Quantum request-answer game with buffer model for online algorithms. Application for The Most Frequent Keyword Problem

Kamil Khadiev

Kazan Federal University, 18 Kremlyovskaya str, Kazan, Tatarstan, 420008, Russia

Abstract

We consider online algorithms as a request-answer game. An adversary that generates input requests, and an online algorithm answers. We consider a generalized version of the game that has a buffer of limited size. The adversary loads data to the buffer, and the algorithm has random access to elements of the buffer. We consider quantum and classical (deterministic or randomized) algorithms for the model. In the paper, we provide a specific problem (The Most Frequent Keyword Problem) and a quantum algorithm that works better than any classical (deterministic or randomized) algorithm in terms of competitive ratio. At the same time, for the problem, classical online algorithms in the standard model are equivalent to the classical algorithms in the request-answer game with buffer model.

Keywords
quantum computation, online algorithm, request-answer game, online minimization problem, buffer, keywords search

1. Introduction

One of the applications for online algorithms is optimization problems [1]. The peculiarity is the following. An algorithm reads an input piece by piece and returns an answer piece by piece immediately, even if an answer can depend on future pieces of the input. The algorithm should return an answer for minimizing an objective function (the cost of an output). The most standard method to define the effectiveness is the competitive ratio [2, 3].

One of the possible point of view to online algorithms is a request-answer game [4]. Here we consider a game of an online algorithm and Adversary that holds input. Adversary requests and the algorithm returns answers. We suggest a reversed version of the game. The algorithm asks an input variable and Adversary returns an answer, but as a price for the answer, Adversary asks to return an output variable. The new version of the game is equivalent to the original one, but we can generalize it. We provide the new model for online algorithms that is called "Request-answer Game with Buffer". The model is a game of three players that are an online algorithm, Adversary and Buffer of limited size. The algorithm can do a request of one of two

types:

- asking Adversary to load the next block of input variables to the Buffer;

- request Buffer for one of the holding variables.

For some integer parameter R, after each R requests Adversary asks an output variable. If the size of Buffer is 1 and $R = 1$, then the model is equivalent to the original one.

Motivation. Online algorithms have different applications. One of them is making a decision in current time with no knowledge about future data. Another one is processing a data stream and output a result data stream in online fashion, for example, streaming video on web sites and others. Many programming languages like Java, C++ [5, 6] and others use buffered data streams that store data in a fast buffer first, and then an algorithm reads data from the buffer. So, our model is like usage of buffered data streams. Additionally, we have asynchronous processing with online output. In other words, we focus on online behavior of the output stream, but when an algorithm reads an input stream, it can skip some data.

Quantum model. In the paper, we consider a quantum version of "Request-answer Game with Buffer" model. Quantum computing itself [7, 8, 9] is one of the hot topics in computer science. There are many problems where quantum algorithms outperform the best known classical algorithms [10, 11, 12, 13, 14]. Superior of quantum over classical was shown for different computational models like query model, streaming processing models, communication models and others [15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28].

Different versions of online quantum algorithms were considered in [21, 20] including quantum streaming algorithms as online algorithms [29, 30], quantum online algorithms with restricted memory [31, 32], quantum online algorithms with repeated test [33]. In these papers, authors show examples of problems that have quantum online algorithms with better competitive ratio comparing to classical online algorithms.

Our results. Here we provide a specific problem and a quantum online algorithm in "Request-answer Game with Buffer" model for it. We show that the quantum online algorithm has better competitive ratio than any classical (deterministic or randomized) counterpart. The problem is "The Most Frequent Keyword Problem". Questions are strings of length k; the problem is searching the most frequent keyword among words of a text and returning it after each word of the text immediately. The problem [34] is one of the most well-studied ones in the area of data streams [35, 36, 37]. Many applications in packet routing, telecommunication logging, and tracking keyword queries in search machines are critically based upon such routines. The similar problem in online fashion was considered in [38].

The paper is organized in the following way. Definitions are in Section 2. A description of the most frequent question problem and the quantum algorithm for the problem are described in Section 3. Section 4 contains lower bounds for classical algorithms.

2. Preliminaries

An online minimization problem consists of a set $\mathcal{I}$ of inputs and a cost function. Each input $I = (x_1, \dots, x_n)$ is a sequence of requests, where n is a length of the input $|I| = n$. Furthermore, a set of feasible outputs (or solutions) $\mathcal{O}(I)$ is associated with each I; an output is a sequence of answers $O = (y_1, \dots, y_n)$. The cost function assigns a positive real value $cost(I, O)$ to $I \in \mathcal{I}$ and $O \in \mathcal{O}(I)$. An optimal solution for $I \in \mathcal{I}$ is $O_{opt}(I) = argmin_{O \in \mathcal{O}(I)} cost(I, O)$.

Let us define an online algorithm for this problem. **A deterministic online algorithm** A computes the output sequence $A(I) = (y_1, \dots, y_n)$ such that y_i is computed by $x_1, \dots, x_i$. We say that A is *c-competitive* if there exists a constant $\alpha \geq 0$ such that, for every n and for any input I of size n, we have: $cost(I, A(I)) \leq c \cdot cost(I, O_{Opt}(I)) + \alpha$, where c is the minimal number that satisfies the inequality. Also we call c the **competitive ratio** of A. If $\alpha = 0, c = 1$, then A is optimal.

A randomized online algorithm R computes an output sequence $R^{\psi}(I) = (y_1, \dots, y_n)$ such that y_i is computed from $\psi, x_1, \dots, x_i$, where ψ is the content of the random tape, i. e., an infinite binary sequence, where every bit is chosen uniformly at random and independently of all the others. By $cost(I, R^{\psi}(I))$ we denote the random variable expressing the cost of the solution computed by R on I. R is c-competitive in expectation if there exists a constant $\alpha > 0$ such that, for every I, $\mathbb{E}\left[cost(I, R^{\psi}(I))\right] \leq c \cdot cost(I, O_{Opt}(I)) + \alpha$. We can say that c is expected competitive ratio for the algorithm.

2.1. Request-answer game with buffer model

The standard model for online algorithms can be considered as a request-answer game [4]. Adversary holds an input, it sends request x_i to an algorithm, and the algorithm sends answer y_i. Here Adversary is an "active" player that rules the game and the algorithm is a "passive" player that answers on each response.

Let us change the point of view to this game. Both are "active" players in some sense.

> **Round** 1. The algorithm asks an input variable x_1. (The algorithm is active on this round).
>
> **Round** 2. Adversary asks an output variable y_1. (Adversary is active on this round).
>
> ...
>
> **Round** $2i - 1$. The algorithm asks an input variable x_i. (The algorithm is active on this round).
>
> **Round** $2i$. Adversary asks an output variable y_i. (Adversary is active on this round).

It is easy to see that the new game is equivalent to the original game and the standard model.

Let us consider the modification of the game that has a buffer. Assume that we have a buffer between the algorithm and Adversary. Let a positive integer K be a size of the buffer. Additionally, there is an integer parameter $R \leq K$. The algorithm will ask to load data to the buffer by blocks of K variables. Let i be a number of the loading block. The algorithm can do the following actions if it is active on some round:

- The algorithm asks to erase the buffer and load the next K input variables $x_{i \cdot K+1}, \ldots, x_{i \cdot K+K}$ to the buffer. After that, i is increased by 1. ($i \leftarrow i + 1$)

- The algorithm requests any variable from the buffer. We consider a query model (decision tree model) for the algorithm that queries variables from the buffer.

The game has the following scenario:

Round 0. We initialize $i \leftarrow 0$

Round 1. The algorithm is active and it does the possible actions that were described before.

Round 2. The algorithm is active and it does the possible actions that were described before.

...

Round R. The algorithm is active and it does the possible actions that were described before.

Round $R + 1$. Adversary is active. He asks output variables $y_1, \ldots, y_R$.

...

Round $(R + 1) \cdot j + 1$. The algorithm is active and it does the possible actions that were described before.

Round $(R + 1) \cdot j + 2$. The algorithm is active and it does the possible actions that were described before.

...

Round $(R + 1) \cdot j + R$. The algorithm is active and it does the possible actions that were described before.

Round $(R + 1) \cdot j + R + 1$. Adversary is active. He asks output variables $y_{j \cdot R+1}, \ldots, y_{j \cdot R+R}$.

Comment. *In the case of $K = 1$ and $R = 1$, the new model is equivalent to the standard online algorithms model.*

In the randomized case, an algorithm that requests data from the buffer can be randomized, and we use a randomized query model in that case. We consider an expected competitive ratio for the model as for the standard model of randomized online algorithms. At the same time, the loading the next block to the buffer is deterministic action.

In the quantum case, an algorithm that requests data from the buffer can be quantum, and we use a quantum query model in that case. Because of the probabilistic behavior of quantum algorithms, we also consider an expected competitive ratio for the model. At the same time, the loading the next block to the buffer is deterministic action.

We skip details of the quantum model and quantum algorithms here because we use them as quantum subroutines and the rest part is classical. More details on quantum query model and quantum algorithms can be found in [7, 8, 9]

3. A quantum algorithm for The Most Frequent Keyword Problem

Let us present the problem formally.

Problem. For some positive integers m, d and k, the input is

$$I = (s^1, \dots, s^d, x^1, \dots, x^m).$$

Here $(s^1, \dots, s^d)$ is a sequence of strings that are interesting keywords for us in the input, $s^j = (s_1^j, \dots, s_k^j) \in \{0, 1\}^k$, for $j \in \{1, \dots, d\}$. Strings $x^1, \dots, x^m$ are words of a text, $x^j = (x_1^j, \dots, x_k^j) \in \{0, 1\}^k$, for $j \in \{1, \dots, m\}$. The input length is $n = (m + d) \cdot k$. A frequency of a string $t \in \{0, 1\}^k$ is $f(t) = \frac{\#(t)}{m}$, where $\#(t) = |\{i : t = x^i, i \in \{1, \dots, m\}\}|$ is a number of occurrence of t in $(x^1, \dots, x^m)$. The index i_0 of the most frequent string s^{i_0} is such that $f(s^{i_0}) = \max\limits_{i \in \{1, \dots, d\}} f(s^i)$ and i_0 is minimal. We should return index i_0 after reading each string x^j. So, the right answer that returns offline algorithm is $(z_1, \dots, z_n)$ where $z_{(j+d) \cdot k} = i_0$ for $j \in \{1, \dots, m\}$ and other output variables are not considered.

The cost of an output $O = (y_1, \dots, y_n)$ is

$$cost(I, O) = 1 + m - \sum_{j=1}^{m} \delta(y_{(j+d) \cdot l}, i_0)$$

Here $\delta(a, b) = 1$ if $a = b$ and $\delta(a, b) = 0$ if $a \neq b$

3.1. Quantum algorithm

Firstly, we discuss a quantum subroutine that compares two strings of length l for some integer $l > 0$.

3.1.1. The quantum algorithm for two strings comparing

Assume that the subroutine is COMPARE_STRINGS(s, t) and it compares s and t in lexicographical order. It returns:

- -1 if $s < t$;

- 0 if $s = t$;

- 1 if $s > t$.

As a base for our algorithm, we will use the algorithm of finding the minimal argument with 1-result of a Boolean-value function. Formally, we have:

Lemma 1. *[39] Suppose, we have a function $f : \{1, \dots, N\} \longrightarrow \{0, 1\}$ for some integer N. There is a quantum algorithm for finding $j_0 = \min\{j \in \{1, \dots, N\} : f(j) = 1\}$. The algorithm finds j_0 with query complexity $\sqrt{N}$ and error probability that is at most $\frac{1}{2}$.*

Let us choose the function $f(j) = (s_j \neq t_j)$. So, we search j_0 that is the index of the first unequal symbol of the strings. We search j_0 among indexes $1, \dots \min(|s|, |t|)$, where $|s|$ is a length of s. Then, we can claim that s precedes t in lexicographical order iff s_{j_0} precedes t_{j_0} in the alphabet for strings. If there are no unequal symbols, then we have one of three options:

- if $|s| < |t|$, then $s < t$;

- if $|s| > |t|$, then $s > t$;

- if $|s| = |t|$, then $s = t$.

We use THE_FIRST_ONE_SEARCH(f, N) as a subroutine from Lemma 1, where $f(j) = (s_j \neq t_j)$. Assume that this subroutine returns $N + 1$ if it does not find any solution.

We apply the standard technique of boosting success probability that was used, for example, in [12]. So, we repeat the algorithm $3 \log_2 m$ times and return the minimal answer, where m is a number of strings in the sequence $(x^1, \dots x^m)$. In that case, the error probability is $O\left(\frac{1}{2^{3 \log m}}\right) = O\left(\frac{1}{m^3}\right)$.

Let us present the algorithm.

Algorithm 1 COMPARE_STRINGS(s, t, k). The quantum algorithm for two strings comparing.

$N \leftarrow min(|s|, |t|)$
j_0THE_FIRST_ONE_SEARCH(f, N) ⊳ The initial value
for $i \in \{1, \dots, 3 \log_2 m\}$ **do**
    $j \leftarrow$ THE_FIRST_ONE_SEARCH(f, N)
    if $j \leq k$ and $s_j \neq s_t$ **then**
        $j_0 \leftarrow \min(j_0, j)$
    end if
end for
if $j_0 = N + 1$ and $|s| = |t|$ **then**
    $result \leftarrow 0$ ⊳ The strings are equal.
end if
if $((j_0 \neq N + 1)$ and $(s_{j_0} < t_{j_0}))$ or $((j_0 = N + 1)$ and $(|s| < |t|))$ **then**
    $result \leftarrow -1$ ⊳ s precedes t.
end if
if $((j_0 \neq N + 1)$ and $(s_{j_0} > t_{j_0}))$ or $((j_0 = N + 1)$ and $(|s| > |t|))$ **then**
    $result \leftarrow 1$ ⊳ t succeeds s.
end if
return $result$

Let us discuss the property of the algorithm:

Lemma 2. *Algorithm 1 compares two strings s and t in lexicographical order with query complexity $O(\sqrt{\min(|s|, |t|)} \log m)$ and error probability $O\left(\frac{1}{m^3}\right)$.*

Proof. The correctness of the algorithm follows from description and lexicographical order.

Let us discuss the error probability. The algorithm has error iff there are error in all $3 \log_2 m$ invocations of THE_FIRST_ONE_SEARCH algorithm. The probability of such event is at most $0.5^{3 \log_2 m} = O\left(\frac{1}{m^3}\right)$. □

3.1.2. A quantum algorithm in request-answer game with buffer model

Firstly, we present an idea of the algorithm.

We use the well-known data structure a self-balancing binary search tree. As an implementation of the data structure, we can use the AVL tree [40, 41] or the Red-Black tree [42, 41]. Both data structures allow us to find and add elements in $O(\log N)$ running time, where N is a size of the tree.

The idea of the algorithm is the following. We store a triple (i, s, c) in a vertex of the tree, where i is the minimal index of a string from $\{s^1, \dots, s^d\}$ such that $s = s^i$ and c is a number of occurrences of the string s among $\{x^1, \dots, x^m\}$. We assume that a triple (i, s, c) is less than a pair (i', s', c') iff s precedes s' in the lexicographical order. So, we use COMPARE_STRINGS(s, s', k) subroutine as the comparator of the vertexes. The tree represents a set of unique strings from $\{s^1, \dots, s^d\}$ with a number of occurrences among $(x^1, \dots, x^m)$.

Firstly, we load all strings $s^1, \dots, s^d$ one by one to Buffer and add a vertex $v = (j, s^j, 0)$ for each string s^j to the tree, here $j \in \{1, \dots, d\}$. We add only one node for each duplicate strings from $s^1, \dots, s^d$ if they exist. The index j in v stores the index of s^j and if there is no a vertex that corresponds to s^j, then j is a minimal index from all possible indexes. 0 in v means that initially we assume that s^j does not occurs among $(x^1, \dots, x^m)$.

Secondly, we load questions (strings) from x^1 to x^m one by one to Buffer and search them in our tree. We increase the number of occurrences. If the string was not found in the tree, then it is not a keyword, i.e. it does not belong to $s^1, \dots s^d$ and we skip it. At the same time, we store

$$(i_{max}, s, c_{max}) = argmax_{(i,t,c)} \text{ in the tree } c$$

and recalculate it in each step. When Adversary requests an output variable, then we return i_{max}.

Let us present the algorithm formally. Let BST be a self-balancing binary search tree such that:

- FIND(BST, x^i) finds a vertex (j, s, c) such that $s = x^i$, or $NULL$ if x^i was not found. The standard algorithm for searching x^i in the tree is comparing with elements of vertexes and moving by the tree according to the result of the comparison. When we invoke the COMPARE_STRINGS subroutine, we request a variable from Buffer for checking a symbol of x^i and request to memory when we check a symbol of a string that is stored in a vertex.

- ADD(BST, j, s^j) adds a vertex $(j, s^j, 0)$ to the tree if a vertex with s^j does not exist; and does nothing otherwise.

- INIT(BST) initializes an empty tree.

Let us discuss the property of the algorithm.

Algorithm 2 A quantum algorithm for The Most Frequent Keyword Problem.

INIT(BST)   ▷ The initialization of the tree.

$c_{max} \leftarrow 1$ ▷ The maximal number of occurrences.

$i_{max} \leftarrow 1$ ▷ The index of most frequent question.

$step \leftarrow 0$

for $j \in \{1, \dots, d\}$ **do**

    LOAD_TO_BUFFER   ▷ Load s^j to Buffer

    $t \leftarrow \}\}''$ ▷ Initially t is an empty string

    for $q \in \{1, \dots, k\}$ **do**   ▷ Reading the string t

        $t \leftarrow t + \text{REQUEST}(q)$ ▷ Requesting q-th variable from Buffer and appending the variable to t

    end for

    ADD(BST, j, t)   ▷ Adding the string $t = s^j$ to the tree as a vertex $(NULL, t, 0)$

end for

for $j \in \{1, \dots, m\}$ **do**

    LOAD_TO_BUFFER   ▷ Load x^i to Buffer

    $v = (i, t, c) \leftarrow \text{FIND}(BST, x^j)$ ▷ Searching x^i in the tree.

    if $v \neq NULL$ **then**   ▷ If x^i belongs to $(s^1, \dots, s^d)$

        $c \leftarrow c + 1$ ▷ Updating the vertex by increasing the number of occurrences.

        $v \leftarrow (i, t, c)$ ▷ Updating the vertex by the new values

        if $c > c_{max}$ **then**   ▷ Updating the maximal value.

            $c_{max} \leftarrow c$

            $i_{max} \leftarrow i$

        end if

    end if

end for

if Adversary request an output variable **then return** i_{max}

end if

Theorem 3. *The expected competitive ratio c for Algorithm 2 is at most C_Q where*

$$C_Q = O\left(1 + \frac{(m \log m) \cdot (\log d)}{\sqrt{k}}\right).$$

Proof. The correctness of the algorithm follows from the description. Let us discuss the query complexity of FIND(BST, x^j). The procedure requires $O(\log d)$ comparing operations COMPARE_STRINGS($x^j, s^{i'}, k$). Due to Lemma 2, each comparing operation requires $O(\sqrt{k} \log m)$ queries. The total query complexity of the FIND procedure is $O\left(\sqrt{k}(\log m) \cdot (\log d)\right)$. So, the algorithm checks all $x^1, \dots, x^m$ in $O\left(m\sqrt{k}(\log m) \cdot (\log d)\right)$ rounds and after that returns right answers for the requests of Adversary. Therefore, the first $O\left(\frac{m\sqrt{k}(\log m) \cdot (\log d)}{k}\right) = O\left(\frac{m(\log m) \cdot (\log d)}{\sqrt{k}}\right)$ "significant" output variables can be wrong and others are right. We call output variable $y_{(j+d) \cdot k}$, for $j \in \{1, \dots, m\}$, as "significant" because the cost depends on these variables. Hence, the cost is at most $1 + O\left(\frac{m(\log m) \cdot (\log d)}{\sqrt{k}}\right)$.

Let us discuss the error probability. Events of error in the algorithm are independent. So, all events should be correct. Due to Lemma 2, the probability of correctness of one event is $1 - \left(1 - \frac{1}{m^3}\right)$. Hence, the probability of correctness of all $O(m \log m)$ events is at least $1 - \left(1 - \frac{1}{m^3}\right)^{\gamma \cdot m \log m}$ for some constant γ.

Note that
$$\lim_{n \to \infty} \frac{\left(1 - \frac{1}{m^3}\right)^{\gamma \cdot m \log m}}{1/m} < 1;$$

Hence, the total error probability is at most $O\left(\frac{1}{m}\right)$.

In a case of an error, all "significant" output variables can be wrong.

Therefore, the expected competitive ratio of the algorithm is at most

$$C_Q = \frac{O(\frac{m-1}{m}) \cdot \left(1 + O\left(\frac{m(\log m) \cdot (\log d)}{\sqrt{k}}\right)\right) + O\left(m \cdot \frac{1}{m}\right)}{1} = O\left(1 + \frac{m(\log m) \cdot (\log d)}{\sqrt{k}}\right).$$

□

4. Lower bounds for classical algorithms for The Most Frequent Keyword Problem

There is an input I_B such that any classical (deterministic or randomized) algorithm returns output with the cost at least $O(m)$.

Theorem 4. *Any randomized algorithm for the problem has competitive ratio c at least $C_R = O(m) > C_Q$ in a case of $(\log_2 m) \cdot (\log_2 d) = o(\sqrt{k})$.*

Proof. Let us show that the problem is equivalent to unstructured search problem. Assume that $m = 2t$ for some integer t. Then, let $x^{t+1}, \dots, x^{2t} = 0^k$ where 0^k is a string of k zeros. We have two cases for other string:

- **case 1:** $x^1, \dots, x^t = 1^k$;

- **case 2:** there are $z \in \{1, \dots, t\}$ and $u \in \{1, \dots, k\}$ such that $x_u^z = 0$ and $x_{u'}^z = 1$ for all $u' \in \{1, \dots, u-1, u+1, \dots, k\}$, $x^{z'} = 1^k$ for $z' \in \{1, \dots, t\} \backslash \{z\}$.

Let $d = 2$, $s^1 = 0^k$ and $s^2 = 1^k$.

In the first case, the answer is 1^k. In the second case, the answer is 0^k. Therefore, the problem is equivalent to search 0 among the first $tk = mk/2$ variables.

Due to [43], the randomized query complexity of unstructed search among $mk/2$ is $\Omega(mk)$.

In a case of odd m, we assign $x^m = 1^{k/2}0^{k/2}$, and it is not used in the search. Then, we can consider only $m-1$ strings. So, $m-1$ is even.

Suppose, we have a randomized algorithm A for finding the most frequent question that uses $o(mk)$ queries to buffer when it reads $x^1, \dots, x^m$. Then, Adversary can construct the input I_B such that A obtains a wrong answer.

Therefore, all "significant" output variables will be wrong and $cost(I_B, A(I_B)) = 1 + m$. The competitive ratio in that case is $C_R = m + 1$.

If the algorithm do $O(mk)$ queries to Buffer for computing answer, then $O(m)$ "significant" output variables should be returned before getting a right answer. Therefore, $cost(I_B, A(I_B)) = O(m)$ and $C_R = O(m)$.

In the case of $(\log_2 m) \cdot (\log_2 d) = o(\sqrt{k})$ we have

$$C_Q = O\left(1 + \frac{m(\log_2 m) \cdot (\log_2 d)}{\sqrt{k}}\right) = o(m) < O(m) = C_R.$$

$\square$

5. Conclusion

We consider a new setting or new model for online algorithms that is useful for real world problems. We show that in the case of $(\log_2 m) \cdot (\log_2 d) = o(\sqrt{k})$ the quantum algorithm shows a better competitive ratio than any classical (deterministic or randomized) algorithm. Note that this setting is reasonable.

Acknowledgments

References

[1] D. Komm, An Introduction to Online Computation: Determinism, Randomization, Advice, Springer, 2016.

[2] D. D. Sleator, R. E. Tarjan, Amortized efficiency of list update and paging rules, Communications of the ACM 28 (1985) 202–208.

[3] A. R. Karlin, M. S. Manasse, L. Rudolph, D. D. Sleator, Competitive snoopy caching, in: FOCS, 1986., 27th Annual Symposium on, IEEE, 1986, pp. 244–254.

[4] S. Albers, BRICS, Mini-Course on Competitive Online Algorithms, Aarhus University, 1996.

[5] Oracle, Java Platform SE 8 documentation, 2021. URL: https://docs.oracle.com/javase/8/docs/api/java/io/BufferedReader.html.

[6] S. B. Lippman, J. Lajoie, C++ Primer, third ed., Massachusetts: Addison-Wesley, 1998.

[7] M. A. Nielsen, I. L. Chuang, Quantum computation and quantum information, Cambridge univ. press, 2010.

[8] A. Ambainis, Understanding quantum algorithms via query complexity, in: Proc. Int. Conf. of Math. 2018, volume 4, 2018, pp. 3283–3304.

[9] F. Ablayev, M. Ablayev, J. Z. Huang, K. Khadiev, N. Salikhova, D. Wu, On quantum methods for machine learning problems part i: Quantum tools, Big Data Mining and Analytics 3 (2019) 41–55.

[10] R. de Wolf, Quantum computing and communication complexity, Ph.D. thesis, 2001.

[11] S. Jordan, Quantum algorithms zoo, 2021. URL: http://quantumalgorithmzoo.org/.

[12] K. Khadiev, L. Safina, Quantum algorithm for dynamic programming approach for dags. applications for zhegalkin polynomial evaluation and some problems on dags, in: Proceedings of UCNC 2019, volume 4362 of *LNCS*, 2019, pp. 150–163.

[13] K. Khadiev, D. Kravchenko, D. Serov, On the quantum and classical complexity of solving subtraction games, in: Proceedings of CSR 2019, volume 11532 of *LNCS*, 2019, pp. 228–236.

[14] K. Khadiev, I. Mannapov, L. Safina, The quantum version of classification decision tree constructing algorithm c5. 0, CEUR Workshop Proceedings 2500 (2019).

[15] A. Ambainis, N. Nahimovs, Improved constructions of quantum automata, Theoretical Computer Science 410 (2009) 1916–1922.

[16] F. Ablayev, A. Vasiliev, On quantum realisation of boolean functions by the fingerprinting technique, Discrete Mathematics and Applications 19 (2009) 555–572.

[17] F. Ablayev, A. Gainutdinova, K. Khadiev, A. Yakaryılmaz, Very narrow quantum OBDDs and width hierarchies for classical OBDDs, in: DCFS, volume 8614 of *LNCS*, Springer, 2014, pp. 53–64.

[18] F. Ablayev, A. Gainutdinova, K. Khadiev, A. Yakaryılmaz, Very narrow quantum OBDDs and width hierarchies for classical OBDDs, Lobachevskii Journal of Mathematics 37 (2016) 670–682.

[19] F. Ablayev, A. Ambainis, K. Khadiev, A. Khadieva, Lower bounds and hierarchies for quantum memoryless communication protocols and quantum ordered binary decision diagrams with repeated test, In SOFSEM 2018, LNCS 10706 (2018) 197–211.

[20] F. Ablayev, M. Ablayev, K. Khadiev, A. Vasiliev, Classical and quantum computations with restricted memory, LNCS 11011 (2018) 129–155.

[21] K. Khadiev, A. Khadieva, I. Mannapov, Quantum online algorithms with respect to space and advice complexity, Lobachevskii Journal of Mathematics 39 (2018) 1210–1220.

[22] K. Khadiev, A. Khadieva, Reordering method and hierarchies for quantum and classical ordered binary decision diagrams, in: CSR 2017, volume 10304 of *LNCS*, Springer, 2017, pp. 162–175.

[23] R. Ibrahimov, K. Khadiev, K. Prūsis, A. Yakaryılmaz, Error-free affine, unitary, and probabilistic OBDDs, Lecture Notes in Computer Science 10952 LNCS (2018) 175–187.

[24] F. Le Gall, Exponential separation of quantum and classical online space complexity, Theory of Computing Systems 45 (2009) 188–202.

[25] K. Khadiev, A. Ilikaev, Quantum algorithms for the most frequently string search, intersection of two string sequences and sorting of strings problems, in: International Conference on Theory and Practice of Natural Computing, 2019, pp. 234–245.

[26] D. Kravchenko, K. Khadiev, D. Serov, R. Kapralov, Quantum-over-classical advantage in solving multiplayer games, Lecture Notes in Computer Science 12448 (2020) 83–98.

[27] A. Glos, N. Nahimovs, K. Balakirev, K. Khadiev, Upperbounds on the probability of finding marked connected components using quantum walks, Quantum Information Processing 20 (2021) 1–23.

[28] A. Ambainis, K. Balodis, J. Iraids, K. Khadiev, V. Kḷevickis, K. Prūsis, Y. Shen, J. Smotrovs, J. Vihrovs, Quantum Lower and Upper Bounds for 2D-Grid and Dyck Language, in: 45th International Symposium on Mathematical Foundations of Computer Science (MFCS 2020), volume 170 of *Leibniz International Proceedings in Informatics (LIPIcs)*, 2020, pp. 8:1–8:14.

[29] K. Khadiev, A. Khadieva, D. Kravchenko, A. Rivosh, R. Yamilov, I. Mannapov, Quantum versus classical online streaming algorithms with logarithmic size of memory, Lobachevskii Journal of Mathematics (2019). (in print). arXiv:1710.09595.

[30] K. Khadiev, A. Khadieva, Quantum online streaming algorithms with logarithmic memory, International Journal of Theoretical Physics (2019). doi:`10.1007/s10773-019-04209-1`.

[31] K. Khadiev, A. Khadieva, Two-way quantum and classical machines with small memory for online minimization problems, in: International Conference on Micro- and Nano-Electronics 2018, volume 11022 of *Proc. SPIE*, 2019, p. 110222T. doi:`10.1117/12.2522462`.

[32] K. Khadiev, A. Khadieva, Two-way quantum and classical automata with advice for online minimization problems, in: Formal Methods. FM 2019 International Workshops, 2020, pp. 428–442.

[33] Q. Yuan, Quantum online algorithms, Ph.D. thesis, University of California, Santa Barbara, 2009.

[34] G. Cormode, M. Hadjieleftheriou, Finding frequent items in data streams, Proceedings of the VLDB Endowment 1 (2008) 1530–1541.

[35] S. Muthukrishnan, Data streams: Algorithms and applications, Foundations and Trends in Theoretical Computer Science 1 (2005) 117–236.

[36] C. C. Aggarwal, Data streams: models and algorithms, volume 31, Springer Science & Business Media, 2007.

[37] L. Becchetti, I. Chatzigiannakis, Y. Giannakopoulos, Streaming techniques and data aggregation in networks of tiny artefacts, Computer Science Review 5 (2011) 27–46.

[38] J. Boyar, K. S. Larsen, A. Maiti, The frequent items problem in online streaming under various performance measures, International Journal of Foundations of Computer Science 26 (2015) 413–439.

[39] R. Kapralov, K. Khadiev, J. Mokut, Y. Shen, M. Yagafarov, Fast classical and quantum algorithms for online k-server problem on trees, arXiv preprint arXiv:2008.00270 (2020).

[40] G. M. Adel'son-Vel'skii, E. M. Landis, An algorithm for organization of information, in: Doklady Akademii Nauk, volume 146, Russian Academy of Sciences, 1962, pp. 263–266.

[41] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, Introduction to Algorithms, McGraw-Hill, 2001.

[42] L. J. Guibas, R. Sedgewick, A dichromatic framework for balanced trees, in: Proceedings of SFCS 1978, IEEE, 1978, pp. 8–21.

[43] C. H. Bennett, E. Bernstein, G. Brassard, U. Vazirani, Strengths and weaknesses of quantum computing, SIAM journal on Computing 26 (1997) 1510–1523.