



Article

Quantum and Classical Log-Bounded Automata for the Online Disjointness Problem

Kamil Khadiev ^{1,*}  and Aliya Khadieva ^{1,2} 

¹ Institute of Computational Mathematics and Information Technologies, Kazan Federal University, Kremlevskaya Str. 18, 420008 Kazan, Russia; AliHadieva@kpfu.ru

² Faculty of Computing, University of Latvia, Raiņa bulvaris 19, LV-1586 Riga, Latvia

* Correspondence: kamil.khadiev@kpfu.ru

Abstract: We consider online algorithms with respect to the competitive ratio. In this paper, we explore one-way automata as a model for online algorithms. We focus on quantum and classical online algorithms. For a specially constructed online minimization problem, we provide a quantum log-bounded automaton that is more effective (has less competitive ratio) than classical automata, even with advice, in the case of the logarithmic size of memory. We construct an online version of the well-known Disjointness problem as a problem. It was investigated by many researchers from a communication complexity and query complexity point of view.

Keywords: quantum computation; online algorithms; streaming algorithms; online minimization problems; automata



Citation: Khadiev, K.; Khadieva, A. Quantum and Classical Log-Bounded Automata for the Online Disjointness Problem. *Mathematics* **2022**, *10*, 143. <https://doi.org/10.3390/math10010143>

Academic Editor: Theodore Andronikos

Received: 26 September 2021

Accepted: 27 December 2021

Published: 4 January 2022

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



Copyright: © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

In the paper, we discuss a computational model used to solve optimization problems. We focus on online algorithms that many researchers explore. An online algorithm processes an input data stream and outputs a data stream in an online fashion. It should return a piece of output variables immediately after reading a piece of input variables. At the same time, the best answer can depend on the whole input. The main goal is a minimization of an objective function (we call it the cost of an output). We can say that an online algorithm solves an online minimization problem. Researchers consider different measures of effectiveness for online algorithms [1,2]. At the same time, the competitive ratio is the most standard and useful [3,4]. If we consider the cost of the output generated by an online algorithm and the cost of the output produced by an optimal offline algorithm, then the competitive ratio is the ratio of these two costs. If the ratio is at most c in the worst case, then we call the algorithm c -competitive. We also can say that the competitive ratio of the algorithm is c .

Typically, online algorithms have no limits for time and memory complexity. The main restriction is a lack of knowledge about future input variables. At the same time, researchers develop online algorithms with different types of restrictions. Some of them are restrictions on memory [5–11], other ones are restrictions on time complexity [12–14]. Often, an algorithm with restricted resources is closer to real-world applications.

In this paper, we focus on the memory complexity of online algorithms. We expect that an input stream is such big that it cannot be stored entirely in memory. An online algorithm processes a data stream. That is why we can consider automata (streaming algorithms, state machines) as online algorithms for this setup. In the paper, we use the “log-bounded automata” term. It is a uniform automata-like model with memory such that the size of memory can depend on the length of an input. This model was explored in [5,7,8]. Note that the automaton is allowed to use only one input head in the considered model. Sometimes, for emphasizing that the model is input data processing model and it is “one-pass”, we call the model one-way log-bounded automata. It means that we

cannot reread the input and store the whole required information about the past in memory. Sometimes, to emphasize that the algorithms are used as online algorithms or for solving an online minimization problem, we call the model one-way log-bounded automata for an online minimization problem.

We consider classical and quantum models of “log-bounded automata” as online algorithms. The classical model was developed in papers of different researchers [5–7,10,11] for a long time. At the same time, the quantum model was introduced several years ago in [8] and discussed in [9]. Quantum computing [15–17] itself is one of the hot topics in computer science of the last few decades. There are many problems where quantum algorithms outperform the best-known classical algorithms [18–20]. A restriction for the memory size is especially important for the quantum model of algorithms because near-future quantum computers are able to manipulate with a few qubits.

In the case of one-way log-bounded automata as online algorithms, we know that the quantum model can be more effective than its classical counterparts with respect to a competitive ratio. The result was proven for two cases. The first one is sublogarithmic size of memory [8]. The second case is polylogarithmic size of memory [21]. Note that if an algorithm can use the linear size of memory, then it can store the whole input in memory. In that case, it is equivalent to the standard online algorithm model without restriction on the memory size. That is why the only sub-linear size of memory is interesting in the case of the restricted size of memory.

Researchers also considered other quantum models. One of them is quantum online algorithms with repeated test [22]; another one is quantum online algorithms with restriction on the size of memory [23,24]; quantum algorithms with a buffer [25,26] and others. The question of comparing quantum and classical models was explored for data stream processing models by different researches [27–40].

Our Results

Let us focus on demonstrating the quantum supremacy in the case of the logarithmic size of memory. For this goal, we choose a problem and demonstrate that a quantum model for the problem is more effective than a classical counterpart.

A similar result was proven in [21]. At the same time, the problem that was considered in [21] has one restriction. It is not defined for all inputs and requires a pre-validation to check whether an input is permissible. Such problems are called “promise problems” or “partial functions”. It is an important restriction for a problem. The paper explores “total functions” or problems without a pre-validation procedure for an input.

We investigate an online minimization problem `onlineDISJ` that is based on the Disjointness Boolean function $DISJ(x, y)$. For some integer b , the function $DISJ$ is defined on two binary strings $S = (S_1, \dots, S_b)$ and $U = (U_1, \dots, U_b)$, and $DISJ(S, U) = \neg \bigvee_{i=1}^b S_i \wedge U_i$. Here S and U are bit-masks (or characteristic vectors) of two input sets $A, B \subset \{1, \dots, b\}$. The result is 1 iff $A \cap B = \emptyset$.

Researchers explored communication complexity and quantum query algorithms for the Disjointness Boolean function [41–43]. Additionally, the function was used for comparing quantum and classical streaming algorithms in [28].

In this paper, we consider an online version of this well-known problem that is `onlineDISJ`. The definition of the `onlineDISJ` problem is based on the problems that were presented in [28,44]. We provide a c -competitive quantum log-bounded automaton for `onlineDISJ`, and we show that any classical counterpart is less effective. Formally, any log-bounded automaton for the problem is c' -competitive and $c' > c$. A similar result was presented in [44]. This paper presents a more significant separation between the competitive ratios of quantum and classical algorithms.

Often, online algorithms are investigated in terms of advice complexity measure [45–47]. We are interested in this measure also. The model with advice was not considered in [44].

The main point of the model is the following one. An online algorithm gets some bits of advice about an input. A trusted Adviser sending these bits knows the whole input and has

unlimited computational power. Such additional information helps the online algorithm for the computation process and allows to obtain a better competitive ratio. Deterministic and randomized online algorithms with advice are considered in [45,48,49]. Typically, online algorithms researchers use the “Adviser” term, and other models researchers use the “Oracle” term.

In this paper, we provide a quantum log-bounded automaton for onlineDISJ with a single advice bit and a logarithmic size of memory that is almost optimal, i.e., \tilde{c} -competitive for $\lim \tilde{c} = 1$. At the same time, any classical algorithm with the logarithmic size of memory is less effective. Formally, any classical algorithm with the logarithmic size of memory is c'' -competitive and $c'' > 1$.

Note that the usage of a specially constructed problem for demonstrating the power of models is a common approach. Sometimes the problems are artificial, but they allow authors to show more representative results [45,49,50].

The paper is organized as follows. We give definitions in Section 2. In Section 3.1 we present our results for the quantum models and Section 3.2 contains results for classical models. The conclusion is presented in Section 4.

2. Preliminaries

In the paper, we use different notations for δ -functions. A function $\delta : \mathbb{R} \times \mathbb{R} \rightarrow \{0, 1\}$ is such that $\delta(a, b) = 1$ iff $a = b$. A function δ_x is such that $\delta_x = 1$ if $x \neq 0$; otherwise, $\delta_x = 0$. So, $\delta_x = 1 - \delta(x, 0)$.

Let us define the computational models.

An online minimization problem is presented by a set of possible inputs \mathcal{I} and a cost function. An input $I = (x_1, \dots, x_n)$ is a sequence of input variables. Here n is a length of the input, shortly $|I| = n$. Let $\mathcal{O}(I)$ be a set of feasible outputs associated with an input I . An output is a sequence of output variables $O = (y_1, \dots, y_n)$. The cost function assigns a positive real value $cost(I, O)$ to an input $I \in \mathcal{I}$ and an output $O \in \mathcal{O}(I)$. We call $O_{opt}(I)$ an optimal solution for $I \in \mathcal{I}$ if it satisfies the next statement

$$O_{opt}(I) = \operatorname{argmin}_{O \in \mathcal{O}(I)} cost(I, O).$$

An online algorithm for this problem is defined as follows. **A deterministic online algorithm** A outputs a sequence $A(I) = (y_1, \dots, y_n)$ such that the variable y_i is computed from input variables x_1, \dots, x_i .

An algorithm A is c -competitive if we can present a constant $\alpha \geq 0$ such that for every positive integer n and for any input $I \in \mathcal{I}$ of size n , the following statement is correct

$$cost(I, A(I)) \leq c \cdot cost(I, Opt(I)) + \alpha.$$

Here $Opt(I)$ is an output of an optimal offline algorithm for the considered problem; c is the minimal number that satisfies the inequality. The number c is the **competitive ratio** of the algorithm A . In a case of $\alpha = 0$ and $c = 1$, the algorithm A is optimal. Typically c is constant. At the same time, c can be a function on n .

We can define **An online algorithm A with advice** as a sequence of algorithms $A = (A^0, \dots, A^{2^b-1})$ where $b = b(n)$. The trusted adviser chooses $\phi \in \{0, \dots, 2^b-1\}$ that depends on an input I and the algorithm A^ϕ computes an output sequence $A^\phi(I) = (y_1, \dots, y_n)$ such that $y_i = y_i(x_1, \dots, x_i)$. An algorithm A is c -competitive with advice complexity $b = b(n)$ if there exists a constant $\alpha \geq 0$ such that, for every n and for any input I of size n , there exists some $\phi \in \{0, \dots, 2^b-1\}$ such that $cost(I, A^\phi(I)) \leq c \cdot cost(I, Opt(I)) + \alpha$.

A randomized online algorithm R computes an output sequence $R^\psi(I) = (y_1, \dots, y_n)$ such that y_i is computed from ψ, x_1, \dots, x_i , where ψ is a 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 $\alpha \geq 0$ such that, for every I , $\mathbb{E} [cost(I, R^\psi(I))] \leq c \cdot cost(I, Opt(I)) + \alpha$.

We use one-way automata for online minimization problems as online algorithms with restricted memory size. In the paper, we use the terminology for branching programs [51] and algorithms. We say that an automaton computes a Boolean function f_m if for any input X of length m , the automaton returns 1 iff $f_m(X) = 1$. We can say that an automaton returns 1 if it accepts an input word and returns 0 otherwise. Additionally, we use the terminology on memory from algorithms. We say that an automaton has s bits of memory if it has 2^s states. In fact, the model that we consider is a stable id-OBDD [30,31,51], that is an algorithm that uses at most s bits of memory and processes a data stream.

Let us present the definitions of log-bounded automata that we use. A **deterministic log-bounded automaton** with $s = s(n)$ bits of memory that processes an input $I = (x_1, \dots, x_n)$ is a 4-tuple $(d_0, D, \Delta, Result)$.

- The set D is a set of states, $|D| = 2^s$, $d_0 \in D$ is the initial state.
- The function Δ is a transition function $\Delta : \{0, \dots, \gamma - 1\} \times D \rightarrow D$, where γ is a size of the input alphabet.
- $Result$ is an output function $Result : D \rightarrow \{0, \dots, \beta - 1\}$, where β is a size of the output alphabet.

It is a uniform model. The computation starts from the state d_0 . Then, on reading an input symbol x_j , it changes the current state $d \in D$ to $\Delta(x_j, d)$. At the end of the computation, the automaton outputs $Result(d)$.

A **probabilistic log-bounded automaton** is a probabilistic counterpart of the model. It chooses from more than one transition in each step such that each transition is associated with a probability. Thus, the automaton can be in a probability distribution over states during the computation. The total probability must be 1, i.e., the sum of probabilities of outgoing transitions from a single state for a single letter must be 1. Thus, a probabilistic automaton returns a result for each input with some probability. For $v \in \{0, \dots, \beta - 1\}$, the automaton returns a result v for an input with bounded-error if the automaton returns the result v with probability at least $1/2 + \varepsilon$ for some $\varepsilon \in (0, 1/2]$. The automaton computes a function $f : \{0, \dots, \gamma - 1\}^n \rightarrow \{0, \dots, \beta\}$ with bounded error if it returns $f(X)$ with bounded error for each $X \in \{0, \dots, \gamma - 1\}^n$. The automaton computes a function f exactly if $\varepsilon = 0.5$.

A **deterministic online streaming algorithm** (a deterministic log-bounded automaton for online minimization problem) with $s = s(n)$ bits of memory that process input $I = (x_1, \dots, x_n)$ is a 4-tuple $(d_0, D, \Delta, Result)$.

- The set D is a set of states, $|D| = 2^s$, $d_0 \in D$ is the initial state.
- Δ is a transition function $\Delta : \{0, \dots, \gamma - 1\} \times D \rightarrow D$.
- $Result$ is an output function $Result : D \rightarrow \{0, \dots, \beta - 1\}$.

The computation starts from the state d_0 . Then on reading an input symbol x_j it change the current state $d \in D$ to $\Delta(x_j, d)$ and outputs $Result(d)$. The main difference between an “online streaming algorithm” and a standard “online algorithm” definition is a restriction on memory and forbidding reading the previous input variables.

A **randomized online streaming algorithm** and a **deterministic online streaming algorithm with advice** have similar definitions, but with respect to definitions of corresponding models of online algorithms.

Comment. Note that any online algorithm can be simulated by an online streaming algorithm (automaton for online minimization problem) using $O(n)$ bits of memory. It is n bits in the case of binary input and $\lceil n \log_2 \gamma \rceil$ bits in the general case.

Let us consider a **quantum online streaming algorithm** (quantum log-bounded automaton for online minimization problem). More information on quantum computation and quantum streaming computational models (automata) can be found in [17,35,52–54]. For some integer $n > 0$, a quantum online algorithm Q with q qubits is defined on an input $I = (x_1, \dots, x_n) \in \{0, \dots, \gamma - 1\}^n$ and outputs $(y_1, \dots, y_n) \in \{0, \dots, \beta - 1\}^n$. A memory of the quantum algorithm is a state of a quantum register of q qubits. In other words, the computation of Q on an input I can be traced by a 2^q -dimensional vector from Hilbert space

over the field of complex numbers. The initial state is a given 2^q -vector $|\psi\rangle_0$ (we use Dirac notation for vectors). The algorithm can perform one of two kinds of operators. The first one is applying a unitary $2^q \times 2^q$ -matrix. The second one is measurement.

Let us describe a measurement process. Suppose that Q is in a state $|\psi\rangle = (v_1, \dots, v_{2^q})$ before a measurement and the algorithm measures the i -th qubit. Let 0-value for the qubit i is associated with states $a_1^0, \dots, a_{2^{q-1}}^0$; let 1-value for the qubit i is associated with states $a_1^1, \dots, a_{2^{q-1}}^1$. If we measure the qubit, then we obtain 1-result with probability $\text{Pr}_1 = \sum_{j=1}^{2^{q-1}} |v_{a_j^1}|^2$ and we obtain 0-result with probability $\text{Pr}_0 = 1 - \text{Pr}_1$. If the result of the measurement is $u \in \{0, 1\}$, then after measurement, the quantum system is converted to the state $|\psi(u)\rangle$ such that $v_{a_j^u}(u) = v_{a_j^u} / \sqrt{\text{Pr}_u}$, where $v_{a_j^u}(u)$ is an element of the new vector and $v_{a_j^u}$ is an element of the old vector. All other elements of the new vector are 0.

Suppose, on j -th step, the algorithm measures v qubit. Then, an outcome of the measurement is $u \in \{0, \dots, 2^v - 1\}$. We test x_j variable on j -th step, for $j \in \{1, \dots, n\}$. The algorithm does all three following operations in the presented order.

1. It does a measurement. Let the outcome be u .
2. The algorithm applies a unitary $2^q \times 2^q$ -matrix $G^{x_j, u}$ that depends on the input variable x_j and the outcome u .
3. It does a measurement. Let the outcome be u' . Then, the algorithm outputs $\text{Result}(u')$ on this step. Here $\text{Result} : \{0, \dots, 2^q - 1\} \rightarrow \{0, \dots, \beta - 1\}$ is a function that converts the result of the measurement to an output variable.

The algorithm Q is c -competitive in expectation if there exists a non-negative constant α such that, for every I , $\mathbb{E}[\text{cost}(I, Q(I))] \leq c \cdot \text{cost}(I, \text{Opt}(I)) + \alpha$.

A quantum log-bounded automaton has a similar definition, but it returns $\text{Result}(u)$ at the end of the computation. See [35,55] for more details on quantum automata.

3. On Quantum Online Streaming Algorithm for the Online Disjointness Problem

Let us define $\text{onlineDISJ}_{t,k,r,w}$ problem that is based on the Disjointness Boolean function. The $\text{DISJ} : \{0, 1\}^b \times \{0, 1\}^b \rightarrow \{0, 1\}$ function is such that

$$\text{DISJ}(\mathcal{S}, \mathcal{U}) = \neg \bigvee_{i=1}^b \mathcal{S}_i \wedge \mathcal{U}_i.$$

We can say that \mathcal{S} and \mathcal{U} are bit-masks or characteristic vectors for sets $A, B \subset \{1, \dots, b\}$. The function is such that $\text{DISJ}(\mathcal{S}, \mathcal{U}) = 1$ iff the corresponding have no intersections, i.e., $A \cap B = \emptyset$. For some positive integers t, k, r , and w such that $k \bmod t = 0, r < w$, the online minimization problem $\text{onlineDISJ}_{t,k,r,w}$ is organized as follows. Figure 1 describes the structure of an input.

Let m be an positive integer. Formally, an input $I = (x_1, \dots, x_n) \in \{0, \dots, 6\}^n$ has the following structure: $I = \text{bin}(m) \ 6 \ L^1 \ 6 \ L^2 \ 6 \ \dots \ 6 \ L^k$, where $\text{bin}(m)$ is a binary representation of m and $L^i \in \{0, \dots, 5\}^*$, where $1 \leq i \leq k$. For some positive integer n , let $|I| = n$. Let $\text{MDISJ}^m : \{0, \dots, 5\}^* \rightarrow \{0, 1\}$ be a function such that $\text{MDISJ}^m(L^i) = 1$ iff the following properties are right:

1. For some positive integer m_i and binary strings $\mathcal{Z}^{i,j}, \mathcal{U}^{i,j}, \mathcal{S}^{i,j}, M^i$, the structure of L^i is $L^i = M^i \ 5 \ \mathcal{S}^{i,1} \ 2 \ \mathcal{U}^{i,1} \ 3 \ \mathcal{Z}^{i,1} \ 4 \ \dots \ 4 \ \mathcal{S}^{i,m_i} \ 2 \ \mathcal{U}^{i,m_i} \ 3 \ \mathcal{Z}^{i,m_i} \ 4$;
2. $\mathcal{S}^{i,1} = \dots = \mathcal{S}^{i,m_i}, \ \mathcal{U}^{i,1} = \dots = \mathcal{U}^{i,m_i}$ and $\mathcal{Z}^{i,1} = \dots = \mathcal{Z}^{i,m_i}$;
3. $\mathcal{S}^{i,1} = \mathcal{Z}^{i,1}$;
4. $|\mathcal{S}^{i,1}| = |\mathcal{U}^{i,1}|$, where $|v|$ is a length of a vector v ;
5. $\text{val}(M^i) = m_i = 2 + 2 \lceil \sqrt{|\mathcal{S}^{i,1}|} \rceil \cdot \lceil \log_2 |\mathcal{S}^{i,1}| \rceil^2$, where M^i is a binary representation of an integer $\text{val}(M^i)$.
6. The value of the function $\text{DISJ}(\mathcal{S}^{i,1}, \mathcal{U}^{i,1}) = 1$.
7. $m_i = m$.

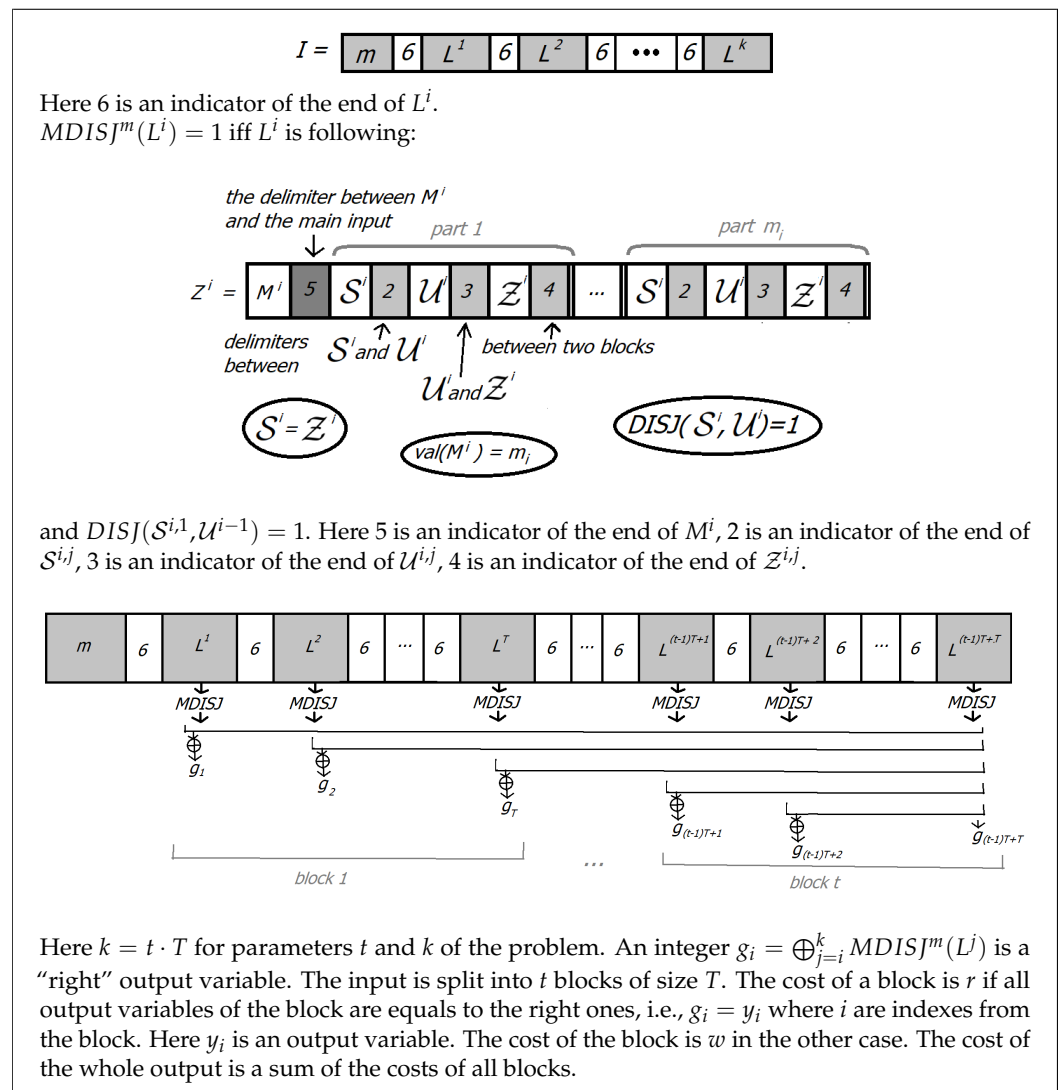


Figure 1. The structure of an input for $onlineDISJ_{t,k,r,w}$.

The authors of [28] have constructed a problem using a similar idea.

For $i \in \{1, \dots, k\}$, let a Boolean variable g_i be such that $g_i = \bigoplus_{j=i}^k MDISJ^m(L^j)$. Let $O' = (y_1, \dots, y_k)$ be output variables that correspond to input variables $x_j = 6$. The cost function is

$$cost(I, O') = tw + (r - w) \cdot \sum_{q=0}^t \prod_{j=q \cdot T+1}^{q \cdot T+T} \delta(g_j, y_j),$$

where $T = k/t$. Recall, parameters t, k, w and r or the problem satisfy conditions $k \bmod t = 0$ and $r < w$. Let us discuss the meaning of the cost function. The output is split into t blocks. The size of each block is T . We say that a block is “right” if all each element y_j from the block satisfies $y_j = g_j$. Otherwise it is a “wrong” block. The cost of a “right” block is r , and the cost of a “wrong” block is w . The sum of costs of all blocks is the total cost of the output. Our goal is to develop an algorithm that minimizes the cost of the output. A similar concept of the cost function was used, for example, in [49].

3.1. A Quantum Online Streaming Algorithm for the Online Disjointness Problem

Let us discuss a quantum log-bounded automaton for $MDISJ$ function. For construction of the automaton, we will use the following existing result:

Lemma 1 ([56]). We can construct a quantum log-bounded automaton that checks the equality of two binary strings with bounded error ϵ' for $\epsilon' > 0$. The size of the memory of the automaton is $O(\log d - \log \epsilon')$.

Note that the proof of the following lemma is similar to the proof from [44]. At the same time, the structure of $MDISJ^m(L^i)$ differs from the structure of the corresponding function from [44]. That is why we present the full proof here. The current form of the function allows us to obtain better error probability.

Lemma 2. Suppose that we have a quantum register $|\mu\rangle$ that stores m , i.e., $|\mu\rangle = |m\rangle$. There is a bounded error quantum log-bounded automaton for $MDISJ^m(L^i)$ that uses $O(\log d)$ qubits of memory and has $O(\frac{1}{d})$ error probability for $d = |L^i|$, where $|L^i|$ is a length of L^i .

Proof. As proof, we construct a quantum automaton for the $MDISJ^m(L^i)$ problem. The automaton consists of five parallel procedures. A procedure per property from the following list:

1. For an integer b , the string L^i have the following structure:

$$L^i = \{0, 1\}^* 5(\{0, 1\}^b 2\{0, 1\}^b 3\{0, 1\}^b 4)^*.$$

Here, we use $*$ for several repetitions of the string.

2. The string M^i satisfies $val(M^i) = m_i = 2 + 2\lceil \sqrt{|S^{i,1}}| \rceil \cdot \lceil \log_2 |S^{i,1}| \rceil^2$ and $m = m_i$;
3. $S^{i,1} = \dots = S^{i,m_i}, U^{i,1} = \dots = U^{i,m_i}$ and $Z^{i,1} = \dots = Z^{i,m_i}$;
4. $S^{i,1} = Z^{i,1}$;
5. $DISJ(S^{i,1}, U^{i,1}) = 1$.

Property 1. We use a deterministic procedure for checking the first property. The size of memory is $O(\log d)$. Firstly, we check whether the strings have the following structure

$$\{0, 1\}^* 5(\{0, 1\}^* 2\{0, 1\}^* 3\{0, 1\}^* 4)^*.$$

We can do it using a constant number of states. Secondly, it checks that all binary parts have the same length and stores its length in a quantum register $|\lambda'\rangle$. We can use $O(d)$ states or $O(\log d)$ memory. The detailed implementation is presented in Appendix A.1.

Property 2. We use a deterministic procedure for checking the second property. We store the M^i string and compute a number of 4s and 5s that is m_i . Then, we check whether $m = val(M^i) = m_i = 2 + 2\lceil \log_2 \sqrt{b} \rceil^2 \cdot \lceil \sqrt{b} \rceil$ is correct, where $b = |S^{i,1}|$. The algorithm stores M^i and counts m_i using $O(d)$ states or $O(\log d)$ bits of memory. The algorithm for the first property already stored b and m is stored in $|\mu\rangle$, that is why we can use them for checking the equality. The detailed implementation is presented in Appendix A.2.

Property 3. We use a quantum fingerprinting technique [56,57] based on a quantum procedure for checking the third property. We assemble two strings from the existing ones:

$$s^1 = (S^{i,1} \circ U^{i,1} \circ Z^{i,1}) \circ (S^{i,2} \circ U^{i,2} \circ Z^{i,2}) \circ \dots \circ (S^{i,m_i-1} \circ U^{i,m_i-1} \circ Z^{i,m_i-1}),$$

$$s^2 = (S^{i,2} \circ U^{i,2} \circ Z^{i,2}) \circ (S^{i,3} \circ U^{i,3} \circ Z^{i,3}) \circ \dots \circ (S^{i,m_i} \circ U^{i,m_i} \circ Z^{i,m_i}),$$

where “ \circ ” is concatenation. If $s^1 = s^2$, then

$$S^{i,1} = S^{i,2}, \quad S^{i,2} = S^{i,3}, \quad \dots, \quad S^{i,m_i-1} = S^{i,m_i},$$

$$U^{i,1} = U^{i,2}, \quad U^{i,2} = U^{i,3}, \quad \dots, \quad U^{i,m_i-1} = U^{i,m_i}$$

and

$$Z^{i,1} = Z^{i,2}, \quad Z^{i,2} = Z^{i,3}, \quad \dots, \quad Z^{i,m_i-1} = Z^{i,m_i}.$$

Let us choose $\epsilon' = 1/d$. Due to Lemma 1, we can present a log-bounded automaton for checking equality of s^1 and s^2 . The size of memory is $O(\log d)$. The detailed implementation is presented in Appendix A.3.

Property 4. We use a quantum procedure for checking the fourth property. The procedure is based on the quantum fingerprinting method also and checks whether $\mathcal{S}^{i,1} = \mathcal{Z}^{i,1}$ is correct. We construct an automaton using Lemma 1 for checking the property with bounded error ϵ'' . Suppose $\epsilon'' = 1/d$, then the automaton reaches an error probability $O(\frac{1}{d})$ using $O(\log d)$ qubits. The detailed implementation is presented in Appendix A.4.

Property 5. We use a quantum procedure for checking the fifth property, and it uses some ideas from [28]. As a base, we use the well-known Grover’s algorithm [58,59] for the unstructured search problem. The size of quantum memory for the procedure is $O(\log d)$ and error probability is $O(1/b)$. Let us present the description of the procedure. Let $\mathcal{S} = \mathcal{S}^{i,1}$ and $\mathcal{U} = \mathcal{U}^{i,1}$. If the input satisfies the first four properties, then we can say that the input is m copies of $\mathcal{S} \circ \mathcal{U} \circ \mathcal{S}$ string. The Grover’s Search algorithm was defined for Query model [16,17] and it does $O(\sqrt{b})$ queries to the oracle or $O(\sqrt{b} \log b)$ queries in a case of unknown number of solutions. We simulate a query to the oracle by reading the input string $\mathcal{S} \circ \mathcal{U} \circ \mathcal{S}$. So, $O(\sqrt{b} \log b)$ copies of this string allow as to provide multiple queries. We simulate the search of the index a_0 such that $\mathcal{S}_{a_0} = \mathcal{U}_{a_0} = 1$ or $\mathcal{S}_{a_0} \wedge \mathcal{U}_{a_0} = 1$. In that case $DISJ(\mathcal{S}^{i,1}, \mathcal{U}^{i,1}) = 0$.

Let us provide the emulation of a query to the oracle. We have $p = \lceil \log_2(b + 1) \rceil$ qubits as a quantum register $|\varphi\rangle$, p qubits as a quantum register $|\varphi_{ones}\rangle$. Additionally, there is a qubit $|\varphi'\rangle$ and a qubit $|\varphi''\rangle$.

Initially, the quantum system is in the $|0\rangle \otimes |0\rangle \otimes |0\rangle \otimes |0\rangle$ state. Firstly, the automaton reads the $\mathcal{S}^{i,2}$ string. In other words, it processes s^2 . We assume that the automaton already knows b on this step. Firstly, we apply the Hadamard transformation for obtaining the following state $|\varphi\rangle \rightarrow \frac{1}{\sqrt{2^p}} \sum_{a=0}^{2^p-1} |a\rangle$. Then, we apply the inversion transformation (X-gate or NOT-gate) for obtaining the following state $|\varphi_{ones}\rangle \rightarrow |1\rangle$.

We split the automaton’s steps to macro-steps. The l -th one is reading $\mathcal{S}^{i,l}, \mathcal{U}^{i,l}, \mathcal{Z}^{i,l}$ strings for $l \in \{2, \dots, m\}$. During checking the third property, we store l in $|\varphi''\rangle$. The computing process of l is presented in Appendix A.3. Assume that the automaton reads $\mathcal{S}^{i,l}$ string’s a -th symbol. If the automaton reads 1, then we apply the V transformation to registers $|\varphi\rangle \otimes |\varphi'\rangle$. Here $V : |a\rangle \otimes |u\rangle \rightarrow |a\rangle \otimes |u \oplus 1\rangle$ for $1 \leq a \leq b$. The automaton does not change the quantum registers if it reads 0.

Assume that the automaton reads $\mathcal{U}^{i,l}$ string’s a -th symbol. If the automaton reads 1, then we apply the W transformation to registers $|\varphi\rangle \otimes |\varphi'\rangle$. Here $W : |a\rangle \otimes |u\rangle \rightarrow (-1)^u |a\rangle \otimes |u\rangle$ for $1 \leq a \leq b$. The automaton does not change the quantum registers if it reads 0. In other words, the amplitude for a state $|a\rangle$ is inverted only if $\mathcal{S}_a = \mathcal{U}_a = 1$. Indexes with such property are our solution. We process the $\mathcal{Z}^{i,l}$ string exactly like we process the $\mathcal{S}^{i,l}$ string.

Remember that we already tested the equality $\mathcal{Z}^{i,l} = \mathcal{S}^{i,l}$ in procedure 3 and procedure 4. Therefore, after processing $\mathcal{Z}^{i,l}$, the effect that $\mathcal{S}^{i,l}$ does to the second register is removed. One macro step simulates one query of Grover’s search algorithm.

Then, we apply Grover’s diffusion transformation to the register $|\varphi\rangle$. The transformation is $D = H^{\otimes p} R H^{\otimes p}$. It rotates all amplitudes near the mean, where R is a unitary transformation that inverts sign for all basis states except zero basis state. Reader can find details on the Grover’s diffusion transformation D in [58,59].

One macro-step (processing of strings $\mathcal{S}^{i,l}, \mathcal{U}^{i,l}, \mathcal{Z}^{i,l}$) and the Grover’s diffusion transformation together correspond to one step of Grover’s search algorithm. Due to [59], the automaton simulate $\frac{\pi}{4} \sqrt{N/v}$ steps of the algorithm if $v = |\{a : \mathcal{S}_a^{i,l} = \mathcal{U}_a^{i,l} = 1\}|$, where $\mathcal{S}_a^{i,l}$ is a -th symbol of the string $\mathcal{S}^{i,l}$. The automaton does $\frac{\pi}{4} \sqrt{N/v}$ steps if $|\varphi_{ones}\rangle = |v\rangle$, where $1 \leq v \leq b$. Initially, $v = 1$. The automaton simulates the required number of Grover’s search steps and measures $|\varphi\rangle |\varphi_{ones}\rangle$. If we obtain $|a\rangle |v\rangle$ result, then we reads one more block $\mathcal{S}^{i,l} \mathcal{U}^{i,l} \mathcal{Z}^{i,l}$ and check the equality $\mathcal{S}_a^{i,l} = \mathcal{U}_a^{i,l} = 1$. For checking the equality, we store

these two bits in $|\varphi'\rangle \otimes |\varphi''\rangle$ qubits. Then, we measure both qubits and check equality of measurement results.

If the condition $S_a^{i,l} = U_a^{i,l} = 1$ is right, then process stops. Otherwise, we assign $|2 \cdot v\rangle$ to the $|\varphi_{ones}\rangle$, initialize $|\varphi'\rangle \otimes |\varphi''\rangle$ by $|0\rangle \otimes |0\rangle$ and repeat the procedure for the new value of v . The probability of correct answer of the algorithm is at least 0.5, due to [59]. Using the standard success boosting procedure, we can get the error probability $1/b^2$ by repeating the algorithm $2\lceil \log_2 b \rceil$ times.

It is easy to see that the size of memory for presented automaton for $MDISJ$ is $O(\log d) = O(\log n)$ qubits. The error probability is $O(\varepsilon)$ where $0 < \varepsilon \leq 1 - (1 - 1/d)^2(1 - 1/b^2) \leq 1 - (1 - 1/d)^3 = O(1/d)$. \square

If we compare the obtained result with results of [28,44], then we can see that the function and the automaton have several modifications. Using this modification we show that the error probability is $O(\frac{1}{d})$. At the same time, similar constructions from [28,44] allow us to obtain only constant error probability.

Using the previous result, we can construct a quantum online streaming algorithm for `onlineDISJ` with a smaller competitive ratio compared to [44]. This property is presented in the following theorem.

Theorem 1. *Suppose $k = O(\log n)$. We provide a quantum online streaming algorithm Q for `onlineDISJ` $_{t,k,r,w}$ that is expected c -competitive, uses $O(\log n)$ qubits of memory, where $c \leq C_Q$. Here $C_Q \rightarrow 0.5(r + w)/r$ for $n \rightarrow \infty$.*

Proof. Firstly, we store m into the quantum register $|\mu\rangle$ of $O(\log m) = O(\log n)$ qubits. We guess g_1 , we choose it randomly and uniformly from the set $\{0, 1\}$. We output the result of guessing as the first output variable y_1 . After that the automaton (streaming online algorithm) computes $MDISJ^m(L^1)$ and outputs y_2 that is $y_2 = y_1 \oplus MDISJ^m(L^1)$. On the i -th step the automaton computes $MDISJ^m(L^{i-1})$ and outputs $y_i = y_{i-1} \oplus MDISJ^m(L^{i-1})$. If $y_1 = g_1$ (we guess g_1 correctly), then for each other output variable is correct, i.e., $y_i = g_i$ for $i \in \{1, \dots, k\}$. If $y_1 \neq g_1$, then all output variables are incorrect, i.e., $y_i \neq g_i$ for $i \in \{1, \dots, k\}$. Let $C_Q = (0.5(1 - \varepsilon)^{t-1} \cdot (r - w) + w)/r$. We can show that $c \leq C_Q$ for $\varepsilon = O(\frac{\log n}{n})$. Let us compute a cost of the output for this algorithm if a probability of computing $MDISJ^m(L^1)$ is ε . Let us remind that all output variables y_i are split to t blocks. Let us change the cost function. The new one $cost'(I, O)$ is such that 1 is the cost of a “right” block, and 0 is the cost of a “wrong” block. Formally,

$$cost'(I, O) = \sum_{r=0}^t \prod_{j=r \cdot (T-1)}^{r \cdot T-1} \delta(g_j, y_j), \text{ for } T = k/t.$$

The main cost function can be computed by the new one using the following statement $cost^t(I, O) = t \cdot w + cost'(I, O) \cdot (r - w)$.

For computing the competitive ratio, we should calculate $\mathbb{E}[cost'(I, O)]$.

Let us compute p_i the probability that a block i is a “right” block (costs 1). Let $i = 1$. So, if the i -th block is “right”, then for all $T - 1$ binary strings L^j inside the block the computation of $MDISJ$ is right and a guess of y_1 is right. A probability of this event is $p_1 = 0.5 \cdot (1 - \varepsilon)^{T-1}$.

Let $i > 1$. If the i -th block is “right”, then two conditions should be true:

- (i) for all $T - 1$ binary strings L^j inside the block, the computation of $MDISJ$ should be right.
- (ii) If we consider a number of errors for computation of $MDISJ$ for preceding L^j s plus 1 if y_1 is wrong. Then this number should be even.

A probability of the first condition is $(1 - \varepsilon)^{T-1}$. Let us compute the probability of the second condition.

Let $E(j)$ be the number of errors before computation of $MDISJ^m(L^j)$. Let λ_j be the number of errors during computation of $MDISJ$. If $y_1 \neq g_1$, then $E(j) = \lambda_j + 1$,

otherwise $E(j) = \lambda_j$. Let the probability $F(j) = Pr\{E(j) \bmod 2 = 0\}$. Hence, $Pr\{E(j) \bmod 2 = 1\} = 1 - F(j)$. If we compute $MDISJ^m(L^j)$ with an error, then we have $E(j - 1) \bmod 2 = 1$. If we compute $MDISJ^m(L^j)$ with no error, then we have $E(j - 1) \bmod 2 = 0$. Hence,

$$F(j) = F(j - 1)(1 - \epsilon) + (1 - F(j - 1))\epsilon = \epsilon + (1 - 2\epsilon)F(j - 1).$$

Remember that $y_1 = g_1$ with probability $\frac{1}{2}$. Hence, $F(1) = \frac{1}{2}$.

We can compute the probability $F(j)$ as follows

$$\begin{aligned} F(j) &= \epsilon + (1 - 2 \cdot \epsilon) \cdot F(j - 1) = \\ &\epsilon + \epsilon \cdot (1 - 2 \cdot \epsilon) + (1 - 2 \cdot \epsilon)^2 \cdot F(j - 2) = \\ &\dots = \epsilon + \epsilon \cdot (1 - 2 \cdot \epsilon) + \dots + \epsilon \cdot (1 - 2 \cdot \epsilon)^{j-2} + (1 - 2 \cdot \epsilon)^{j-1} \cdot F(j - j + 1) = \\ &\epsilon \cdot \sum_{l=0}^{j-2} (1 - 2 \cdot \epsilon)^l + (1 - 2 \cdot \epsilon)^{j-1} \cdot F(1) = 0.5 \cdot (1 - (1 - 2 \cdot \epsilon)^{j-1}) + 0.5 \cdot (1 - 2 \cdot \epsilon)^{j-1} = \frac{1}{2} \end{aligned}$$

Hence, $p_i = 0.5 \cdot (1 - \epsilon)^{T-1}$. Let us compute the expected cost:

$$\mathbb{E} [cost^t(I, A(I))] = \sum_{i=1}^t (p_i \cdot 1 + (1 - p_i) \cdot 0) = \sum_{i=1}^t p_i = 0.5 \cdot (1 - \epsilon)^{T-1} \cdot t.$$

Therefore, $\mathbb{E} [cost^t(I, A(I))] = 0.5 \cdot (1 - \epsilon)^{T-1} \cdot t(r - w) + tw$.

Let us compute the expected competitive ratio.

$$c \leq \frac{0.5 \cdot (1 - \epsilon)^{T-1} \cdot t(r - w) + tw}{tr} = \left(0.5 \cdot (1 - \epsilon)^{T-1} \cdot (r - w) + w\right) / r = C_Q.$$

Let us estimate $\lim_{n \rightarrow \infty} C_Q$.

$$\begin{aligned} \frac{w + (1 - \epsilon)^{k-1} \cdot \frac{1}{2} \cdot (r - w)}{r} &\leq C_Q \leq \frac{w + (1 - \epsilon) \cdot \frac{1}{2} \cdot (r - w)}{r} \\ \lim_{n \rightarrow \infty} \frac{w + (1 - \epsilon)^{k-1} \cdot \frac{1}{2} \cdot (r - w)}{r} &\leq \lim_{n \rightarrow \infty} C_Q \leq \lim_{n \rightarrow \infty} \frac{w + (1 - \epsilon) \cdot \frac{1}{2} \cdot (r - w)}{r} \\ \lim_{n \rightarrow \infty} \frac{w + (1 - \frac{\log n}{n})^{\log n} \cdot \frac{1}{2} \cdot (r - w)}{r} &\leq \lim_{n \rightarrow \infty} C_Q \leq \lim_{n \rightarrow \infty} \frac{w + (1 - \frac{\log n}{n}) \cdot \frac{1}{2} \cdot (r - w)}{r} \\ (0.5(r - w) + w) / r &\leq \lim_{n \rightarrow \infty} C_Q \leq (0.5 \cdot (r - w) + w) / r \end{aligned}$$

Hence, $\lim_{n \rightarrow \infty} C_Q = \frac{0.5(r+w)}{r}$. \square

The quantum streaming algorithm with advice is presented in the following theorem.

Theorem 2. *There is the expected c -competitive quantum online streaming algorithm Q with $O(\log n)$ qubits of memory and single advice bit for $\text{onlineDISJ}_{t,k,r,w}$, where $c \leq C_{QA}$ and $C_{QA} \rightarrow 1$ for $k = O(\log n)$ and $n \rightarrow \infty$.*

Proof. The algorithm is the same as in the Theorem 1, but y_1 is obtained as an advice bit. Let us compute p_i similarly as in the proof of the previous theorem. Assume that $\epsilon = O(\frac{\log n}{n})$ is an error for computing $MDISJ$. Let $E(j)$ be the number of errors before computation of $MDISJ^m(L^j)$. It is a number of errors for computation of $MDISJ$ for preceding L s. Let the probability $F(j) = Pr\{E(j) \bmod 2 = 0\}$. Hence, $Pr\{E(j) \bmod 2 = 1\} = 1 - F(j)$. Remember that we obtain y_1 as the advice bit; therefore, $F(1) = 1$.

$$F(j) = \epsilon + (1 - 2 \cdot \epsilon) \cdot F(j - 1) =$$

$$\begin{aligned} \dots &= \varepsilon \cdot \sum_{l=0}^{j-2} (1 - 2 \cdot \varepsilon)^l + (1 - 2 \cdot \varepsilon)^{j-1} \cdot F(1) = \\ &0.5 \cdot (1 - (1 - 2 \cdot \varepsilon)^{j-1}) + (1 - 2 \cdot \varepsilon)^{j-1} = 0.5 \cdot (1 + (1 - 2\varepsilon)^{j-1}). \end{aligned}$$

So, $F(j) = \frac{(1-2\varepsilon)^{j-1} + 1}{2}$.

The probability p_i of the event is $p_i = 0.5 \cdot (1 + (1 - 2\varepsilon)^{(i-1)T+1-1}) \cdot (1 - \varepsilon)^{T-1}$. We can compute the expected value for the cost

$$\begin{aligned} \mathbb{E} [\text{cost}'(I, O)] &= \sum_{i=1}^t (1 \cdot p_i + 0 \cdot (1 - p_i)) = p_1 + \sum_{i=2}^t p_i = \\ &(1 - \varepsilon)^{T-1} \cdot \left(1 + \sum_{i=2}^t (0.5 + 0.5(1 - 2\varepsilon)^{(i-1)T}) \right) = \\ &0.5(1 - \varepsilon)^{T-1} \cdot \left(t + 1 + \sum_{i=2}^t (1 - 2\varepsilon)^{(i-1)T} \right) \end{aligned}$$

Let us compute

$$\begin{aligned} \lim_{n \rightarrow \infty} \mathbb{E} [\text{cost}'(I, O)] &= \lim_{n \rightarrow \infty} 0.5(1 - \frac{\log n}{n})^{T-1} \cdot \left(t + 1 + \sum_{i=2}^t (1 - 2 \frac{\log n}{n})^{(i-1)T} \right) \leq \\ &\leq \lim_{n \rightarrow \infty} 0.5(1 - \frac{\log n}{n}) \cdot \left(t + 1 + \sum_{i=2}^t (1 - 2 \frac{\log n}{n}) \right) = 0.5(1 - 0) \cdot (t + 1 + t - 1) = t \end{aligned}$$

Therefore, $\lim_{n \rightarrow \infty} C_{QA} \leq \mathbb{E} [\text{cost}'(I, O)]/t = t/t = 1$ and $C_{QA} \geq 1$. Hence, we obtain the claim of the theorem. \square

3.2. Bounds for Classical Automata for the Online Disjointness Problem

We use some results from the communication complexity theory. We briefly remind the reader of the model (see [60]). There are two players, Alice and Bob. They want to compute a Boolean function $f(X)$. Alice has variables from a set $X_A \subset X$ and Bob variables from a set $X \setminus X_A$. Alice starts computation and sends a message to Bob. Then, Bob continues and sends a message to Alice, etc. The player who can compute $f(X)$ returns an answer. Alice and Bob can use a probabilistic mechanism. The probabilistic communication complexity is the total number of bits in all messages. It is known that

Lemma 3 ([42]). *The probabilistic communication complexity of the Boolean function $DISJ(\mathcal{S}, \mathcal{U})$ is $\Omega(b)$, where $\mathcal{S}, \mathcal{U} \in \{0, 1\}^b$, Alice has x and Bob has y .*

Using this result, we can show the following one.

Lemma 4. *Suppose, we have a promise that properties 1–5 and 7 of $MDISJ^m(L^i)$ are correct. Suppose the length of the string $\mathcal{S}^{i,1}$ is $b = |\mathcal{S}^{i,1}|$ and the integer m such that $m < b$. Then, there is no bounded error probabilistic log-bounded automata for $MDISJ^m(L^i)$ that uses $o(b/m)$ bits of memory.*

Proof. Suppose that there is a probabilistic automaton R for $MDISJ^m(L^i)$ with $d = o(b/m)$ bits of memory. Then, we can simulate the work of R by a probabilistic communication protocol such that Alice knows $\mathcal{S}^{i,1}$ and Bob knows $\mathcal{U}^{i,1}$. Each player simulates R on his part of the input. Then he/she sends a state of the memory of R to the opposite player. See more details on an emulation of stream processing models by communication protocols, for example, in [37,51,61]. Therefore, the communication protocol uses $d \cdot m = o(b)$ bits of memory for computing $DISJ(\mathcal{S}^{i,1}, \mathcal{U}^{i,1})$. This fact contradicts with Lemma 3. \square

Using this result, we can show that for probabilistic and deterministic automata, the $\text{onlineDISJ}_{t,k,r,w}$ problem is hard.

Theorem 3. *There is no c -competitive deterministic online streaming algorithm for $\text{onlineDISJ}_{t,k,r,w}$ that uses $o(\sqrt{n})$ bits of memory, where $c < \frac{w}{r}$, $k = O(\log n)$.*

Proof. Let us consider any online streaming algorithm A for the $\text{onlineDISJ}_{t,k,r,w}$ problem such that the size of memory is at most $o(\sqrt{n})$ bits. Assume, the algorithm A returns y_1 as the first output variable. Let us prove that there are two parts of the input $L_0^1, L_1^1 \in \{0, 1\}^d$ such that A returns the same value y_2 for both, but $\text{MDISJ}(L_0^1) = 0, \text{MDISJ}(L_1^1) = 1$.

Assume that there is no such triple (y_2, L_0^1, L_1^1) . Then it means that we can construct an automaton A' that uses $o(\sqrt{n})$ bits of memory and has the following property: $A'(L^1) = A'(L^{1'})$ iff $f(L^1) = f(L^{1'})$, for any $L^1, L^{1'} \in \{0, 1\}^d$. The automaton A' simulates A . Hence, the automaton A' computes the function MDISJ or the function $\neg\text{MDISJ}$. If it computes the function $\neg\text{MDISJ}$, then we can provide an automaton A'' such that $A''(L^1) = \neg A'(L^1)$. This statement contradicts with the main claim of the theorem. Similarly, we can prove that we have analogous triples (y_{i+1}, L_0^i, L_1^i) for $i \in \{2, \dots, k\}$.

Let us choose $\sigma_i = y_i \oplus 1 \oplus \bigoplus_{j=i+1}^k \sigma_j$, for $i \in \{1, \dots, k\}$.

Let I_A be such that $L^i = L_{\sigma_i}^i$. An optimal offline solution is (g_1, \dots, g_k) where $g_i = \bigoplus_{j=i}^k \sigma_j$.

Let us prove that $g_i \neq y_i$ for each $i \in \{1, \dots, k\}$. We have $\sigma_i = y_i \oplus 1 \oplus \bigoplus_{j=i+1}^k \sigma_j$, hence $y_i = 1 \oplus \sigma_i \oplus \bigoplus_{j=i+1}^k \sigma_j = 1 \oplus \bigoplus_{j=i}^k \sigma_j = 1 \oplus g_i$, so we obtain $y_i = \neg g_i$.

Therefore, all output variables are “wrong” and the cost is $\text{cost}^t(I_A, A(I_A)) = tw$. Hence, the competitive ratio c more than $\frac{tw}{tr} = \frac{w}{r}$. \square

Theorem 4. *There is no expected c -competitive randomized online streaming algorithm for $\text{onlineDISJ}_{t,k,r,w}$ that uses $o(\sqrt{n})$ bits of memory, where $k = o(\log n)$ and $c < \frac{1}{2^T} + (1 - \frac{1}{2^T})\frac{w}{r}$, $T = \frac{k}{i}$.*

Proof. We can show that a bounded error randomized online streaming algorithm A cannot compute y_i . We prove the claim using the idea from the proof of Theorem 3. It is easy to see that the only option for an algorithm is to guess the value of y_i and randomly uniformly choose it. Using this strategy, we can obtain the required competitive ratio. If we want to get a cost r for a block, then it should guess all output bits of the block. So, the cost of the i -th block is $c_i = w(1 - \frac{1}{2^T}) + \frac{r}{2^T}$. Hence, $\text{cost}^t(I_A, A(I_A)) = (w(1 - \frac{1}{2^T}) + \frac{r}{2^T}) \cdot t$. Therefore, the algorithm A is c competitive in expectation, for $c \geq t \cdot ((1 - \frac{1}{2^T})w + \frac{r}{2^T}) / (tr) = 2^{-T} + (1 - 2^{-T})w/r$. \square

For proofs of properties for classical models with advice (Theorems 5 and 6), we show that if the model does not have enough memory, then the problem can be interpreted as the “String Guessing, Unknown History” (2-SQUH) from [50].

The following result for the 2-SQUH is known:

Lemma 5 ([50]). *Consider an input string of length k for 2-SQUH, for some positive integer k . Any online algorithm that is correct in more than αk characters, for $0.5 \leq \alpha < 1$, needs to read at least $(1 + (1 - \alpha) \log_2(1 - \alpha) + \alpha \log_2 \alpha)k$ advice bits.*

Using above result we show complexity of the $\text{onlineDISJ}_{t,k,r,w}$ problem.

Theorem 5. *There is no c -competitive deterministic online streaming algorithm for $\text{onlineDISJ}_{t,k,r,w}$ that uses $o(\sqrt{n})$ bits of memory and b advice bits, where $c < \frac{w(t-h)+rh}{tr}$, $T = \frac{k}{i}$, $h = \lfloor \frac{v}{T} \rfloor$, v is such that $b = (\frac{v}{k} \log_2 \frac{v}{k} + (1 - \frac{v}{k}) \log_2(1 - \frac{v}{k}) + 1)k$, $\frac{k}{2} \leq v < k$ and $k = o(\log n)$.*

Proof. We start with a proof of the following claim. For any online algorithm for *MDISJ* that obtains b advice bits, we can suggest an input such that the algorithm has at least $k - b$ errors in computation. We prove it by induction.

Firstly, let us prove the claim for $b = k$. Then the adviser can send (g_1, \dots, g_k) , where $g_i = \bigoplus_{j=i}^k MDISJ^m(L^j)$. Using this advice we return all correct output variables y_1, \dots, y_k .

Secondly, let us show the correctness of the claim in a case of $b = 0$. Therefore, there is no advice at all. So, it is exactly the Theorem 3 situation.

Thirdly, let us show the correctness of the claim in a case of $b \notin \{0, k\}$. Assume that the claim is correct for a pair (b'', k') such that $b' \leq b, k' \leq k$ and at least one of these inequalities is strict. Let us consider the computation process of the $MDISJ^m(L^1)$ function.

Assume that there is an input $L^1 \in \{0, 1\}^d$ such that there is no way to compute an output with bounded error. Then using the input, we obtain the case for $k - 1$ input variables and b advice bits. So, the computation of *MDISJ* for $k - b - 1$ binary strings L^i is incorrect, and the computation for L^1 is incorrect also.

Assume that the algorithm computes an output for $MDISJ^m(L^1)$ with a bounded error always. So we can describe the process of communication with the adviser in the following way: the adviser separates all possible inputs into 2^b non-overlapping groups G_1, \dots, G_{2^b} . After that, he sends a group number containing current input to the algorithm. Then, the algorithm A processes the input with the knowledge that an input can be only from this group.

Let us consider three sets of groups:

- $I_0 = \{G_i : \forall \sigma \in \{0, 1\}^d \text{ such that } \sigma \text{ is } L^1 \text{ and } MDISJ^m(\sigma) = 0\},$
- $I_1 = \{G_i : \forall \sigma \in \{0, 1\}^{m_1} \text{ such that } \sigma \text{ is } L^1 \text{ and } MDISJ^m(\sigma) = 1\},$
- $I_{10} = \{G_1, \dots, G_{2^b}\} \setminus (I_1 \cup I_0).$

Let $|I_a| \neq 0$, for some $a \in \{0, 1\}$. If $|I_a| \leq 2^{b-1}$, then as Z^1 we take any input from any group $G \in I_a$. Hence we have at most 2^{b-1} possible groups for the adviser that distinguish next L^i 's. We can say that the adviser can encode them using $b - 1$ bits. Therefore, we get the situation for $b - 1$ advice bits and $k - 1$ binary strings L^i . The claim is true for this situation. If $|I_a| > 2^{b-1}$, then we pick an input S^1 from any group $G \notin I_a$. Therefore, there are at most 2^{b-1} possible groups for the adviser and the same situation. For this case, the claim is correct.

Suppose the size of sets I_0 and I_1 is $|I_0| = |I_1| = 0$. Assume that the algorithm solves the problem, and the memory size is s' , where $s' < s - b$. The automaton B with the following structure can simulate the algorithm's work on L^1 . The automaton B has two memory registers: a register M_1 of b bits and a register M_2 of s' bits. We assume that M_1 is initialized by advice bits. After that, the automaton B invokes A depending on the value of advice bits stored in M_1 . The memory size of B is $s' + b < s$ and it computes *MDISJ* by simulating A . We obtain a contradiction with the theorem's claim. Hence, transferring the answer of $MDISJ^m(L^1)$ as an advice bit is the only way to compute the result of the function. So, we have a situation for $k - 1$ binary strings L^i and $b - 1$ advice bits.

So, it means that for the algorithm, the problem is the same as the *String Guessing Problem with Unknown History (2-SGUH)* from [50].

Due to Lemma 5, for obtaining v right answers y_i , we need $b = (1 + (1 - \frac{v}{k}) \log_2(1 - \frac{v}{k}) + \frac{v}{k} \log_2 \frac{v}{k})k$ advice bits.

Because of the cost function properties, the best case for the algorithm is getting all y_i 's of a block. Therefore, the algorithm can obtain advice bits on $h = \lfloor \frac{v}{T} \rfloor$ full blocks, for $T = \frac{k}{t}$. In that case, the cost of these blocks is r . Each of the rest blocks has at least one "wrong" output variable. So, such a block costs w . Hence, we suggest the input such that the corresponding output costs

$$\lfloor v/T \rfloor \cdot r + (t - \lfloor v/T \rfloor)w, \text{ for } b = \left(1 + \left(1 - \frac{v}{k}\right) \log_2\left(1 - \frac{v}{k}\right) + \frac{v}{k} \log_2 \frac{v}{k}\right)k.$$

Let us compute the competitive ratio c

$$c \geq \frac{\lfloor v/T \rfloor \cdot r + (t - \lfloor v/T \rfloor)w}{tr}, \text{ for } b = \left(1 + \left(1 - \frac{v}{k}\right) \log_2\left(1 - \frac{v}{k}\right) + \frac{v}{k} \log_2 \frac{v}{k}\right)k.$$

□

Next, we show an analog of Theorem 5 for randomized case. Ideas from [62–65] are the basis for the proof of the following theorem.

Theorem 6. *Suppose $s = o(\sqrt{n})$. Any randomized online streaming algorithm A using b advice bits, less than $s - b$ bits of memory, and solving $\text{onlineDISJ}_{t,k,r,w}$, has the expected competitive ratio*

$$c \geq \frac{hr + \delta_u \cdot (2^{u-T}r + (1 - 2^{u-T})w) + (t - h - \delta_u)(2^{-T}r + (1 - 2^{-T})w)}{tr},$$

for $h = \lfloor \frac{v}{T} \rfloor, T = \frac{k}{t}, u = v - hz, v$ is such that $b = \left(1 + \left(1 - \frac{v}{k}\right) \log_2\left(1 - \frac{v}{k}\right) + \frac{v}{k} \log_2 \frac{v}{k}\right)k, \frac{k}{2} \leq v < k$ and $k = o(\log n)$.

Proof. Let us suggest the proof that is based on the proof of Theorem 5. Suppose that an online algorithm obtains b advice bits. Then, we can suggest an input such that on this input, the computation of $M\text{DISJ}^m(L^i)$ has an error for at least $k - b$ strings L^i . The claim can be proven by the way similar to the proof of Theorem 5. At the same time, we use the probabilistic automaton B to simulate A , and the part of the automaton that uses memory M_2 is probabilistic.

Therefore, for the algorithm, our problem is equivalent to the *String Guessing Problem with Unknown History (2-SGUH)* from [50].

Hence, if we want to obtain v right answers for y_i output variables, then we should use

$$b = k \cdot \left(\frac{v}{k} \log_2 \frac{v}{k} + \left(1 - \frac{v}{k}\right) \log_2\left(1 - \frac{v}{k}\right) + 1\right).$$

We can show that if we do not know y_i from the adviser, then we cannot compute the output variable with bounded error. Hence, the only way to obtain the variables is randomly uniformly choosing them. We can apply the proof technique as in Theorem 4. We use the same approach for all segments between “known” output variables. □

4. Conclusions

We provide an expected \mathcal{C}_Q -competitive quantum online streaming algorithm with no advice and an expected \mathcal{C}_{QA} -competitive quantum online streaming algorithm with a single advice bit. They are such that $\mathcal{C}_Q \rightarrow \frac{r+w}{2r}$ and $\mathcal{C}_{QA} \rightarrow 1$ for $n \rightarrow \infty$. At the same time, any classical algorithm with a logarithmic size of memory is $(\mathcal{C}_Q + \gamma_1)$ -competitive for a model with no advice and $(\mathcal{C}_{QA} + \gamma_2)$ -competitive for a model with $o(\log n)$ advice bits, where $\gamma_1, \gamma_2 > 0$.

Author Contributions: The main idea and lower bounds, K.K.; constructions and concepts, K.K. and A.K. All authors have read and agreed to the published version of the manuscript.

Funding: This paper has been supported by the Kazan Federal University Strategic Academic Leadership Program (“PRIORITY-2030”).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

Appendix A. Details of the Proof for Lemma 2

Appendix A.1. The Implementation of the Procedure for Checking the First Property from Lemma 2

Let us remind the reader of the property that we are checking.

- $L^i = \{0, 1\}^* 5(\{0, 1\}^b 2\{0, 1\}^b 3\{0, 1\}^b 4)^*$ for some integer b , where $*$ means repeating the string several times

We present the description of the procedure as a quantum automaton. The automaton uses the following registers $|\phi'\rangle|\xi'\rangle|\lambda'\rangle|\lambda'_1\rangle$ of $3 + 2\lceil\log_2(d + 1)\rceil$ qubits. The register $|\phi'\rangle$ contains 2 qubits. The possible values are as follows.

- $|\phi'\rangle = |0\rangle$ means the automaton reads a binary string before 5.
- $|\phi'\rangle = |1\rangle$ means the automaton reads a binary string from 4 or 5 to 2.
- $|\phi'\rangle = |2\rangle$ means the automaton reads a binary string from 2 to 3.
- $|\phi'\rangle = |3\rangle$ means the automaton reads a binary string from 3 to 4.

The register $|\xi'\rangle$ contains 1 qubit. The possible values are as follows.

- $|\xi'\rangle = |1\rangle$ means the automaton rejects the input.
- $|\xi'\rangle = |0\rangle$ means the automaton continues computation.

We use the register $|\lambda'\rangle$ of $\lceil\log_2(d + 1)\rceil$ qubits for counting a length of the current binary string. We store a length of the first binary string in the register $|\lambda'_1\rangle$ of $\lceil\log_2(d + 1)\rceil$.

We start the computation from $|0\rangle|0\rangle|0\rangle|0\rangle$ state.

Let us present the behavior of the automaton depending on an input symbol $u \in \{0, \dots, 6\}$. Assume that we have a transformation

$$NEXT' : |\lambda'_1\rangle \rightarrow |(j + 1) \bmod \lceil d + 1 \rceil\rangle.$$

Note that we can implement *if*-operator using control-operators [17]. Let us describe the transition function as an Algorithm A1.

Appendix A.2. The Implementation of the Procedure for Checking the Second Property from Lemma 2

Let us remind the reader of the property that we are checking.

- $val(M^i) = m_i = 2 + 2\lceil\sqrt{|S^{i,1}|}\rceil \cdot \lceil\log_2 |S^{i,1}|\rceil^2$ and $m_i = m$;

We present the description of the procedure as a quantum automaton. The automaton uses the following registers:

- a register $|\phi''\rangle$ of $\lceil\log_2(d + 1)\rceil$ qubits;
- a register $|\xi''\rangle$ of $\lceil\log_2(d + 1)\rceil$ qubits;
- a single qubit $|\lambda''\rangle$;
- a register $|\theta''\rangle$ of $\lceil\log_2(d + 1)\rceil$ qubits.

We start the computation from $|0\rangle|0\rangle|0\rangle|0\rangle$ state.

Let us consider the behavior of the automaton depending on an input symbol v .

Let $v \in \{0, 1\}$. If $|\lambda''\rangle = |0\rangle$, then the automaton reads M^i and stores bits one by one in qubits of $|\xi''\rangle$ using XOR^v operator.

$$XOR^v : |a\rangle \rightarrow |a \oplus v\rangle.$$

The automaton stores the current index in $|\theta''\rangle$ using $NEXT''$

$$NEXT'' : |j\rangle \rightarrow |(j + 1) \bmod (d + 1)\rangle.$$

Let $v = 5$. Then, the automaton converts $|\lambda''\rangle \rightarrow |1\rangle$.

Let $v = 4$. Then, the automaton applies $NEXT''$ transformation to $|\phi''\rangle$.

Let $v \in \{0, 1, 2, 3\}$. Then, the automaton the identity transformation.

Let $v = 6$. Then, the automaton measures $|\phi''\rangle$. If the result of the measurement is u , then we check an equality $u = 2\lceil\sqrt{b}\rceil \cdot \lceil\log_2 \sqrt{b}\rceil^2 + 2$. After that, the automaton measures $|\xi''\rangle|\mu\rangle$ and checks an equality $m = u = val(M_i)$. The last equality means that the string

L^i satisfies the forth condition. If at least one of these two equality are wrong, then the automaton rejects the input.

Recall, at this moment, the automaton already have stored b in the register $|\lambda'\rangle$ and m in the register $|\mu\rangle$.

Let us describe the behavior of the automaton depending on an input symbol $u \in \{0, \dots, 6\}$. Note, that we can implement if -operator using control-operators [17]. Let us describe the transition function as an Algorithm A2.

Algorithm A1 Transition on a symbol $u \in \{0, \dots, 6\}$

```

if  $u \in \{0, 1\}$  and  $|\phi'\rangle \neq |1\rangle$  then
   $|\lambda\rangle \leftarrow NEXT'|\lambda\rangle$ 
end if
if  $u = 5$  then
   $res_{\phi'} \leftarrow MEASURE(|\phi'\rangle)$ 
  if  $res_{\phi'} = 0$  then
     $|\phi'\rangle \leftarrow |1\rangle$ 
  else
     $|\xi'\rangle \leftarrow |1\rangle$  ▷ the automaton rejects the input
  end if
end if
if  $u \in \{2, 3, 4\}$  then
   $res_{\phi'} \leftarrow MEASURE(|\phi'\rangle)$ 
   $res_{\lambda'} \leftarrow MEASURE(|\lambda'\rangle), res_{\lambda'_1} \leftarrow MEASURE(|\lambda'_1\rangle)$ 
  if  $res_{\phi'} = u - 1$  then
     $|\phi'\rangle \leftarrow |((u - 1) \bmod 3) + 1\rangle$ 
  else
     $|\xi'\rangle \leftarrow |1\rangle$  ▷ the automaton rejects the input
  end if
  if  $res_{\lambda'_1} = 0$  then
     $|\lambda'_1\rangle \leftarrow |res_{\lambda'_1}\rangle$ 
  else
    if  $res_{\lambda'_1} \neq res_{\lambda'}$  then
       $|\xi'\rangle \leftarrow |1\rangle$  ▷ the automaton rejects the input
    end if
  end if
   $|\lambda'\rangle \leftarrow |0\rangle$  ▷ Initialization of a length counter
end if
if  $u = 6$  then ▷ End of the input
   $res_{\xi'} \leftarrow MEASURE(|\xi'\rangle), res_{\phi'} \leftarrow MEASURE(|\phi'\rangle)$ 
  if  $res_{\xi'} = 0$  and  $res_{\phi'} = 1$  then
    return "Accept the input"
  else
    return "Reject the input"
  end if
end if

```

Algorithm A2 Transition on a symbol $u \in \{0, \dots, 6\}$

```

if  $u \in \{0, 1\}$  and  $|\lambda''\rangle = |0\rangle$  then
     $|\xi''\rangle[|\theta''\rangle] \leftarrow \text{XOR}^u |\xi''\rangle[|\theta''\rangle]$ 
     $|\theta''\rangle \leftarrow \text{NEXT}'' |\theta''\rangle$ 
end if
if  $u = 4$  then
     $|\phi''\rangle \leftarrow \text{NEXT}'' |\phi''\rangle$ 
end if
if  $u = 6$  then
     $j \leftarrow \text{MEASURE}(|\phi''\rangle), \text{val}(M_i) \leftarrow \text{MEASURE}(|\xi''\rangle)$ 
     $b \leftarrow \text{MEASURE}(|\lambda''\rangle), m \leftarrow \text{MEASURE}(|\mu\rangle)$ 
    if  $j = 2 + 2\lceil \log_2 \sqrt{b} \rceil^2 \cdot \lceil \sqrt{b} \rceil$  and  $m = j = \text{val}(M_i)$  then
        return "Accept the input"
    else
        return "Reject the input"
    end if
end if

```

Appendix A.3. The Implementation of the Procedure for Checking the Third Property from Lemma 2

Let us remind the reader of the property that we are checking.

- $S^{i,1} = \dots = S^{i,m_i}, U^{i,1} = \dots = U^{i,m_i}$ and $Z^{i,1} = \dots = Z^{i,m_i}$;

Let us present the quantum automaton for checking the property. It based on the quantum fingerprinting technique [56,57]. Let us assemble two string:

$$s^1 = S^{i,1} \circ U^{i,1} \circ Z^{i,1} \circ S^{i,2} \circ U^{i,2} \circ Z^{i,2} \circ \dots \circ S^{i,m_i-1} \circ U^{i,m_i-1} \circ Z^{i,m_i-1},$$

$$s^2 = S^{i,2} \circ U^{i,2} \circ Z^{i,2} \circ S^{i,3} \circ U^{i,3} \circ Z^{i,3} \circ \dots \circ S^{i,m_i} \circ U^{i,m_i} \circ Z^{i,m_i},$$

here "o" is concatenation.

If $s^1 = s^2$, then

$$S^{i,1} = S^{i,2}, S^{i,2} = S^{i,3}, \dots, S^{i,m_i-1} = S^{i,m_i},$$

$$U^{i,1} = U^{i,2}, U^{i,2} = U^{i,3}, \dots, U^{i,m_i-1} = U^{i,m_i} \text{ and}$$

$$Z^{i,1} = Z^{i,2}, Z^{i,2} = Z^{i,3}, \dots, Z^{i,m_i-1} = Z^{i,m_i}.$$

For some constant $\epsilon' > 0$, we suggest a quantum algorithm for checking the equality of these strings with bounded error ϵ' . It uses $O(\log d)$ qubits of memory and is based on the automaton from Lemma 1.

Here we describe the main idea of the automaton. A reader can find more details in [36,56,57].

We use the following quantum registers:

- $|\psi\rangle$ is a quantum register of $q = \lceil \log_2(2d/\epsilon') \rceil$ qubits.
- $|\psi_{targ}\rangle$ is a single qubit.
- $|\psi_{ind}\rangle$ is a quantum register of $\lceil \log_2(d+1) \rceil$ qubits. We store an index of a symbol of a string in this register.

A set of special parameters $S = (k_1, \dots, k_{2^q})$ from [56,57] is used by the algorithm.

We start from the $|0\rangle|0\rangle|0\rangle$ state. The automaton do the following transformations if $|\lambda''\rangle = |1\rangle$. This condition means the automaton read and stored M_i .

Firstly, if the automaton reads a symbol 5, then it applies the Hadamard transformation to $|\psi\rangle$. Hence, the quantum system is in the state

$$|0\rangle|0\rangle|0\rangle \rightarrow \frac{1}{\sqrt{2^q}} \sum_{a=0}^{2^q} |a\rangle|0\rangle|0\rangle.$$

For all next steps and symbols 0 or 1, we apply the transformation *NEXT* to $|\psi_{ind}\rangle$ that is

$$|j\rangle \rightarrow |(j + 1) \bmod [d + 1]\rangle.$$

Assume that the automaton reads the j -th symbol of s^1 that is $s_j^1 \in \{0, 1\}$, and $|\phi''\rangle = |0\rangle$. The condition $|\phi''\rangle = |0\rangle$ means $j \leq 3b$ and the automaton reads on of symbols of $\mathcal{S}^{i,1} \circ \mathcal{U}^{i,1} \circ \mathcal{Z}^{i,1}$.

If $s_j^1 = 0$, then the automaton does nothing. If $s_j^1 = 1$, then the automaton applies the G^j transformation that is the rotation of $|\psi_{targ}\rangle$ qubit to the angle $\alpha_{a,j}$ with respect to the state $|a\rangle$ of $|\psi\rangle$ and $|j\rangle$ of $|\psi_{ind}\rangle$. Here $\alpha_{a,j} = \frac{2\pi k_a 2^j}{2^d}$.

Assume that the automaton reads the j -th symbol of s^1 that is $s_j^1 \in \{0, 1\}$, and $|\phi''\rangle \neq |0\rangle$ and $|\phi''\rangle \neq |M^i - 1\rangle$. The condition $|\phi''\rangle \neq |0\rangle$ and $|\phi''\rangle \neq |M^i - 1\rangle$ means $|s^1| \geq j > 3b$. The same symbol is the $(j - 3b)$ -th symbol of s^2 that is $s_{j-3b}^2 \in \{0, 1\}$.

If $s_j^1 = 0$, then the automaton does nothing. If $s_j^1 = 1$, then the automaton applies the transformation $G^{j,b}$ that rotates $|\psi_{targ}\rangle$ to the angle $\alpha_{a,j} - \alpha_{a,j-3b}$ with respect to the state $|a\rangle$ of $|\psi\rangle$ and $|j\rangle$ of $|\psi_{ind}\rangle$.

Note, that on this step $|\lambda'\rangle = |b\rangle$ and $|\zeta''\rangle = |M^i\rangle$ are already computed.

Assume that the automaton reads the j -th symbol of s^1 that is $s_j^1 \in \{0, 1\}$, and $|\phi''\rangle = |M^i - 1\rangle$. The condition $|\phi''\rangle = |M^i - 1\rangle$ means the first string is finished and we read the $(j - 3b)$ -th symbol of s^2 $s_{j-3b}^2 \in \{0, 1\}$ and $j > |s^1|$.

If $s_{j-3b}^2 = 0$, then the automaton does nothing. If $s_{j-3b}^2 = 1$, then the automaton applies the transformation $G^{j,b}$ that rotates $|\psi_{targ}\rangle$ to the angle $-\alpha_{a,j-3b}$ with respect to the state $|a\rangle$ of $|\psi\rangle$ and $|j\rangle$ of $|\psi_{ind}\rangle$.

Finally, the automaton on symbol 6 applies the Hadamard transformation to $|\psi\rangle$ and measures $|\psi_{targ}\rangle$. If we obtain 0 as a result of measurement, then these strings are equal; otherwise, they are unequal. So, memory size is $O(\log d - \log \epsilon')$ qubits. Assume that $\epsilon' = \frac{1}{d}$, then the automaton computes the function with error probability $O(\frac{1}{d})$ and uses $O(\log d)$ qubits.

Appendix A.4. The Implementation of the Procedure for Checking the Forth Property from Lemma 2

Let us remind the reader of the property that we are checking.

- $\mathcal{S}^{i,1} = \mathcal{Z}^{i,1}$.

The fourth procedure is similar to the third procedure. We check an equality $\mathcal{S}^{i,1} = \mathcal{Z}^{i,1}$. We construct an automaton using Lemma 1 for checking the property with bounded error ϵ'' . Suppose $\epsilon'' = 1/d$, then the automaton reaches an error probability $O(\frac{1}{d})$ using $O(\log d)$ qubits.

The quantum memory is a quantum register $|\psi'\rangle$ of $q = \lceil \log_2(2d/\epsilon') \rceil$ qubits, and a qubit $|\psi'_{targ}\rangle$. We have the same set of special parameters $S = (k_1, \dots, k_{2^q})$ as for the third procedure. We start from the $|0\rangle|0\rangle$ state.

Firstly, using the Hadamard transformation for $|\psi'\rangle$, we obtain

$$|0\rangle|0\rangle \rightarrow \frac{1}{\sqrt{2^q}} \sum_{a=0}^{2^q} |a\rangle|0\rangle.$$

Let the automaton read the j -th symbol of s^1 that is $s_j^1 \in \{0, 1\}$, for $j \leq b$. Therefore, the automaton reads the j -th symbol of $\mathcal{S}^{i,1}$.

If $s_j^1 = 0$, then the automaton does nothing. If $s_j^1 = 1$, then the automaton applies a transformation that rotates $|\psi'_{targ}\rangle$ to an angle $\alpha_{a,j}$ with respect to the state $|a\rangle$ of the register $|\psi'\rangle$ and $|j\rangle$ of $|\psi_{ind}\rangle$.

Let the automaton read the j -th symbol of s^1 that is $s_j^1 \in \{0, 1\}$, for $2b < j \leq 3b$. It means the automaton reads the j -th symbol of $\mathcal{Z}^{i,1}$.

If $s_j^1 = 0$, then the automaton does nothing. If $s_j^1 = 1$, then the automaton applies the transformation that rotates $|\psi'_{targ}\rangle$ to an angle $-\alpha_{a,j-2b}$ with respect to the state $|a\rangle$ of the register $|\psi'\rangle$ and $|j\rangle$ of $|\psi_{ind}\rangle$.

If the input symbol is 4, then the automaton applies the Hadamard transformation to $|\psi'\rangle$ and measures $|\psi'_{targ}\rangle$. If the result of the measurement is 0, then $S^{i,1} = Z^{i,1}$. Otherwise, the input does not satisfy the fourth condition. Suppose $\epsilon'' = \frac{1}{d}$, then the automaton has an error probability $O(\frac{1}{d})$ and uses quantum memory of size $O(\log d)$.

References

1. Boyar, J.; Irani, S.; Larsen, K.S. A comparison of performance measures for online algorithms. In *Workshop on Algorithms and Data Structures*; Springer: Berlin/Heidelberg, Germany, 2009; Volume 5664, pp. 119–130.
2. Dorrigiv, R.; López-Ortiz, A. A survey of performance measures for on-line algorithms. *SIGACT News* **2005**, *36*, 67–81.
3. Sleator, D.D.; Tarjan, R.E. Amortized efficiency of list update and paging rules. *Commun. ACM* **1985**, *28*, 202–208. [[CrossRef](#)]
4. Karlin, A.R.; Manasse, M.S.; Rudolph, L.; Sleator, D.D. Competitive snoopy caching. In *Proceedings of the 27th Annual Symposium on Foundations of Computer Science*, Toronto, ON, Canada, 27–29 October 1986; pp. 244–254.
5. Becchetti, L.; Koutsoupias, E. Competitive Analysis of Aggregate Max in Windowed Streaming. In *International Colloquium on Automata, Languages, and Programming*; Springer: Berlin/Heidelberg, Germany, 2009; Volume 5555, pp. 156–170.
6. Giannakopoulos, Y.; Koutsoupias, E. Competitive analysis of maintaining frequent items of a stream. *Theor. Comput. Sci.* **2015**, *562*, 23–32. [[CrossRef](#)]
7. Boyar, J.; Larsen, K.S.; Maiti, A. The frequent items problem in online streaming under various performance measures. *Int. J. Found. Comput. Sci.* **2015**, *26*, 413–439. [[CrossRef](#)]
8. Khadiev, K.; Khadieva, A.; Mannapov, I. Quantum Online Algorithms with Respect to Space and Advice Complexity. *Lobachevskii J. Math.* **2018**, *39*, 1210–1220. [[CrossRef](#)]
9. Ablayev, F.; Ablayev, M.; Khadiev, K.; Vasiliev, A. Classical and quantum computations with restricted memory. In *Adventures Between Lower Bounds and Higher Altitudes*; Springer: Cham, Switzerland, 2018; Volume 11011, pp. 129–155.
10. Baliga, G.R.; Shende, A.M. On space bounded server algorithms. In *Proceedings of the ICCI'93: 5th International Conference on Computing and Information*, Sudbury, ON, Canada, 27–29 May 1993; pp. 77–81.
11. Hughes, S. A new bound for space bounded server algorithms. In *Proceedings of the 33rd Annual on Southeast Regional Conference*, Clemson, SC, USA, 17–18 March 1995; pp. 165–169.
12. Flammini, M.; Navarra, A.; Nicosia, G. Efficient offline algorithms for the bicriteria k-server problem and online applications. *J. Discret. Algorithms* **2006**, *4*, 414–432. [[CrossRef](#)]
13. Rudec, T.; Baumgartner, A.; Manger, R. A fast work function algorithm for solving the k-server problem. *Cent. Eur. J. Oper. Res.* **2013**, *21*, 187–205. [[CrossRef](#)]
14. Kapralov, R.; Khadiev, K.; Mokut, J.; Shen, Y.; Yagafarov, M. Fast Classical and Quantum Algorithms for Online k-server Problem on Trees. *arXiv* **2020**, arXiv:2008.00270.
15. Nielsen, M.A.; Chuang, I.L. *Quantum Computation and Quantum Information*; Cambridge University Press: Cambridge, UK, 2010.
16. Ambainis, A. Understanding Quantum Algorithms via Query Complexity. *Proc. Int. Conf. Math.* **2018**, *4*, 3283–3304.
17. Ablayev, F.; Ablayev, M.; Huang, J.Z.; Khadiev, K.; Salikhova, N.; Wu, D. On quantum methods for machine learning problems part I: Quantum tools. *Big Data Min. Anal.* **2019**, *3*, 41–55. [[CrossRef](#)]
18. de Wolf, R. *Quantum Computing and Communication Complexity*; Institute for Logic, Language and Computation: Amsterdam, The Netherlands, 2001.
19. Jordan, S. Quantum Algorithms Zoo. 2021. Available online: <http://quantumalgorithmzoo.org/> (accessed on 25 December 2021).
20. Montanaro, A. Quantum algorithms: An overview. *NPJ Quantum Inf.* **2016**, *2*, 1–8. [[CrossRef](#)]
21. Khadiev, K.; Khadieva, A.; Kravchenko, D.; Rivosch, A.; Yamilov, R.; Mannapov, I. Quantum versus Classical Online Streaming Algorithms with Logarithmic Size of Memory. *arXiv* **2019**, arXiv:1710.09595.
22. Yuan, Q. Quantum Online Algorithms. Ph.D. Thesis, UC Santa Barbara, Santa Barbara, CA, USA, 2009.
23. Khadiev, K.; Khadieva, A. Two-Way Quantum and Classical Automata with Advice for Online Minimization Problems. In *International Symposium on Formal Methods*; Springer, Cham, Switzerland, 2020; pp. 428–442.
24. Khadiev, K.; Khadieva, A. Two-way quantum and classical machines with small memory for online minimization problems. In *International Conference on Micro-and Nano-Electronics 2018*; International Society for Optics and Photonics: Bellingham, WA, USA, 2019; Volume 11022, p. 110222T. [[CrossRef](#)]
25. Khadiev, K.; Lin, D. Quantum online algorithms for a model of the request-answer game with a buffer. *Uchenye Zap. Kazan. Univ. Seriya Fiz. Mat. Nauk.* **2020**, *162*, 367–382. [[CrossRef](#)]
26. Khadiev, K. Quantum request-answer game with buffer model for online algorithms. Application for the Most Frequent Keyword Problem. *CEUR Workshop Proc.* **2021**, *2850*, 16–27.
27. Le Gall, F. Exponential Separation of Quantum and Classical Online Space Complexity. In *Proceedings of the 18th ACM Symposium on Parallelism in Algorithms and Architectures*, Cambridge, MA, USA, 30 July–2 August 2006; pp. 67–73.

28. Le Gall, F. Exponential separation of quantum and classical online space complexity. *Theory Comput. Syst.* **2009**, *45*, 188–202. [[CrossRef](#)]
29. Gavinsky, D.; Kempe, J.; Kerenidis, I.; Raz, R.; de Wolf, R. Exponential Separations for One-way Quantum Communication Complexity, with Applications to Cryptography. In Proceedings of the 39th Annual ACM Symposium on Theory of Computing, San Diego, CA, USA, 11–13 June 2007; pp. 516–525.
30. Ablayev, F.; Gainutdinova, A.; Karpinski, M.; Moore, C.; Pollett, C. On the computational power of probabilistic and quantum branching program. *Inf. Comput.* **2005**, *203*, 145–162. [[CrossRef](#)]
31. Ablayev, F.; Gainutdinova, A.; Khadiev, K.; Yakaryılmaz, A. Very narrow quantum OBDDs and width hierarchies for classical OBDDs. *Lobachevskii J. Math.* **2016**, *37*, 670–682. [[CrossRef](#)]
32. Gainutdinova, A. Comparative complexity of quantum and classical OBDDs for total and partial functions. *Russ. Math.* **2015**, *59*, 26–35. [[CrossRef](#)]
33. Sauerhoff, M.; Sieling, D. Quantum branching programs and space-bounded nonuniform quantum complexity. *Theor. Comput. Sci.* **2005**, *334*, 177–225. [[CrossRef](#)]
34. Ambainis, A.; Yakaryılmaz, A. Superiority of exact quantum automata for promise problems. *Inf. Process. Lett.* **2012**, *112*, 289–291. [[CrossRef](#)]
35. Ambainis, A.; Yakaryılmaz, A. Automata and Quantum Computing. *arXiv* **2015**, arXiv:1507.01988
36. Khadiev, K.; Khadieva, A. Reordering Method and Hierarchies for Quantum and Classical Ordered Binary Decision Diagrams. In *International Computer Science Symposium in Russia*; Springer: Cham, Switzerland, 2017; Volume 10304, pp. 162–175.
37. Ablayev, F.; Ambainis, A.; Khadiev, K.; Khadieva, A. Lower Bounds and Hierarchies for Quantum Memoryless Communication Protocols and Quantum Ordered Binary Decision Diagrams with Repeated Test. In *International Conference on Current Trends in Theory and Practice of Informatics*; Edizioni della Normale: Cham, Switzerland, 2018; Volume 10706, pp. 197–211.
38. Gainutdinova, A.; Yakaryılmaz, A. Nondeterministic unitary OBDDs. In *International Computer Science Symposium in Russia*; Springer: Cham, Switzerland, 2017; pp. 126–140.
39. Gainutdinova, A.; Yakaryılmaz, A. Unary probabilistic and quantum automata on promise problems. *Quantum Inf. Process.* **2018**, *17*, 28. [[CrossRef](#)]
40. Ibrahimov, R.; Khadiev, K.; Průsis, K.; Yakaryılmaz, A. Error-free affine, unitary, and probabilistic OBDDs. *Int. J. Found. Comput. Sci.* **2021**, *32*, 827–847. [[CrossRef](#)]
41. Buhrman, H.; Cleve, R.; Wigderson, A. Quantum vs. classical communication and computation. In Proceedings of the the 30th Annual ACM Symposium on Theory of Computing, Dallas, TX, USA, 24–26 May 1998; pp. 63–68.
42. Razborov, A.A. On the distributional complexity of disjointness. In *International Colloquium on Automata, Languages, and Programming*; Springer: Berlin/Heidelberg, Germany, 1990; pp. 249–253.
43. Chattopadhyay, A.; Pitassi, T. The story of set disjointness. *ACM SIGACT News* **2010**, *41*, 59–85. [[CrossRef](#)]
44. Khadiev, K.; Khadieva, A. Quantum Online Streaming Algorithms with Logarithmic Memory. *Int. J. Theor. Phys.* **2021**, *60*, 608–616. [[CrossRef](#)]
45. Komm, D. *An Introduction to Online Computation: Determinism, Randomization, Advice*; Springer: Cham, Switzerland, 2016.
46. Boyar, J.; Favrholt, L.; Kudahl, C.; Larsen, K.; Mikkelsen, J. Online Algorithms with Advice: A Survey. *ACM Comput. Surv.* **2017**, *50*, 19.
47. Böckenhauer, H.J.; Komm, D.; Kráľovič, R.; Kráľovič, R.; Mömke, T. On the advice complexity of online problems. In *International Symposium on Algorithms and Computation*; Springer: Berlin/Heidelberg, Germany, 2009; Volume 5878, pp. 331–340.
48. Hromkovic, J. *Design and Analysis of Randomized Algorithms: Introduction to Design Paradigms*; Springer: Berlin/Heidelberg, Germany, 2005.
49. Böckenhauer, H.J.; Hromkovič, J.; Komm, D.; Kráľovič, R.; Rossmann, P. On the power of randomness versus advice in online computation. In *Languages Alive*; Springer: Berlin/Heidelberg, Germany, 2012; pp. 30–43.
50. Böckenhauer, H.J.; Hromkovič, J.; Komm, D.; Krug, S.; Smula, J.; Sprock, A. The string guessing problem as a method to prove lower bounds on the advice complexity. *Theor. Comput. Sci.* **2014**, *554*, 95–108. [[CrossRef](#)]
51. Wegener, I. *Branching Programs and Binary Decision Diagrams: Theory and Applications*; SIAM: Philadelphia, PA, USA, 2000.
52. Ablayev, F.; Gainutdinova, A.; Karpinski, M. On Computational Power of Quantum Branching Programs. In *International Symposium on Fundamentals of Computation Theory*; Springer: Berlin/Heidelberg, Germany, 2001; Volume 2138, pp. 59–70.
53. Ablayev, F.M.; Gainutdinova, A. On the Lower Bounds for One-Way Quantum Automata. In *Mathematical Foundations of Computer Science 2000*; Springer: Berlin/Heidelberg, Germany, 2000; Volume 1893, pp. 132–140.
54. Ablayev, F.; Gainutdinova, A. Complexity of Quantum Uniform and Nonuniform Automata. In *Developments in Language Theory*; Springer: Berlin/Heidelberg, Germany, 2005; Volume 3572, pp. 78–87.
55. Say, A.C.; Yakaryılmaz, A. Quantum finite automata: A modern introduction. In *Computing with New Resources*; Springer: Berlin/Heidelberg, Germany, 2014; pp. 208–222.
56. Ablayev, F.; Vasiliev, A. On quantum realisation of Boolean functions by the fingerprinting technique. *Discret. Math. Appl.* **2009**, *19*, 555–572. [[CrossRef](#)]
57. Ablayev, F.M.; Vasiliev, A. Algorithms for Quantum Branching Programs Based on Fingerprinting. *Int. J. Softw. Inform.* **2013**, *7*, 485–500. [[CrossRef](#)]

58. Grover, L.K. A fast quantum mechanical algorithm for database search. In Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing, Philadelphia, PA, USA, 22–24 May 1996; pp. 212–219.
59. Boyer, M.; Brassard, G.; Høyer, P.; Tapp, A. Tight bounds on quantum searching. *Fortschritte Phys.* **1998**, *46*, 493–505. [[CrossRef](#)]
60. Kushilevitz, E.; Nisan, N. *Communication Complexity*; Cambridge University Press: Cambridge, UK, 1997; pp. I–XIII, 1–189.
61. Khadiev, K. On the Hierarchies For Deterministic, Nondeterministic and Probabilistic Ordered Read-k-Times Branching Programs. *Lobachevskii J. Math.* **2016**, *37*, 682–703. [[CrossRef](#)]
62. Khadiev, K.; Ibrahimov, R.; Yakaryilmaz, A. New Size Hierarchies for Two Way Automata. *Lobachevskii J. Math.* **2018**, *39*, 997–1009.
63. Dwork, C.; Stockmeyer, L.J. A time complexity gap for two-way probabilistic finite-state automata. *SIAM J. Comput.* **1990**, *19*, 1011–1123. [[CrossRef](#)]
64. Ablayev, F.; Khadiev, K. Extension of the hierarchy for k-OBDDs of small width. *Russ. Math.* **2013**, *53*, 46–50. [[CrossRef](#)]
65. Shepherdson, J.C. The reduction of two-way automata to one-way automata. *IBM J. Res. Dev.* **1959**, *3*, 198–200. [[CrossRef](#)]