$, and $i \neq j$. In the result of these queries the amplitude is multiplied either to (-1) or to (+1) as described in the table above. Fourier transform applied to these results. Lemmas 1 and 4 ensure that our quantum algorithm accepts all permutations in GR and rejects all permutations not in GR with probability 1. □

References

1. Farid M. Ablayev, Rūsiņš Freivalds. Why Sometimes Probabilistic Algorithms Can Be More Effective. *Lecture Notes in Computer Science,* vol. 233, p. 1–14, 1986.

2. Andris Ambainis. Quantum lower bounds by quantum arguments. Journal of Computer and System Sciences, vol. 64(4), p.750–767, 2002.
3. Andris Ambainis. Polynomial degree vs. quantum query complexity. *Proceedings of FOCS'98,* p.230–240, 1998.
4. Andris Ambainis and Rūsiņš Freivalds. 1-way quantum finite automata: strengths, weaknesses and generalizations. *Proceedings of FOCS'98*, p. 332–341. Also quant-ph/9802062.
5. Andris Ambainis and Ronald de Wolf. Average-case quantum query complexity. *Proceedings of STACS'2000,* p.133–144, 2000.
6. Harry Buhrman and Ronald de Wolf. Complexity measures and decision tree complexity: a survey. *Theoretical Computer Science,* vol. 288(1), p. 21–43, 2002.
7. Robert Beals, Harry Buhrman, Richard Cleve, Michele Mosca and Ronald de Wolf. Quantum lower bounds by polynomials. *Journal of the ACM,* vol. 48(4), p. 778—797, 2001.
8. Harry Buhrman, Richard Cleve, Ronald de Wolf and Christof Zalka. Bounds for small-error and zero-error quantum algorithms. *Proceedings of FOCS'99,* p.358–368, 1999.
9. Richard Cleve, Arthur Ekert, Chiarra Macchiavello and Michele Mosca. Quantum algorithms revisited. *Proceedings of the Royal Society of London,* vol. A 454, p. 339–354, 1998.
10. David Deutsch and Richard Jozsa. Rapid solutions of problems by quantum computation. *Proceedings of the Royal Society of London,* vol. A 439, p. 553, 1992.
11. Rūsiņš Freivalds. Languages recognizable by quantum finite automata. *Lecture Notes in Computer Science,* vol. 3845, p. 1–14, 2006.
12. Rūsiņš Freivalds, Kazuo Iwama. Quantum Queries on Permutations with a Promise. *Lecture Notes in Computer Science,* vol. 5642, p. 208–216, 2009.
13. I. Simon. String matching algorithms and automata. *Lecture Notes in Computer Science,* vol. 814, p. 386–395, 1994.