TUCS

Mika Hirvensalo
Abuzer Yakaryılmaz (Eds.)

Proceedings of Workshop on Quantum Computing and Quantum Information

Turku Centre *for* Computer Science

TUCS Lecture Notes
No 30, June 2019

Mika Hirvensalo
Abuzer Yakaryılmaz
(Editors)

Proceedings of

Workshop on Quantum Computing and Quantum Information

June 3, 2019, Tokyo, Japan

ISBN 978-952-12-3840-6
ISSN 1797-8831

Preface

The Workshop on Quantum Computing and Quantum Information was held on June 3, 2019 at Tokyo, Japan, in conjunction with the 18$^{\text{th}}$ International Conference on Unconventional Computation and Natural Computation (UCNC 2019). The Workshop was initiated by UCNC program committee members Shinnosuke Seki and Jarkko Kari, who suggested to the organizers that such a workshop could be beneficial. In the end of 2018, the program committee was assembled, and the call for papers was published subsequently.

The topic area of the workshop was defined very broadly, and from the very beginning it was clear that the workshop should distinguish from the main event so that presentations including non-finished ideas should be also welcomed, and this was explicitly mentioned in the letter calling for papers. All submitted papers were peer-reviewed and the program committee accepted four contributions to be presented in the workshop. Unfortunately one manuscript was withdrawn by the authors, due to an unsuspected personal restriction for presenting the manuscript in the workshop.

In addition to the contributing papers, the workshop included one plenary talk. Mika Hirvensalo gave a lecture on the role of interference in computing.

The organizers gratefully acknowledge the support by the Academy of Finland, University of Electro-Communications, Tokyo, and University of Turku, Finland.

June 2019

Mika Hirvensalo
Abuzer Yakaryılmaz
Program Chairs

Workshop on Quantum Computing
and Quantum Information

Workshop on Quantum Computing and Quantum Information

Organizers

Mika Hirvensalo	(University of Turku)
Abuzer Yakaryılmaz	(University of Latvia)

Program Commitee

Alexandrs Belovs	(University of Latvia, Riga, Latvia)
Aida Gainutdinova	(University of Kazan, Tatarstan, Russia)
Mika Hirvensalo	(University of Turku, Finland) (co-chair)
Kamil Khadiev	(University of Kazan, Tatarstan, Russia)
François Le Gall	(Kyoto University, Japan)
Franklin Marquezino	(Federal University of Rio de Janeiro, Brazil)
Masaki Nakanishi	(Yamagata University, Japan)
Krišjānis Prūsis	(University of Latvia, Riga, Latvia)
Māris Ozols	(University of Amsterdam, The Netherlands)
Abuzer Yakaryılmaz	(University of Latvia, Riga, Latvia) (co-chair)

Sponsoring Institutions

Academy of Finland
University of Electro-Communications, Tokyo
University of Turku, Finland

Table of Contents

Quantum Algorithms for the Most Frequently String Search, Intersection of Two String Sequences and Sorting of Strings Problems

Kamil Khadiev[1,2] and Artem Ilikaev[2]

[1] Smart Quantum Technologies Ltd., Kazan, Russia
[2] Kazan Federal University, Kazan, Russia
`kamil.hadiev@kpfu.ru, artemka.tema1998@gmail.com`

Abstract. We study algorithms for solving three problems on strings. The first one is the Most Frequently String Search Problem. The problem is the following. Assume that we have a sequence of n strings of length k. The problem is finding the string that occurs in the sequence most often. We propose a quantum algorithm that has a query complexity $\tilde{O}(n\sqrt{k})$. This algorithm shows speed-up comparing with the deterministic algorithm that requires $\Omega(nk)$ queries.
The second one is searching intersection of two sequences of strings. All strings have the same length k. The size of the first set is n and the size of the second set is m. We propose a quantum algorithm that has a query complexity $\tilde{O}((n+m)\sqrt{k})$. This algorithm shows speed-up comparing with the deterministic algorithm that requires $\Omega((n+m)k)$ queries.
The third problem is sorting of n strings of length k. On the one hand, it is known that quantum algorithms cannot sort objects asymptotically faster than classical ones. On the other hand, we focus on sorting strings that are not arbitrary objects. We propose a quantum algorithm that has a query complexity $O(n(\log n)^2\sqrt{k})$. This algorithm shows speed-up comparing with the deterministic algorithm (radix sort) that requires $\Omega((n+d)k)$ queries, where d is a size of the alphabet.
Keywords: quantum computation, quantum models, quantum algorithm, query model, string search, sorting

1 Introduction

Quantum computing [26, 5] is one of the hot topics in computer science of last decades. There are many problems where quantum algorithms outperform the best known classical algorithms [12, 17, 19, 18].

One of these problems are problems for strings. Researchers show the power of quantum algorithms for such problems in [25, 6, 29].

In this paper, we consider three problems:

⋆ the Most Frequently String Search problem;
⋆ Strings sorting problem;
⋆ Intersection of Two String Sequences problem.

Our algorithms use some quantum algorithms as a subroutine, and the rest part is classical. We investigate the problems in terms of query complexity. The query model is one of the most popular in the case of quantum algorithms. Such algorithms can do a query to a black box that has access to the sequence of strings. As a running time of an algorithm, we mean a number of queries to the black box.

The first problem is the following. We have n strings of length k. We can assume that symbols of strings are letters from any finite alphabet, for example, binary, Latin alphabet or Unicode. The problem is finding the string that occurs in the sequence most often. The best known deterministic algorithms require $\Omega(nk)$ queries because an algorithm should at least test all symbols of all strings. The deterministic solution can use the Trie (prefix tree) [11, 7, 9, 21] that allows to achieve the required complexity.

We propose a quantum algorithm that uses a self-balancing binary search tree for storing strings and a quantum algorithm for comparing strings. As a self-balancing binary search tree we can use the AVL tree [2, 10] or the Red-Black tree [14, 10]. As a string comparing algorithm, we propose an algorithm that is based on the first one search problem algorithm from [22–24]. This algorithm is a modification of Grover's search algorithm [13, 8]. Our algorithm for the most frequently string search problem has query complexity $O(n(\log n)^2 \cdot \sqrt{k}) = \tilde{O}(n\sqrt{k})$, where $\tilde{O}$ does not consider a log factors. If $\log_2 n = o(k^{0.25})$, then our algorithm is better than deterministic one. Note, that this setup makes sense in practical cases.

The second problem is String sorting. Assume that we have n strings of length k. It is known [15, 16] that no quantum algorithm can sort arbitrary comparable objects faster than $O(n \log n)$. At the same time, several researchers tried to improve the hidden constant [28, 27]. Other researchers investigated space bounded case [20]. We focus on sorting strings. In a classical case, we can use an algorithm that is better than arbitrary comparable objects sorting algorithms. It is radix sort that has $O((n+d)k)$ query complexity [10], where d is a size of the alphabet. Our quantum algorithm for the string sorting problem has query complexity $O(n(\log n)^2 \cdot \sqrt{k}) = \tilde{O}(n\sqrt{k})$. It is based on standard sorting algorithms like Merge sort [10] or Heapsort [30, 10] and the quantum algorithm for comparing strings.

The third problem is the Intersection of Two String Sequences problem. Assume that we have two sequences of strings of length k. The size of the first set is n and the size of the second one is m. The first sequence is given and the second one is given in online fashion, one by one. After each requested string from the second sequence, we want to check weather this string belongs to the first sequence. We propose two quantum algorithms for the problem. Both algorithms has query complexity $O((n + m) \cdot \log n \cdot \log(n + m)\sqrt{k}) = \tilde{O}(n\sqrt{k})$. The first algorithm uses a self-balancing binary search tree like the solution of the first problem. The second algorithm uses a quantum algorithm for sorting strings and has better big-O hidden constant. At the same time, the best known deterministic algorithm requires $O((n + m)k)$ queries.

The structure of the paper is the following. We present the quantum subroutine that compares two strings in Section 2. Then we discussed three problems: the Most Frequently String Search problem in Section 3, Strings Sorting problem in Section 4 and Intersection of Two String Sequences problem in Section 5.

2 The Quantum Algorithm for Two Strings Comparing

Firstly, we discuss a quantum subroutine that compares two strings of length k. Assume that this subroutine is COMPARE_STRINGS(s, t, k) 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. *[22–24] Suppose, we have a function $f : \{1, \dots, N\} \to \{0, 1\}$ 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 expected query complexity $\sqrt{j_0}$ 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. Then, we can claim that s precedes t in lexicographical order iff s_{j_0} precedes t_{j_0} in alphabet Σ. If there are no unequal symbols, then the strings are equal.

We use the standard technique of boosting success probability. So, we repeat the algorithm $3 \log_2 n$ times and return the minimal answer, where n is a number of strings in the sequence s. In that case, the error probability is $O\left(\frac{1}{2^{3 \log n}}\right) = \left(\frac{1}{n^3}\right)$.

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

Let us discuss the property of the algorithm:

Lemma 2. *Algorithm 1 compares two strings of length k in lexicographical order with query complexity $O(\sqrt{k} \log n)$ and error probability $O\left(\frac{1}{n^3}\right)$.*

3 The Most Frequently String Search Problem

Let us formally present the problem.

Problem. For some positive integers n and k, we have the sequence of strings $s = (s^1, \dots, s^n)$. Each $s^i = (s^i_1, \dots, s^i_k) \in \Sigma^k$ for some finite size alphabet Σ. Let $\#(s) = |\{i \in \{1, \dots, m\} : s^i = s\}|$ be a number of occurrences of string s. We search $s = argmax_{s^i \in S} \#(s^i)$.

Algorithm 1 COMPARE_STRINGS(s, t, k). The Quantum Algorithm for Two Strings Comparing.

```
$j_0 \leftarrow$ THE_FIRST_ONE_SEARCH$(f, k)$                    ▷ The initial value
for $i \in \{1, \dots, 3\log_2 n\}$ do
    $j_0 \leftarrow \min(j_0,$ THE_FIRST_ONE_SEARCH$(f, k))$
end for
if $j_0 = k + 1$ then
    $result \leftarrow 0$                                       ▷ The strings are equal.
end if
if $(j_0 \neq k + 1)\&(s_{j_0} < t_{j_0})$ then
    $result \leftarrow -1$                                      ▷ $s$ precedes $t$.
end if
if $(j_0 \neq k + 1)\&(s_{j_0} > t_{j_0})$ then
    $result \leftarrow 1$                                       ▷ $s$ succeeds $t$.
end if
return $result$
```

3.1 The Quantum algorithm

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 [2, 10] or the Red-Black tree [14, 10]. Both data structures allow as 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 pairs (i, c) in vertexes of the tree, where i is an index of a string from s and c is a number of occurrences of the string s^i. We assume that a pair (i, c) is less than a pair (i', c') iff s^i precedes $s^{i'}$ in the lexicographical order. So, we use COMPARE_STRINGS$(s^i, s^{i'}, k)$ subroutine as the compactor of the vertexes. The tree represents a set of unique strings from $(s^1, \dots, s^n)$ with a number of occurrences.

We consider all strings from s^1 to s^n and check the existence of a string in our tree. If a string exists, then we increase the number of occurrences. If the string does not exist in the tree, then we add it. At the same time, we store $(i_{max}, c_{max}) = argmax_{(i,c) \text{ in the tree}} c$ and recalculate it in each step.

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

- ⋆ FIND(BST, s^i) finds vertex (i, c) or returns $NULL$ if such vertex does not exist;
- ⋆ ADD(BST, s^i) adds vertex $(i, 0)$ to the tree and returns the vertex as a result;
- ⋆ INIT(BST) initializes an empty tree;

Let us discuss the property of the algorithm.

Theorem 1. *Algorithm 2 finds the most frequently string from $s = (s^1, \dots, s^n)$ with query complexity $O(n(\log n)^2 \cdot \sqrt{k})$ and error probability $O\left(\frac{1}{n}\right)$.*

Algorithm 2 The Quantum Algorithm for Most Frequently String 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 requently string.
for $i \in \{1, \dots, n\}$ do
    $v = (i, c) \leftarrow$ Find($BST, s^i$)          ▷ Searching $s^i$ in the tree.
    if $v = NULL$ then
        $v = (i, c) \leftarrow$ Add($BST, s^i$)       ▷ If there is no $s^i$, then we add it.
    end if
    $c \leftarrow c + 1$     ▷ Updating the vertex by increasing the number of occurrences.
    if $c > c_{max}$ then                    ▷ Updating the maximal value.
        $c_{max} \leftarrow c$
        $i_{max} \leftarrow i$
    end if
end for
return $s^{i_{max}}$
```

Proof. The correctness of the algorithm follows from the description. Let us discuss the query complexity. Each operation Find(BST, s^i) and Add(BST, s^i) requires $O(\log n)$ comparing operations Compare_strings($s^i, s^{i'}, k$). These operations are invoked n times. Therefore we have $O(n \log n)$ comparing operations. Due to Lemma 2, each comparing operation requires $O(\sqrt{k} \log n)$ queries. The total query complexity is $O(n\sqrt{k}(\log n)^2)$.

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 $1 - \left(1 - \frac{1}{n^3}\right)$. Hence, the probability of correctness of all $O(n \log n)$ events is at least $1 - \left(1 - \frac{1}{n^3}\right)^{\alpha \cdot n \log n}$ for some constant α.

Note that

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

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

□

The data structure that we used can be considered as a separated data structure. We call it *"Multi-set of strings with quantum comparator"*. Using this data structure, we can implement

- ⋆ *"Set of strings with quantum comparator"* if always $c = 1$ in pair (i, c) of a vertex;
- ⋆ *"Map with string key and quantum comparator"* if we replace c by any data $r \in \Gamma$ for any set Γ. In that case the data structure implements mapping $\Sigma^k \to \Gamma$.

All of these data structures has $O((\log n)^2 \sqrt{k})$ complexity of basic operations (Find, Add, Delete).

3.2 On the Classical Complexity of the Problem

The best known classical algorithm stores string to Trie (prefix tree) [11, 7], [9, 21] and do the similar operations. The running time of such algorithm is $O(nk)$. At the same time, we can show that if the algorithm tested $o(nk)$ variables, then it can return a wrong answer.

Theorem 2. *Any deterministic algorithm for the Most Frequently String Search problem has $\Omega(nk)$ query complexity.*

Proof. Suppose, we have a deterministic algorithm A for the Most Frequently String Search problem that uses $o(nk)$ queries.

Let us consider an adversary that suggest an input. The adversary wants to construct an input such that the algorithm A obtains a wrong answer.

Without loss of generality, we can say that n is even. Suppose, a and b are different symbols from an input alphabet. If the algorithm requests an variable s^i_j for $i \leq n/2$, then the adversary returns a. If the algorithm requests an variable s^i_j for $i > n/2$, then the adversary returns b.

Because of the algorithm A uses $o(nk)$ queries, there are at least one $s^{z'}_{j'}$ and one $s^{z''}_{j''}$ that are not requested, where $z' \leq n/2$, $z'' > n/2$ and $j', j'' \in \{1, \dots, k\}$.

Let s' be a string such that $s'_j = a$ for all $j \in \{1, \dots, k\}$. Let s'' be a string such that $s''_j = b$ for all $j \in \{1, \dots, k\}$.

Assume that A returns s'. Then, the adversary assigns $s^{z'}_{j'} = b$ and assigns $s^i_j = b$ for each $i > n/2, j \in \{1, \dots, k\}$. Therefore, the right answer should be s''.

Assume that A returns a string $s \neq s'$. Then, the adversary assigns $s^{z''}_{j''} = a$ and assigns $s^i_j = a$ for each $i \leq n/2, j \in \{1, \dots, k\}$. Therefore, the right answer should be s'.

So, the adversary can construct the input such that A obtains a wrong answer. □

4 Strings Sorting Problem

Let us consider the following problem.

Problem. For some positive integers n and k, we have the sequence of strings $s = (s^1, \dots, s^n)$. Each $s^i = (s^i_1, \dots, s^i_k) \in \Sigma^k$ for some finite size alphabet Σ. We search order $ORDER = (i_1, \dots, i_n)$ such that for any $j \in \{1, \dots, n-1\}$ we have $s^{i_j} \leq s^{i_{j+1}}$ in lexicographical order.

We use Heap sort algorithm [30, 10] as a base and Quantum algorithm for comparing string from Section 2. We can replace Heap sort algorithm by any other sorting algorithm, for example, Merge sort [10]. In a case of Merge sort, the big-O hidden constant in query complexity will be smaller. At the same time, we need more additional memory.

Let us present Heap sort for completeness of the explanation. We can use Binary Heap [30]. We store indexes of strings in vertexes. As in the previous section, if we compare vertexes v and v' with corresponding indexes i and i', then $v > v'$ iff $s^i > s^{i'}$ in lexicographical order. We use $\textsc{Compare\_strings}(s^i, s^{i'}, k)$ for comparing strings. Binary Heap BH has three operations:

- ⋆ $\textsc{Get\_min\_and\_delete}(BH)$ returns minimal s^i and removes it from the data structure.
- ⋆ $\textsc{Add}(BH, s^i)$ adds vertex with value i to the heap;
- ⋆ $\textsc{Init}(BH)$ initializes an empty heap;

The operations Get_min_and_delete and Add invoke Compare_strings subroutine $\log_2 t$ times, where t is the size of the heap.

The algorithm is the following.

Algorithm 3 The Quantum Algorithm for Sorting Problem.

```
$\textsc{Init}(BH)$                                          ▷ The initialization of the heap.
for $i \in \{1, \dots, n\}$ do
    $\textsc{Add}(BH, s^i)$                                    ▷ Adding $s^i$ to the heap.
end for
for $i \in \{1, \dots, n\}$ do
    $ORDER \leftarrow ORDER \cup \textsc{Get\_min\_and\_delete}(BH)$   ▷ Getting minimal string.
end for
return $ORDER$
```

If we implement the sequence s as an array, then we can store the heap in the same array. In this case, we do not need additional memory.

We have the following property of the algorithm that can be proven by the same way as Theorem 1.

Theorem 3. *Algorithm 4 sorts $s = (s^1, \dots, s^n)$ with query complexity $O(n(\log n)^2 \cdot \sqrt{k})$ and error probability $O\left(\frac{1}{n}\right)$.*

The lower bound for deterministic complexity can be proven by the same way as in Theorem 2.

Theorem 4. *Any deterministic algorithm for Sorting problem has $\Omega(nk)$ query complexity.*

The Radix sort [10] algorithm almost reaches this bound and has $O((n + |\Sigma|)k)$ complexity.

5 Intersection of Two Sequences of Strings Problem

Let us consider the following problem.

Problem. For some positive integers n, m and k, we have the sequence of strings $s = (s^1, \dots, s^n)$. Each $s^i = (s^i_1, \dots, s^i_k) \in \Sigma^k$ for some finite size alphabet Σ. Then, we get m requests $t = (t^1 \dots t^m)$, where $t^i = (t^i_1, \dots, t^i_k) \in \Sigma^k$. The answer to a request t^i is 1 iff there is $j \in \{1, \dots, n\}$ such that $t^i = s^j$. We should answer 0 or 1 to each of m requests.

We have two algorithms. The first one is based on *"Set of strings with quantum comparator"* data structure from Section 3. We store all strings from s to a self-balancing binary search tree BST. Then, we answer each request using FIND(BST, s^i) operation. Let us present the Algorithm 4.

Algorithm 4 The Quantum Algorithm for Intersection of Two Sequences of Strings Problem using *"Set of strings with quantum comparator"* .

```
INIT($BST$)                                  ▷ The initialization of the tree.
for $i \in \{1, \dots, n\}$ do
    ADD($BST, s^i$)                          ▷ We add $s^i$ to the set.
end for
for $i \in \{1, \dots, m\}$ do
    $v \leftarrow$ FIND($BST, t^i$)          ▷ We search $t^i$ in the set.
    if $v = NULL$ then
        return 0
    end if
    if $v \neq NULL$ then
        return 1
    end if
end for
```

The second algorithm is based on Sorting algorithm from Section 4. We sort strings from s. Then, we answer to each request using binary search in the sorted sequence of strings [10] and COMPARE_STRINGS subroutine for comparing strings during the binary search. Let us present the Algorithm 5. Assume that the sorting Algorithm 4 is the subroutine SORT_STRINGS(s) and it returns the order $ORDER = (i_1, \dots, i_n)$. The binary search algorithm with COMPARE_STRINGS subroutine as comparator is subroutine BINARY_SEARCH_FOR_STRINGS$(t, s, OREDER)$ and it searches t in the ordered sequence $(s^{i_1}, \dots, s^{i_n})$. Suppose that the subroutine BINARY_SEARCH_FOR_STRINGS returns 1 if it finds t and 0 otherwise.

Algorithm 5 The Quantum Algorithm for Intersection of Two Sequences of Strings Problem using sorting algorithm .

```
$ORDER \leftarrow$ SORT_STRINGS($s$)         ▷ We sort $s = (s^1, \dots, s^n)$.
for $i \in \{1, \dots, m\}$ do
    $ans \leftarrow$ BINARY_SEARCH_FOR_STRINGS($t, s, OREDER$)    ▷ We search $t^i$ in the
ordered sequence.
    return $ans$
end for
```

The algorithms have the following query complexity.

Theorem 5. *Algorithm 4 and Algorithm 5 solve Intersection of Two Sequences of Strings Problem with query complexity $O((n+m)\sqrt{k}\cdot\log n\cdot\log(n+m))$ and error probability $O\left(\frac{1}{n+m}\right)$.*

Proof. The correctness of the algorithms follows from the description. Let us discuss the query complexity of the first algorithm. As in the proof of Theorem 1, we can show that constructing of the search tree requires $O(n\log n)$ comparing operations. Then, the searching of all strings t^i requires $O(m\log n)$ comparing operations. The total number of comparing operations is $O((m+n)\log n)$. We will use little bit modified version of the Algorithm 1 where we run it $3(\log(n+m))$ times. We can prove that comparing operation requires $O(\sqrt{k}\log(n+m))$ queries. The proof is similar to the proof of corresponding claim from the proof of Lemma 2. So, the total complexity is $O((n+m)\sqrt{k}\cdot\log n\cdot\log(n+m))$.

The second algorithm also has the same complexity because it uses $O(n\log n)$ comparing operations for sorting and $O(m\log n)$ comparing operations for all invocations of the binary search algorithm.

Let us discuss the error probability. Events of error in the algorithm are independent. So, all events should be correct. We can prove that the error probability for comparing operation is $O(1/(n+m)^3)$. The proof is like the proof of Lemma 2. So, the probability of correctness of one event is $1-\left(1-\frac{1}{(n+m)^3}\right)$. Hence, the probability of correctness of all $O((n+m)\log n)$ events is at least $1-\left(1-\frac{1}{(n+m)^3}\right)^{\alpha\cdot(n+m)\log n}$ for some constant α.

Note that

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

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

□

Note that Algorithm 5 has a better big-O hidden constant than Algorithm 4, because the Red-Black tree or AVL tree has a height that greats $\log_2 n$ constant times. So, adding elements to the tree and checking existence has bigger big-O hidden constant than sorting and binary search algorithms.

The lower bound for deterministic complexity can be proven by the same way as in Theorem 2.

Theorem 6. *Any deterministic algorithm for Intersection of Two Sequences of Strings Problem has $\Omega((n+m)k)$ query complexity.*

This complexity can be reached if we implement the set of strings s using Trie (prefix tree) [11, 7, 9, 21].

Note, that we can use the quantum algorithm for element distinctness [4],[3] for this problem. The algorithm solves a problem of finding two identical elements in the sequence. The query complexity of the algorithm is $O(D^{2/3})$, where D is a number of elements in the sequence. The complexity is tight because of [1]. The algorithm can be the following. On j-th request, we can add the string t^j to the sequence $s^1, \dots, s^n$ and invoke the element distinctness algorithm that finds a collision of t^j with other strings. Such approach requires $\Omega(n^{2/3})$ query for each request and $\Omega(mn^{2/3})$ for processing all requests. Note, that the streaming nature of requests does not allow us to access to all $t^1, \dots, t^m$ by Oracle. So, each request should be processed separately.

6 Conclusion

In the paper we propose a quantum algorithm for comparing strings. Using this algorithm we discussed four data structures: *"Multi-set of strings with quantum comparator"*, *"Set of strings with quantum comparator"*, *"Map with a string key and quantum comparator"* and *"Binary Heap of strings with quantum comparator"*. We show that the first two data structures work faster than the implementation of similar data structures using Trie (prefix tree) in a case of $\log_2 n = o(k^{0.25})$. The trie implementation is the best known classical implementation in terms of complexity of simple operations (add, delete or find). Additionally, we constructed a quantum strings sort algorithm that works faster than the radix sort algorithm that is the best known deterministic algorithm for sorting a sequence of strings.

Using these two groups of results, we propose quantum algorithms for two problems: the Most Frequently String Search and Intersection of Two String Sets. These quantum algorithms are more efficient than deterministic ones.

Acknowledgement

We thank Aliya Khadieva and Farid Ablayev for useful discussions.

The work is performed according to the Russian Government Program of Competitive Growth of Kazan Federal University.

References

1. Aaronson, S., Shi, Y.: Quantum lower bounds for the collision and the element distinctness problems. Journal of the ACM (JACM) 51(4), 595–605 (2004)
2. Adel'son-Vel'skii, G.M., Landis, E.M.: An algorithm for organization of information. In: Doklady Akademii Nauk. vol. 146, pp. 263–266. Russian Academy of Sciences (1962)
3. Ambainis, A.: Quantum walk algorithm for element distinctness. In: Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science. pp. 22–31. FOCS '04 (2004)
4. Ambainis, A.: Quantum walk algorithm for element distinctness. SIAM Journal on Computing 37(1), 210–239 (2007)
5. Ambainis, A.: Understanding quantum algorithms via query complexity. arXiv preprint arXiv:1712.06349 (2017)

6. Bennett, C.H., Bernstein, E., Brassard, G., Vazirani, U.: Strengths and weaknesses of quantum computing. SIAM journal on Computing 26(5), 1510–1523 (1997)
7. Black, P.E.: Dictionary of algorithms and data structures— nist. Tech. rep. (1998)
8. Boyer, M., Brassard, G., Høyer, P., Tapp, A.: Tight bounds on quantum searching. Fortschritte der Physik 46(4-5), 493–505 (1998)
9. Brass, P.: Advanced data structures, vol. 193. Cambridge University Press Cambridge (2008)
10. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms-Secund Edition. McGraw-Hill (2001)
11. De La Briandais, R.: File searching using variable length keys. In: Papers presented at the the March 3-5, 1959, western joint computer conference. pp. 295–298. ACM (1959)
12. De Wolf, R.: Quantum computing and communication complexity (2001)
13. Grover, L.K.: A fast quantum mechanical algorithm for database search. In: Proceedings of the twenty-eighth annual ACM symposium on Theory of computing. pp. 212–219. ACM (1996)
14. Guibas, L.J., Sedgewick, R.: A dichromatic framework for balanced trees. In: 19th Annual Symposium on Foundations of Computer Science (sfcs 1978). pp. 8–21. IEEE (1978)
15. Høyer, P., Neerbek, J., Shi, Y.: Quantum complexities of ordered searching, sorting, and element distinctness. In: International Colloquium on Automata, Languages, and Programming. pp. 346–357. Springer (2001)
16. Høyer, P., Neerbek, J., Shi, Y.: Quantum complexities of ordered searching, sorting, and element distinctness. Algorithmica 34(4), 429–448 (2002)
17. Jordan, S.: Bounded error quantum algorithms zoo, https://math.nist.gov/quantum/zoo
18. Khadiev, K., Kravchenko, D., Serov, D.: On the quantum and classical complexity of solving subtraction games. In: Proceedings of CSR 2019. LNCS (2019)
19. Khadiev, K., Safina, L.: Quantum algorithm for dynamic programming approach for dags. applications for zhegalkin polynomial evaluation and some problems on dags. In: Proceedings of Unconventional Computation and Natural Computation 2019, LNCS, vol. 4362 (2019)
20. Klauck, H.: Quantum time-space tradeoffs for sorting. In: Proceedings of the thirty-fifth annual ACM symposium on Theory of computing. pp. 69–76. ACM (2003)
21. Knuth, D.: Searching and sorting, the art of computer programming, vol. 3 (1973)
22. Kothari, R.: An optimal quantum algorithm for the oracle identification problem. In: 31st International Symposium on Theoretical Aspects of Computer Science. p. 482 (2014)
23. Lin, C.Y.Y., Lin, H.H.: Upper bounds on quantum query complexity inspired by the elitzur-vaidman bomb tester. In: 30th Conference on Computational Complexity (CCC 2015). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik (2015)
24. Lin, C.Y.Y., Lin, H.H.: Upper bounds on quantum query complexity inspired by the elitzur–vaidman bomb tester. Theory OF Computing 12(18), 1–35 (2016)
25. Montanaro, A.: Quantum pattern matching fast on average. Algorithmica 77(1), 16–39 (2017)
26. Nielsen, M.A., Chuang, I.L.: Quantum computation and quantum information. Cambridge university press (2010)
27. Odeh, A., Abdelfattah, E.: Quantum sort algorithm based on entanglement qubits {00, 11}. In: 2016 IEEE Long Island Systems, Applications and Technology Conference (LISAT). pp. 1–5. IEEE (2016)

28. Odeh, A., Elleithy, K., Almasri, M., Alajlan, A.: Sorting n elements using quantum entanglement sets. In: Third International Conference on Innovative Computing Technology (INTECH 2013). pp. 213–216. IEEE (2013)
29. Ramesh, H., Vinay, V.: String matching in $o(\sqrt{n}+\sqrt{m})$ quantum time. Journal of Discrete Algorithms 1(1), 103–110 (2003)
30. Williams, J.W.J.: Algorithm 232 - heapsort. Commun. ACM 7(6), 347–349 (Jun 1964)

Quantum Random Number Generation with the Superconducting Quantum Computer IBM 20Q Tokyo

Kentaro Tamura[1] and Yutaka Shikano[2,3][0000−0003−2107−7536]

[1] Department of Applied Physics and Physico-Informatics, Keio University,
3-14-1 Hiyoshi, Kohoku, Yokohama, 223-8522 Japan
`cicero@keio.jp`
[2] Quantum Computing Center, Keio University,
3-14-1 Hiyoshi, Kohoku, Yokohama, 223-8522 Japan
`yutaka.shikano@keio.jp`
[3] Institute for Quantum Studies, Chapman University,
1 University Dr., Orange, CA 92866 USA

Abstract. Quantum random number generators (QRNGs) produce theoretically unpredictable random numbers. A typical QRNG is implemented in quantum optics [Herrero-Collantes, M., Garcia-Escartin, J. C.: Quantum Random Number Generators. Rev. Mod. Phys. **89**, 015004 (2017)]. Quantum computers become QRNGs when given certain programs. The simplest example of such a program applies the Hadamard gate on all qubits and performs measurement. As a result of repeatedly running this program on a 20-qubit superconducting quantum computer (IBM 20Q Tokyo), we obtained a sample with a length of 43,560. However, statistical analysis showed that this sample was biased and correlated. One of the post-processed samples passed statistical tests. To show the effectiveness of post-processing, a larger sample size is required. The present study of quantum random number generation and statistical testing may provide a potential candidate for benchmarking tests of actual quantum computing devices.

Keywords: true number generator· quantum random number generator· IBM Q· randomness extraction· min-entropy.

1 Introduction

Random numbers are a fundamental resource in various fields of science and industry [1]. This is for two reasons: Firstly, random numbers are used to simulate stochastic phenomena such as Brownian motion. Secondly, random numbers are a source of unpredictability. Examples of such applications are below.

Cryptography. Binary information can be encrypted with the use of random numbers. By XORing the binary representation of a message with a random number sequence of equal length, an information-theoretically secure encrypted message is obtained.

Simulation. Random numbers are used to simulate random events such as Brownian motion.

Gambling. Various situations in gambling require randomness. To simulate a fair shuffle of a deck of cards, for example, random numbers are essential.

Sampling. In situations where samples need to be extracted from a group, random numbers enable unbiased and uncorrelated extraction.

Given the wide variety of applications above, the generation of random numbers is still an ongoing field of research.

There are three main types of random number generators (RNGs): pseudorandom number generators (PRNGs), classical physical random number generators, and quantum physical random number generators (QRNGs). PRNGs adopt deterministic functions that produce seemingly random bits when given an initial value called a seed. There are several advantages to PRNGs. Firstly, they can be easily implemented on the classical computer. Secondly, one needs only to store the function and the seed in order to reproduce the output. Finally, the generation rate is generally faster than other types of random number generators. The problem with PRNGs is that the output can be predicted with knowledge of the seed and function. While PRNGs use deterministic functions to produce pseudorandom numbers, physical random number generators apply measurement to physical phenomena and convert the resulting values into bits. In the case of classical physical random number generators, the output can be predicted from the initial condition of the physical phenomena. QRNGs, on the other hand, produce theoretically unpredictable bits. The reason behind this is that under the axioms of quantum physics, the measurement outcome is probabilistic.

Quantum computers can be taken as QRNGs when assigned certain programs. In this study, we experimented with a program that applies the Hadamard gate and performs measurement on all qubits available. A sample sequence with a length of 43,560 was obtained by repeatedly running this program on IBM's superconducting computer (IBM 20Q Tokyo). It became clear through statistical tests that this sample was biased and correlated. In order to see whether the bias and correlation can be corrected, we applied several post-processors. Among the ones we applied, the combination of Samuelson's extractor and von Neumann's extractor produced a sample that passed the six statistical tests. As the combination of post-processing significantly reduced the sample size, the effectiveness is of the post-processor is debatable.

The existence of bias and correlation in the quantum random numbers generated by the quantum computer carries information about the device. Our next task is to determine the reason behind these flaws.

The rest of this paper is organized as follows. Section 2 briefly reviews randomness, random number generators, tests for randomness, and post-processing. We present the procedure for quantum random number generation on the IBM 20Q Tokyo and show the test results for the sample before and after post-processing in Sec. 3. An interpretation for the results are also discussed. Section 4 is devoted to the summary. In Sec. 5, we discuss some open questions on QRNGs.

2 Preparation

2.1 Randomness

The key to understanding randomness is to acknowledge the distinction between the following two types of randomness.

Product randomness. The apparent randomness that can be statistically identified by examining the output of a random process is called product randomness. Product randomness is practically important, as the length of a random number sequence is always finite. To check whether a coin toss yields a random number sequence, one can toss the coin a finite number of times and see if the heads and tails outcomes are unbiased and independent. This is the idea of product randomness.

Process randomness. The inherent randomness of the process of generation is called process randomness, which is an important factor of a true random number generator. In the example of a coin toss, process randomness corresponds to a symmetrically designed coin and a fair tossing process. By making sure that the process is random, one can expect the output to have product randomness as well.

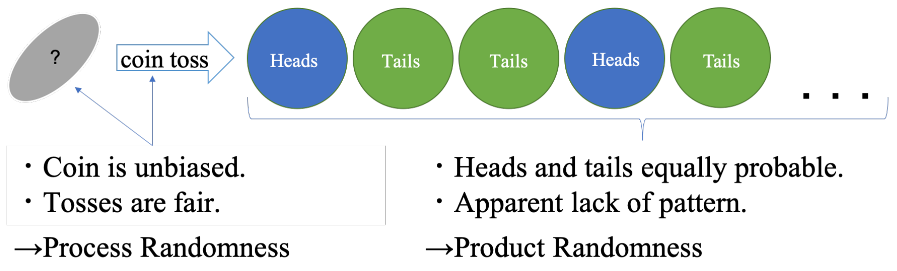

Fig. 1. Product randomness and process randomness of a coin toss.

While the goal of random number generation is to identify whether a generator is capable of producing random numbers, one can only observe finite samples of outputs. In practice, therefore, a generator is characterized from the samples obtained. For example, to check whether a coin toss is fair, one must toss the coin repeatedly and look for signs of bias and correlation.

2.2 Essential Characteristics of Random Number Generators

There are three main qualities that a random number generator must possess in order to qualify as legitimate.

Uniformity. A coin (random variable) with uniformity yields heads "1" and tails "0" with equal probability.

Independence. Independent coin tosses (trials) are such that any toss does not affect any of the other tosses. In other words, the tosses are not correlated.

Unpredictability. Unpredictability means that it is impossible to predict any of the future outcomes.

Ideally, every random number generator should possess all qualities shown above. However, only quantum random number generators possess all three qualities. Because one can only ever hope to examine a random number sequence of finite length, identifying the qualities above can only be done by applying randomness tests to the output.

2.3 Types of Random Number Generators

Random number generators can be classified into the following groups. Each group has its advantages and disadvantages, and the means of generation is chosen in light of what application the user has in mind. Only the final group can be considered true random number generators, which produce uniform, independent, and unpredictable bits.

Pseudorandom Number Generators The present computer (classical computer) is deterministic due to the deterministic principles of classical physics. As a result, classical computers are incapable of producing true random numbers. In order to overcome this problem, pseudorandom numbers have been used as a substitute for true random numbers.

The idea behind pseudorandom numbers is that as far as product randomness is concerned, a deterministic function that produces seemingly random outputs suffices for some applications. For instance, random numbers used in simulation need not be unpredictable.

While pseudorandom numbers are predictable and do not possess process randomness, advantages do exist: pseudorandom numbers are reproducible. This means that one needs only to remember the initial value and generation process to obtain the same sequence, which is not the case with true random numbers. In other words, pseudorandom numbers can be compressed. Under circumstances where the same random number sequence is required multiple times, pseudorandom numbers are convenient. Another advantage is that the generation of pseudorandom numbers is generally faster than physical random numbers [1]. This is convenient for simulation purposes where long sequences of random numbers are required.

Despite such advantages, there exist applications where pseudorandom numbers cannot be used as a substitute for true random numbers. These applications require unpredictability or fairness. In cryptography, for example, it is crucial that the future values are unpredictable. This cannot be achieved by pseudorandom numbers alone. Furthermore, pseudorandom numbers are not fair. This means that with knowledge of the initial value called a seed along with the function, one can easily manipulate the output. This is why pseudorandom numbers should not be used for choosing winning lottery tickets.

In areas where unpredictability or fairness is required, true random numbers cannot be replaced with pseudorandom numbers.

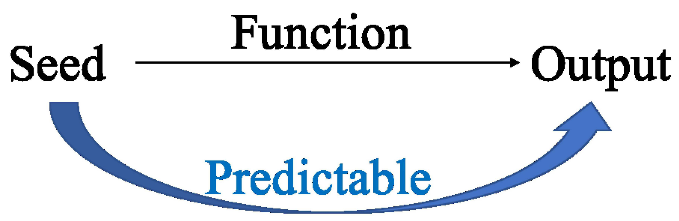

Fig. 2. Conceptual diagram of pseudorandom number generation.

As shown in Fig. 2, pseudorandom number generators are functions that when given an initial value called a seed, the rest of the output is automatically determined in the form of a recurrence formula. Random numbers produced in this manner have patterns (the generation formula itself) and periods. It is thus crucial that the user understands the limitations of pseudorandom number generators. For a better grasp of this concept, several examples of such functions or algorithms are presented in this section.

Middle-Square Method. The middle-square method is one of the first methods of pseudorandom number generation. It produces pseudorandom numbers based on the following algorithm. In this example, the seed is a decimal integer with 4 or more digits. The resulting sequence consists of 4-digit decimal integers.

1. Square the seed.
   e.g.) If seed = 12345, seed^2 = 152399025.
2. Add zeroes at the beginning of the outcome until the number of digits becomes an even number of 4 or larger.
   e.g.) 152399025 has 9 digits, so a single 0 is added. The number becomes 0152399025.
3. Extract the middle four digits and assign it as the new seed.
   e.g.) 015**2399**025 → **2399**. New seed = 2399.
4. Go back to initial step.

The problem with this method is that firstly, once the seed degenerates to zero, the following numbers will all be zero and secondly, that the sequence can end up in a cycle (e.g. 6100 → 2100 → 4100 → 8100 → 6100) [2].

Pseudorandom numbers are predictable, periodic, and correlated. While pseudorandom numbers produced by more recent methods are less predictable and have longer periods, they still cannot be considered true random numbers.

Classical Physical Random Number Generators The simplest form of classical random number generation is through coin tosses: when the coin yields heads the generator outputs 1, and when the coin yields tails it outputs 0. This example captures the essence of classical physical random number generation, which is that a classical physical phenomena is measured and encoded to binary numbers.

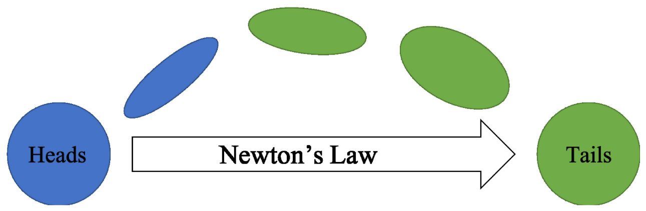

Fig. 3. Conceptual diagram of classical physical sources.

In classical physical random number generation, the initial condition of the physical phenomena is the seed. The problem is that in classical physics, all future measurement values are determined by this initial condition. Therefore, like pseudorandom numbers, classical physical random numbers are deterministic and thus predictable in theory [1].

Quantum Physical Random Number Generators A quantum physical random number generator is similar to its classical counterpart. The only difference is that the source is quantum physical instead of classical. Unlike classical physics, the measurement results in quantum physics cannot be determined by the initial condition [1]. This is due to the stochastic laws of quantum physics which state that the measurement results are not predetermined.

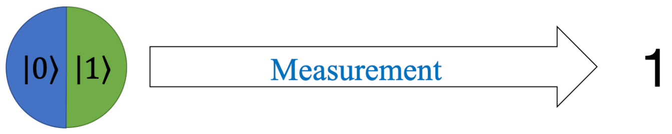

Fig. 4. Conceptual diagram of quantum physical sources.

A qubit is a quantum coin. While a classical coin is either heads or tails at all times, a qubit can be in multiple states at once (superposition). For example, Figure 4 shows a qubit that is initially prepared as a superposition between $|0\rangle$ and $|1\rangle$. Note that $|0\rangle$ is a state that when measured, its measurement value is 0 with certainty. Likewise, $|1\rangle$ always yields 1. One can control the probability of 0s and 1s at will, creating a quantum coin whose output is unpredictable.

Various quantum phenomena can be considered qubits. One example is the two paths of a photon after passing through a beam splitter as shown in Fig. 5.

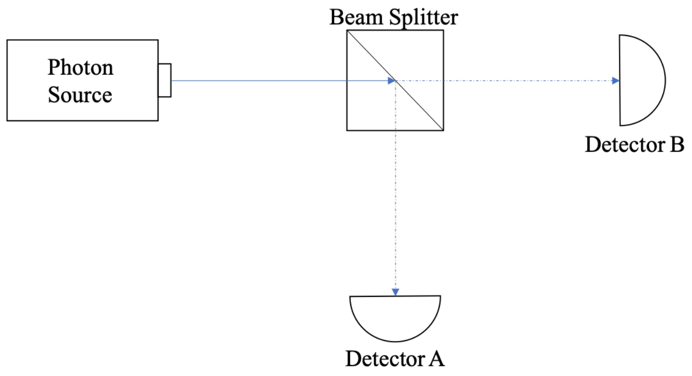

Fig. 5. Quantum random number generation using a photon source [1].

When a photon is detected by detector A, the output is 1. If it is detected by detector B, the output is 0. Optical qubits are a popular means of quantum random number generation. Firstly, optical qubits are easy to prepare compared to other types such as superconducting qubits. Secondly, optical methods tend to achieve a higher generation rate compared to other quantum physical methods. As large samples are required in both application and randomness testing, optical qubits are convenient.

2.4 Tests of Randomness

The goal of randomness testing is to statistically evaluate the generator's capability of producing unbiased and uncorrelated bits. This is done by examining the generator's output. Common test suites are the NIST test suite [3], TestU01 [4], and the dieharder test [5].

Let there be a coin with certain probabilities for yielding heads and tails. By flipping the coin 10 times, one obtains a sequence of heads and tails of length 10. Randomness tests aim to decide whether the sequence in question was obtained from an ideal random number generator that outputs unbiased and uncorrelated bits. This is done with the following steps.

1. Obtain a sample of appropriate length.
   e.g.) 0111111111
2. Characterize the sample with a test statistic.
   e.g.) the number of 1s within the sequence is 9.
3. Calculate the probability (p-value) that an ideal random number generator outputs a sample with such a test statistic.
   e.g.) the probability that an ideal random number generator outputs a sequence of length 10 containing 9 ones is approximately 0.009765625.
4. Decide whether the probability (p-value) is acceptable by comparing it to the level of significance α.
   e.g.) if $\alpha = 0.01$, then p-value $< \alpha$. In this case, the test result is that the sample was not obtained from an ideal random number generator.

One could apply various tests by characterizing the sample using various test statistics. One must, however, keep in mind that the ideal probability distribution of the test statistic must be known beforehand.

2.5 Post-processing

True random number generators have physical sources. The process of physical random number generation begins with the measurement of some observable value of a physical system as shown in Fig. 6. At this stage, the effect of classical noise becomes a matter of concern. The measured values are then encoded into binary information. At this encoding stage, errors must be taken into consideration. As a result of these properties, physical random number generators require post-processing where noise and errors are excluded as much as possible.

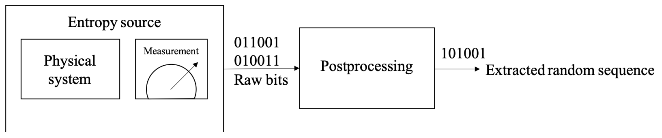

Fig. 6. The process of random number generation based on physical sources [1].

von Neumann Extractor The simplest example of a randomness extractor would be the von Neumann extractor. This extractor interprets the input binary sequence as pairs and translates 01s to 0s and 10s to 1s [6]. This extractor can only be effectively used for certain sources, namely, independent and identically distributed (iid)-bit sources [7]. Iid-bit sources are sources that consist of discrete random variables $X_1, X_2, \cdots, X_n \in \{0, 1\}$ where for any $i \in \{1, \cdots n\}$, $Pr[X_i = 1] = \delta$ for some unknown constant δ[9, Chapter 6]. A coin toss with a constant bias throughout the tosses is, for instance, an iid-bit source. Such a source should yield 01s and 10s with equal probability, hence the method turns the biased sequence into a uniform one.

Samuelson Extractor The Samuelson extractor is similar to the von Neumann extractor, except that it translates 10s to 0s and 11s to 1s [8]. It is often the case that the use of the Samuelson extractor is followed by the use of the von Neumann extractor. While using both extractors reduces the sequence to 1/4 on average[9, Chapter 6], this is sometimes necessary as the Samuelson extractor targets different sources compared to the von Neumann extractor. This extractor is effective when used on a source where the previous bit affects the probability of the next bit. For example, if a source is more likely to produce $0 \to 1$ than $1 \to 0$, the von Neumann extractor cannot be used. The Samuelson extractor, however, fixes the previous bit to 1, which turns the input into a sequence expected from an iid-bit source.

3 Experimental Procedure and Results

In light of the recent trend of quantum random number generation and quantum computing, we wonder if actual quantum computers are capable of producing random numbers. The following experiment aims to answer this question.

3.1 Random Number Generation

In this section, the procedure for generating random numbers on a quantum computer is introduced. A quantum computer is a device that contains multiple qubits. The IBM 20Q Tokyo device contains 20 qubits. Below are the steps for random number generation.

1. Prepare all qubits in the state of superposition between state $|0\rangle$ and $|1\rangle$, so that both states have equal probabilities. This can be achieved with the use of the Hadamard gate.
2. Measure all qubits once, and collect the result.
   e.g.) $|00101101 \cdots 1\rangle$.
3. Return to step 1.

By repeating the procedure above and concatenating the results in order, a sequence with a length of 43,560 was obtained. This procedure was conducted in July of 2018. It took approximately one month to obtain a sequence with a length of 43,560. Let us call this raw sample rand43560.

When the user specifies how many times which algorithm should be run, the IBM 20Q Tokyo device returns a histogram showing the probability of each measurement outcome. In order to obtain a sequence of measurement outcomes, we set the number of runs at 1 and kept running the same algorithm. This way, the histogram shows the measurement outcome of each run.

3.2 Post-Processing

The three post-processing extractors presented below were used.

1. The von Neumann Extractor: 01 $\rightarrow$ 0, 10 $\rightarrow$ 1.
   Sample name after post-processing: Neumann
2. The Samuelson Extractor: 10 $\rightarrow$ 0, 11 $\rightarrow$ 1.
   Sample name after post-processing: Samuelson
3. A combination of the above: Samuelson $\rightarrow$ Neumann.
   Sample name after post-processing: SN

The resulting samples were statistically examined along with the raw sample rand43560.

3.3 Test Results of Post-Processed Samples

The first 6 tests from the NIST test suite were applied to the respective samples. Figure 7 shows the result of the frequency test, which is a test for uniformity. The test statistic z is defined as below, and is known to follow a standard normal distribution.

$$z \equiv \frac{x - np}{\sqrt{np(1-p)}} \tag{1}$$

Note that x is the number of 1s in the sample, n is the length of the sample, and p is the ideal probability of 1s, which is 0.5. z yields 0 for a sample that contains equal proportions of 0s and 1s.

Figure 7 shows the values of z for each sample, as well as the corresponding probability densities. The probability that an ideal random number generator generates a sample with a value of z falling into the regions colored in green is lower than 1 %. When the value of z is in the green region, the sample is not uniform. This decision is based on a level of significance of $\alpha = 0.01$.

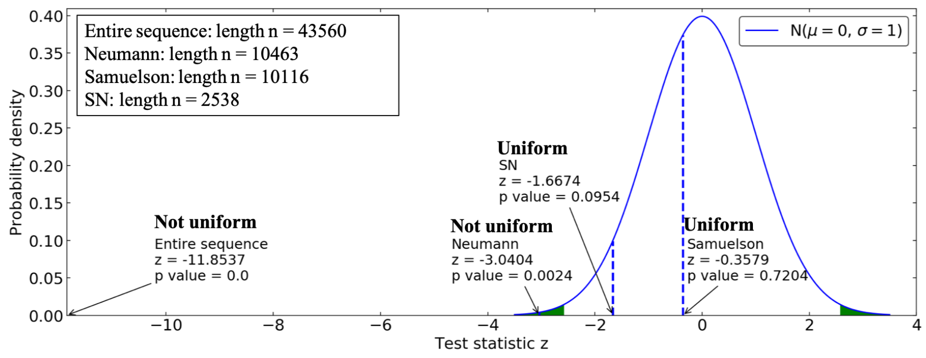

Fig. 7. Uniformity of rand43560 before and after randomness extraction.

We observe that the raw sample rand43560 is not uniform. This points to the possibility that the qubits were not prepared to yield equal probabilities of 0s and 1s, which provides insight into the accuracy of the gates and measurement of the device. Another reason is that the device is unstable. With the device going through calibration on a daily basis, it is questionable whether the device maintains the same properties from one day to another.

The fact that the von Neumann extractor failed to sufficiently correct the bias in the sequence suggests that the 20 qubits cannot be considered an iid-bit source.

The Samuelson extractor and the combination of the Samuelson extractor and the von Neumann extractor, on the other hand, proved effective. The effectiveness of the former extractor implies a Markovian property in the sequence. This means that the output of adjacent qubits tend to be correlated. The latter extractor is not as effective as the former, which suggests that even after the Samuelson extractor is applied, the sequence is still not iid.

Table 1. The NIST test results and corresponding p-values of samples.

Test	Rand43560	Neumann	Samuelson	SN
Length	43560	10463	10116	2538
Frequency	0.0000	0.0024	0.7204	0.0954
Frequency (block)	0.0000 (396)	0.0052 (126)	0.0161 (85)	0.2099 (22)
Runs	0.0000	0.1135	0.2333	0.7388
Runs (block)	0.0000 (128)	0.3125 (128)	0.0325 (128)	0.3952 (8)
Matrix rank	0.5285	0.9523	0.7138	0.7419
DFT	1.0000	0.1493	0.0003	0.4072
Result	Fail	Fail	Fail	Pass

Table 1 shows the p-values corresponding to the 6 tests from the NIST test suite of the respective samples. If a p-value is lower than $\alpha = 0.01$, the test is a failure. Unless the sample passes all six tests, it is not regarded as random. Block means that the sequence was divided into blocks of certain lengths, which are shown next to the p-values in brackets.

According to Table 1, only sample SN passes all 6 tests. This could be due to the fact that the size of the sample is small. In general, smaller samples are more likely to pass these tests.

4 Conclusion

The sequence of length 43,560 was obtained by repeatedly performing a quantum coin toss using all 20 qubits of the IBM 20Q Tokyo device. Following the conventional treatment of physical random number generation, three post-processing extractors were applied to the raw sample. The extractors are the von Neumann extractor, the Samuelson extractor, and the two combined.

As a result of applying the first 6 tests from the NIST test suite, it was revealed that only the combination of the von Neumann and Samuelson extractors succeeded in producing a sample that passes all 6 tests. Due to the limited sample size, however, it was not clear whether this method was effective.

Regarding the generation rate (approximately 43,560 digits per month) and precision of the device, using the IBM 20Q Tokyo device as a QRNG is not a practical idea. However, examining the quality of the output quantum random numbers may lead to benchmarking of actual quantum computing devices.

5 Open Questions

While using the IBM 20Q Tokyo device as a quantum random number generator is impractical, producing quantum random numbers with the device may not be an entirely pointless attempt.

In theory, quantum computers should be capable of producing quantum random numbers without post-processing. Therefore, the reason why the IBM 20Q Tokyo device failed to achieve this is worth pursuing.

A quantum computer is a multipurpose device, and judging its condition in light of a single purpose is of no use. However, there is a need for a comprehensive indicator of the device's condition, such that the user could refer to that indicator when deciding which device to use and when to use it. We believe that the quality of quantum random numbers produced by the device is a potential candidate of such an indicator.

The following are some open questions.

1. How can the quality of quantum random numbers and generators be quantified and compared?
2. What is the relationship between the quality of the random numbers and the device's condition?
3. Given that the quality of the random numbers is poor, what can be done to improve their quality?
4. How does the poor quality of random numbers affect other algorithms run on the same device?
5. What is the relationship between randomness and quantum computers?

Acknowledgement

This work is partially supported by JSPS KAKENHI (Grant Nos. 17K05082 and 19H05156). The results presented in this paper were obtained in part using an IBM Q quantum computing system as part of the IBM Q Network. The views expressed are those of the authors and do not reflect the official policy or position of IBM or the IBM Q team.

References

1. Herrero-Collantes, M., Garcia-Escartin, J. C.: Quantum Random Number Generators. Rev. Mod. Phys. **89**, 015004 (2017)

2. Knuth, D. E.: The Art of Computer Programming, Volume 2: Seminumerical Algorithms. 3rd edn. Addison-Wesley Longman Publishing Co. Inc., Redwood City, CA, USA (1997)
3. Bassham, L. E., Rukhin, A. L., Soto, J., Nechvatal, J. R., Smid, M. E., Leigh, S. D., Levenson, M., Vangel, M., Heckert, N. A., Banks, D. L.: A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic, NIST Special Publication **800**, 163 (2001)
4. L'Ecuyer, P., Simard, R.: TestU01: A C library for empirical testing of random number generators. ACM Trans. Math. Softw. (TOMS) **33**, 22 (2007).
5. Brown, R. G., Eddelbuettel, D., Bauer, D.: `https://webhome.phy.duke.edu/~rgb/General/dieharder.php` (Version 3.31.1, checked by online on April 25, 2019).
6. von Neumann, J.: Various techniques used in connection with random digits. J. Res. Nat. Bur. Stand. Appl. Math. Series **3**, 36 – 38 (1951)
7. Kwok, S. H., Ee, Y. L., Chew, G., Zheng, K., Khoo, K., Tan, C. H.: A Comparison of Post-Processing Techniques for Biased Random Number Generators. In: Ardagna, C. A., Zhou, J. (eds.): Information Security Theory and Practice. Security and Privacy of Mobile Devices in Wireless Communication. WISTP 2011. Lecture Notes in Computer Science **6633**, pp. 175 – 190, Springer, Berlin, Heidelberg (2011)
8. Samuelson, P.: Constructing an Unbiased Random Sequence. J. Am. Stat. Assoc. **63**, 1526 – 1527 (1968)
9. Vadhan, S.: Pseudorandomness. Found. Trends Theor. Comput. Sci. **7**, 1–336 (2012)

Deterministic Construction of QFAs based on the Quantum Fingerprinting Technique

Mansur Ziatdinov[1] and Aliya Khadieva[1,2]

[1] Kazan Federal University, Kazan, Russia
[2] University of Latvia, Riga, Latvia
gltronred@gmail.com, aliya.khadi@gmail.com

Abstract. It is known that for some languages quantum finite automata are more effective than classical counterparts. Particularly, a QFA recognizing the language MOD_p has an exponential advantage over the classical finite automata. However, the construction of such QFA is probabilistic. In the current work, we propose a deterministic construction of the QFA for the language MOD_p. We construct a QFA for a promise problem $Palindrome_p$.
Keywords: quantum computing, automata, quantum automata, quantum fingerprinting, palindrome

1 Introduction

Quantum finite automata are a mathematical model for quantum computers with limited memory. A quantum finite automaton has a finite state space and applies a sequence of transformations corresponding to a letter of an input word to this state space. In the end, the state of the quantum automaton is measured, and the input word is accepted or rejected, depending on the outcome of the measurement.

Quantum automata are discussed and compared with classical ones in [13, 17, 20, 18, 16, 31, 27, 22, 19, 32, 15].

In 1998 Freivalds and Ambainis presented a technique for construction of quantum finite automata exponentially smaller than classical counterparts for several languages [13]. In this work, they used a fingerprinting approach for construction of a QFA. In papers [21, 14], Ambainis and Nahimovs used a similar technique and showed an improved exponential separation between quantum and classical finite automata. They constructed a QFA for the language MOD_p. Define the $MOD_p = \{a^i | i \text{ is divisible by } p\}$, where p is a prime.

In papers [21, 14] a construction of the QFA is probabilistic. It employs a sequence of parameters $(k_1 \cdots k_d)$ that are chosen at random and hardwired into the QFA. The idea of construction is the following. The QFA uses a register of d qubits. Reading input symbol the automaton rotates each qubit $|q_i\rangle$ on it's own angle depending on parameter $k_i \in \{k_1 \cdots k_d\}$. d depends on the probability of error. Existence of such set $\{k_1 \cdots k_d\}$ of parameters is proven in [21, 14] by using Azuma's theorem from Probability Theory.

In these papers ([21, 14]) authors gave two techniques of explicit finding this set. The first of them gives a QFA with $O(\log p)$ states, but only numerical experiments show the correctness of a QFA. The second one gives an explicit construction of a QFA with $O(\log^{2+\epsilon}(p))$ states, where $\epsilon > 0$ is a probability of error.

Vasiliev, Latypov and Ziatdinov [28] used simulated annealing and genetic heuristical algorithms for finding this set of required parameters $\{k_1 \cdots k_d\}$. The algorithm for construction of a QFA was generalized by Ablayev and Vasiliev [9]. They improved the technique and constructed an OBDD for a Boolean function $f = MOD_p(x_1, \ldots, x_n)$, where p is any positive integer, not only prime. An OBDD is a modification of automata where transition functions depend on a position of an input head, and in OBDD we can read input symbols in different orders [29]. Quantum and classical OBDDs were considered and compared in [4, 5, 3, 10, 11, 1, 23, 2, 24, 7, 6].

In the current paper, we consider the same language MOD_p and a QFA with $O(\log p)$ states recognizing this language. It uses a set of parameters mentioned above. However, we propose a deterministic algorithm for computing the required parameters. It allows an explicit construction of a QFA for MOD_p with $O(\log p)$ states. This technique is based on the results of [30]. We use a sequence of functions called pessimistic estimators. The theorem from [30] claims that if the sequence of such pessimistic estimators is defined, then the required set of parameters can be computed deterministically. Using this algorithm, we obtain explicit numbers as parameters for construction of the QFA.

Additionally, we construct a QFA for a $Palindrome_p$ language as one more example for using this technique. Thus, we propose that this approach can be used for solving a large class of languages.

The paper is organized in the following way. Section 2 contains some preliminaries and previous results on an improved construction of a QFA. In Section 3 we give a method of deterministic finding a set of parameters. In Section 4 there is one more example for applying the quantum fingerprinting technique. Here we describe a scheme of the QFA for the $Palindrome_p$ language. Section 5 is a conclusion.

2 Preliminaries

We use a definition of 1-way quantum finite automata (QFA) given in [25]. A 1-way QFA is a tuple $M = (Q, \Sigma, \delta, q_0, Q_{acc}, Q_{rej})$ where Q is a finite set of states, Σ is an input alphabet, δ is a transition function, $q_0 \in Q$ is a starting state, Q_{acc} and Q_{rej} are disjoint sets of accepting and rejecting states and $Q = Q_{acc} \cup Q_{rej}$. $\circledcirc$ and \$ are symbols that do not belong to Σ. We use $\circledcirc$ and \$ as the left and the right end markers, respectively. The working alphabet of M is $\Gamma = \Sigma \cup \{\circledcirc, \$\}$.

A superposition of M is any element of $l_2(Q)$ (the space of mappings from Q to $\mathbb{C}$ with l_2 norm). For $q \in Q$, $|q\rangle$ denotes the unit vector with value 1 at q and 0 elsewhere. All elements $|\psi\rangle$ of $l_2(Q)$ can be expressed as linear combinations of vectors $|q\rangle$. We will use $|\psi\rangle$ to denote elements of $l_2(Q)$.

A transition function δ maps $Q \times \Gamma \times Q$ to $\mathbb{C}$. The value $\delta(q_1, a, q_2)$ is an amplitude of $|q_2\rangle$ in the superposition of states to which M goes from $|q_1\rangle$ after reading a. For $a \in \Gamma$, V_a is a linear transformation on $l_2(Q)$ defined by

$$V_a(|q_1\rangle) = \sum_{q_2 \in Q} \delta(q_1, a, q_2)|q_2\rangle$$

We require all V_a to be unitary. The computation of the QFA starts from the superposition $|q_0\rangle$. Then transformations corresponding to the left end marker $\circledcirc$, the letters of the input word x and the right end marker \$ are applied. The transformation corresponding to $a \in \Gamma$ is just V_a. If the superposition before reading a is $|\psi\rangle$, then the superposition after reading a is $V_a(|\psi\rangle)$. After reading the right endmarker, the current state $|\psi\rangle$ is measured with respect to the observable $E_{acc} \oplus E_{rej}$ where $E_{acc} = span\{|q\rangle : q \in Q_{acc}\}$, $E_{rej} = span\{|q\rangle : q \in Q_{rej}\}$. This observation gives $|\psi\rangle \in E_i$ with the probability equal to the square of the projection of $|\psi\rangle$ to E_i.

After that, the superposition collapses to this projection. If we get $|\psi\rangle \in E_{acc}$, the input is accepted. If $|\psi\rangle \in E_{rej}$, the input is rejected.

Let L be a language in the alphabet Σ. We say a bounded error QFA recognizes a language L iff after reading an input word x the automaton terminates in an accepting state $\in Q_{acc}$.

- with probability more than $1 - \epsilon$ if $x \in L$
- with probability less than ϵ if $x \notin L$

2.1 Previous results

Let p be a prime. We consider the language $MOD_p = \{a^i | i \text{ is divisible by } p\}$. It is easy to see that any deterministic 1-way finite automaton recognizing MOD_p has at least p states. Ambainis and Freivalds in [13] constructed an efficient QFA. They have shown that a QFA with $O(\log p)$ states can recognize MOD_p with bounded error ϵ. A big-O constant in this result depends on ϵ. For $x \in MOD_p$, an answer is always correct with probability 1. There is a QFA with $16 \log_2 p$ states that is correct with probability at least 1/8 on inputs $x \notin MOD_p$. In a general case, [13] gives a QFA with $poly(1/\epsilon) \log_2 p$ states that is correct with probability at least $1-\epsilon$ on inputs $x \notin MOD_p$ (where $poly(z)$ is some polynomial in z).

The papers [21, 14] present a simple design of QFAs that achieve a better big-O constant. Ambainis and Nahimovs show that for any $\epsilon > 0$, there is a QFA with $4\frac{\log_2 2p}{\epsilon}$ states recognizing MOD_p with probability at least $1 - \epsilon$. Denote this QFA by M. It is constructed by combining d subautomata M_i, each has 2 states $q_{i,0}$ and $q_{i,1}$.

The set of states of M consists of $2d$ states $\{q_{1,0}, q_{1,1}, q_{2,0}, q_{2,1}, \cdots , q_{d,0}, q_{d,1}\}$. The starting state is $q_{1,0}$. The set of accepting states Q_{acc} consists of one state $q_{1,0}$. All other states $q_{i,j}$ belong to Q_{rej}. A transformation for the left endmarker $\circledcirc$ is such that $V_{\circledcirc}(|q_{1,0}\rangle) = \frac{1}{\sqrt{d}}(|q_{1,0}\rangle + |q_{2,0}\rangle + \cdots + |q_{d,0}\rangle)$.

A transformation for the input symbol a is

- $V_a(|q_{i,0}\rangle) = \cos\frac{2k_i\pi}{p}|q_{i,0}\rangle + \sin\frac{2k_i\pi}{p}|q_{i,1}\rangle$
- $V_a(|q_{i,1}\rangle) = -\sin\frac{2k_i\pi}{p}|q_{i,0}\rangle + \cos\frac{2k_i\pi}{p}|q_{i,1}\rangle$,

where $k_1, k_2, \cdots k_d$ is a sequence of numbers.

A transformation for the endmarker \$ is the following.

- The states $|q_{i,1}\rangle$ are not changed,
- $V_\$(|q_{i,0}\rangle) = \frac{1}{\sqrt{d}}|q_{1,0}\rangle$ plus some other state.
  In particular, $V_\$(\frac{1}{\sqrt{d}}(|q_{1,0}\rangle + |q_{2,0}\rangle + \cdots + |q_{d,0}\rangle)) = |q_{1,0}\rangle$.

If the input word is a^j and j is divisible by p, then M accepts the word with probability 1. If the input word is a^j and j is not divisible by p, then M accepts with probability

$$\frac{1}{d^2}\left(\cos\frac{2k_1\pi j}{p} + \cos\frac{2k_2\pi j}{p} + \cdots + \cos\frac{2k_d\pi j}{p}\right)^2.$$

In their proof, the authors use a theorem from Probability Theory (variant of Azuma's theorem):

Theorem 1. *[26] Let $X_1, \cdots, X_d$ be independent random variables such that $E[X_i] = 0$ and the value of X_i is always between -1 and 1. Then,*

$$Pr\left[|\sum_{i=1}^{d} X_i| \geq \lambda\right] \leq 2e^{-\frac{\lambda^2}{2d}}.$$

Define X_i as $\cos\frac{2k_i\pi j}{p}$ where each k_i is from $\{0, \cdots p-1\}$. By the theorem, it is possible to choose $k_1, \cdots, k_d$ values to ensure

$$\frac{1}{d^2}\left(\sum_{i=1}^{d}\cos\frac{2k_i\pi j}{p}\right)^2 < \epsilon. \qquad (1)$$

This inequality gives a bound for $d = 2^{\frac{\log_2 2p}{\epsilon}}$, a number of states for M is $4^{\frac{\log_2 2p}{\epsilon}}$.

The proposed QFA construction depends on d parameters and accepts an input word $a^j \notin MOD_p$ with probability

$$\frac{1}{d^2}\left(\sum_{i=1}^{d}\cos\frac{2k_i\pi j}{p}\right)^2$$

However, this proof is by a probabilistic argument and does not give an explicit sequence $k_1, \ldots, k_d$.

Two approaches for construction of specific sequences are presented in [21, 14].

The first one is based on numerical experiments and gives a QFA with $O(\log p)$ states. It is based on using so-called cyclic sequences $S_g = \{k_i = g^i$

mod $p\}_{i=1}^{d}$ where g is a primitive root modulo p. The authors checked all $p \in \{2, \ldots, 9973\}$, all generators g and all sequence lengths $d < p$. They experimentally compared two strategies: using a randomly chosen sequence and using a cyclic sequence. In $99.98\% - 99.99\%$ of the experiments, random sequences satisfied the theoretical upper bound (1), but cyclic sequences substantially outperformed random ones in almost all the cases. For some p, in 1.81% of the cases, the random sequences gave better values of ϵ. The numerical experiments showed that almost all the observed sequences satisfied the bound (1).

However, it is still open whether one could find the desired generator without an exhaustive search of all generators for all values of p.

The second approach gives an explicit construction of a QFA. This approach is based on a result of Ajtai et al. [12]. Nevertheless, the QFA has $O(\log^{2+3\epsilon} p)$ states that is more than $O(\log p)$.

3 Deterministic Algorithm for Construction of a QFA

In this section, we suggest an explicit algorithm for deterministic finding the set of parameters $S = (k_1, \ldots, k_d)$. By using this approach, we can explicitly construct a QFA for MOD_p with $O(\log p)$ states. The QFA is designed as in [21, 14]. However, the set S of required parameters is not randomly chosen, but deterministically computed.

The method is based on an algorithm from [30]. That is the explicit algorithm for deterministic construction of a small-biased set.

3.1 Definitions

Let us introduce some necessary definitions.

Definition 1. *Let us denote $[n]$ the set of integers $\{0, \ldots, n-1\}$.*

Definition 2. *We let $A \geq 0$ denote that A is positive semidefinite (p.s.d.) matrix (i.e. all of its eigenvalues are non-negative).*

Let A and B be symmetric matrices. Let $A \leq B$ denote that $B - A \geq 0$.

Let us denote $[A; B]$ the set of all symmetric matrices such that $A \leq C$ and $C \leq B$.

Definition 3 (Pessimistic estimators). *Let $\sigma : [p]^d \to \{0, 1\}$ be an event.*

Pessimistic estimators $\phi_0, \ldots, \phi_d$ are functions $\phi_i : [p]^i \to [0, 1]$, such that for each i and for each $x_1, \ldots, x_i \in [p]$:

$$\Pr_{X_{i+1},\ldots,X_d}[\sigma(x_1, \ldots, x_i, X_{i+1}, \ldots, X_d) = 0] \leq \phi_i(x_1, \ldots, x_i) \tag{2}$$

and

$$\mathbb{E}_{X_{i+1}}\left[\phi_{i+1}(x_1, \ldots, x_i, X_{i+1})\right] \leq \phi_i(x_1, \ldots, x_i) \tag{3}$$

3.2 Algorithm for Finding a Small-Biased Set

To compute a set of parameters we generalize a problem. Let H be a group and $\chi : H \to \mathbb{C}$ be a characteristic of H. Suppose that we are able to find a set $S \subset H$ such that

$$\frac{1}{|S|}\left|\sum_{s\in S}\chi(s)\right| \leq \epsilon.$$

If we take $H = \mathbb{Z}_p$ then $\chi(h) = \cos\frac{2\pi h}{p} + i\sin\frac{2\pi h}{p}$.
Therefore,

$$\begin{aligned}\epsilon &\geq \frac{1}{|S|}\left|\sum_{s\in S}\chi(s)\right| \\ &= \frac{1}{d}\left|\sum_{j=1}^{d}\cos\frac{2\pi k_j}{p} + i\sin\frac{2\pi k_j}{p}\right| \\ &\geq \frac{1}{d}\left|\sum_{j=1}^{d}\cos\frac{2\pi k_j}{p}\right|,\end{aligned}$$

where $S = \{k_1, \ldots, k_d\}$ is the desired set of parameters.
Given $\epsilon < 1$, we want to find a set $S \subset [p]^d$ such that

$$\frac{1}{|S|}\left|\sum_{s\in S}\chi(s)\right| \leq \epsilon,$$

where p is a non-negative number, d is a size of the desired set of parameters S, ϵ is a probability of error.

The solution for this problem is given by Wigderson [30] in the following theorem and its proof. Let us reformulate this theorem in our notation.

Theorem 2 ([30]). *Let $\sigma : [p]^d \to \{0, 1\}$ be an event.*
If there are pessimistic estimators $\phi_0, \ldots, \phi_d$ of σ, then we can efficiently find $x_1, \ldots, x_d$ by deterministic algorithm such that $\sigma(x_1, \ldots, x_d) = 1$.

Proof. Pick $x_1, \ldots, x_d$ one by one.

- At step 0 $\phi_0 < 1$.
- At step i we have $x_1, \ldots, x_i$ and $\phi_i(x_1, \ldots, x_i) < 1$. Enumerate over $x_{i+1} \in [p]$ and choose a value such that $\phi_{i+1}(x_1, \ldots, x_{i+1}) \leq \phi_i(x_1, \ldots, x_i) < 1$. An existence of x_{i+1} is guaranteed by inequality (3).
- At step d we have $x_1, \ldots, x_d$ and $\phi_d(x_1, \ldots, x_d) < 1$. By inequality (2), $\Pr[\sigma(x_1, \ldots, x_d) = 0] \leq \phi_d(x_1, \ldots, x_d) < 1$, therefore $\sigma(x_1, \ldots, x_d) = 1$.

The definition of required functions f can be found in [30, Theorem 5.2]. Let us reformulate it in our notation.

Theorem 3 ([30]). *Let $\gamma < 1$. Let H be a group with character $\chi : H \to \mathbb{C}$. Denote $p = |H|$.*

There exists a set S of size $|S| = O(\frac{1}{\gamma}\log n)$ such that

$$\frac{1}{|S|}\left|\sum_{s\in S}\chi(s)\right| \le \gamma,$$

and this set can be found deterministically.

Proof. Let P_h for $h \in H$ be a $p \times p$ permutation matrix of the action of h by right multiplication. Let matrix-valued $f : H \to [-I_p, I_p]$ be $f(h) = (I - J/p)\frac{1}{2}(P_h + P_{h^{-1}})$, where I is unit matrix and J is all-one matrix. Let $\sigma : [p]^d \to \{0, 1\}$ be the event $\sigma(x_1, \dots, x_d) = 1$ iff $\frac{1}{d}\sum_{i=1}^{d} f(x_i) \le \gamma I$. Let $t = \gamma/2$.

By [30, Theorem 4.1] the following functions are pessimistic estimators for $\sigma(x_1, \dots, x_d)$:

$$\phi_0 = pe^{-t\gamma d}\left|\left|\mathbb{E}[\exp(tf(X))]\right|\right|^{d} \le pe^{-\gamma^2 d/4}$$

$$\phi_i(x_1, \dots, x_i) = pe^{-t\gamma d}\operatorname{Tr}\left(\exp\left(t\sum_{j=1}^{i} f(x_j)\right)\right)\left|\left|\mathbb{E}[\exp(tf(X))]\right|\right|^{d-i}$$

Therefore, by Theorem 2, there exists polynomial algorithm to find $x_1, \dots, x_d$ such that $\sigma(x_1, \dots, x_d) = 1$.

So we get the following deterministic Algorithm 1 for computing a set of parameters. Here p is a prime, d is a size of the desired set of parameters S, ϵ is a probability of error.

See Appendix A for numerical results.

4 One More Example for Applying the Quantum Fingerprinting Technique

The paper [8] gives a notation of Equality-related problems in a context of quantum Ordered Binary Decision Diagrams. Authors apply a new modification of the fingerprinting technique to such problems as Equality, Palindrome, Periodicity, Semi-Simon, Permutation Matrix Test Function. We conclude that the given deterministic method of finding a required set of parameters works for the mentioned class of problems. In this section, we present a construction of a QFA for a promise problem $Palindrome_p$.

Define the promise problem $Palindrome_p$ as follows. Let p be some even integer, an input is a string X in an alphabet $\{0, 1, \#\}$. The input word is bordered by a left end marker $\circledcirc$ and a right end marker \$. We are promised that after each p input symbols belonging to $\{0, 1\}$ a symbol # appears. We split the input into so called subwords of size p divided by the symbol #.

$$x_1x_2\dots x_p\#x_{p+1}\dots x_{2p}\#\dots\#x_{ip+1}x_{ip+2}\dots x_{(i+1)p}\#\dots$$

Algorithm 1 Algorithm to find a set of parameters

```
Require: $p, d, \epsilon$
Ensure: $S = \{k_1, \dots, k_d\}$, s.t. $\frac{1}{d}\left|\sum_{i=1}^{d} \cos\frac{2\pi k_i}{p}\right| \le \epsilon$
  $t \leftarrow \epsilon/2$
  $f \leftarrow Z$
  for all $i \in \{1, \dots, p\}$ do
    $f[i] \leftarrow (I - J/n)(P_i + P_{i-1})/2$
  end for
  $S \leftarrow \{\}$
  for all $i \in \{1, \dots, d\}$ do
    $m \leftarrow 0$
    for all $j \in \{1, \dots, p-1\}$ do
      if $\phi_{i+1}(s_1, \dots, s_i, j) < \phi_i(s_1, \dots, s_i)$ then
        $m \leftarrow j$
      end if
    end for
    $S \leftarrow S \cup \{m\}$
  end for
  return $S$
```

Define these subwords as $\omega_0, \omega_1, \dots, \omega_i, \dots$. If a length of the input is not divisible by p, then the remaining part of the input is ignored.

Define $Palindrome(x_1, x_2, \dots x_p) \equiv (x_1 x_2 \dots x_{p/2} = x_{p/2+1} x_{p/2+2} \dots x_p)$.

$$Palindrome_p \equiv [Palindrome(\omega_0) \& Palindrome(\omega_1) \& \dots]$$

For solving the promise problem $Palindrome_p$, we consider a QFA that allows a measuring several times. Construction of the automaton is based on the technique described above. Let the automaton be defined by P. Here we use a quantum register $|\psi\rangle$ of $t = \lceil \log_2 d \rceil + 1$ qubits, where $d = 2\frac{\log_2 2p}{\epsilon}$ is a size of the set of parameters S. An additional register $|\phi\rangle$ of $\lceil \log_2 p \rceil$ qubits is needed for storing an index of an observing symbol in a subword. The QFA P consists of $2^t \cdot p \approx 2dp$ states.

$$|\psi_1 \psi_2 \dots \psi_{t-1}\rangle |\psi_{target}\rangle |\phi\rangle = |00\dots0\rangle|0\rangle|0\rangle$$

is an initial state. An accepting state is $|00\dots0\rangle|0\rangle|0\rangle$. All other states are rejecting.

Reading the left end marker P maps the initial state to a superposition of d states by applying the Hadamard transformation for the first $t-1$ qubits.

$$|00\dots0\rangle|0\rangle|0\rangle \to \frac{1}{\sqrt{d}} \sum_{i=0}^{d} |i\rangle|0\rangle|0\rangle$$

The register $|\phi\rangle$ is changed after reading a symbol x_j of a subword by applying a transformation $U : |j\rangle \to |(j+1)\ mod\ p\rangle$, where $j \in \{0, \dots, p-1\}$. A transformation $p \times p$ matrix U is the following.

$$U = \begin{bmatrix} 0\,0 \cdots 0\,1 \\ 1\,0 \cdots 0\,0 \\ 0\,1 \cdots 0\,0 \\ \cdots \\ 0\,0 \cdots 1\,0 \end{bmatrix}$$

On a symbol $x_j = 1$ the qubit $|\psi_{target}\rangle$ is transformed by a rotating G on an angle $\frac{2\pi k_i 2^j}{2^{p/2}}$ if $j < p/2 \pmod p$, and on an angle $-\frac{2\pi k_i 2^{p-j-1}}{2^{p/2}}$ if $j \geq p/2 \pmod p$. The transformation is the following.

$$G = \begin{bmatrix} \cos\alpha & \sin\alpha \\ -\sin\alpha & \cos\alpha \end{bmatrix},$$

where $\alpha = \frac{2\pi k_i 2^j}{2^{p/2} p}$ if $j < p/2 \pmod p$,

$\alpha = -\frac{2\pi k_i 2^{p-j-1}}{2^{p/2}}$ if $j \geq p/2 \pmod p$.

On $x_j = 0$ the system is not transformed.

The set $S = \{k_1, k_2 \dots k_d\}$ can be computed in a way described in the previous section.

Reading the symbol # the automaton maps the first $t-1$ qubits to the $|00\dots0\rangle$ by applying the Hadamard transformation, and $|\psi_{target}\rangle$ is measured. If the measured value is $|0\rangle$, then the computation is continued on the next subword starting from the same initial state $|00\dots0\rangle|0\rangle|0\rangle$. If no, then P stops and rejects the input with probability 1.

Theorem 4. *If the input word $x \in Palindrome_p$, then P accepts it with probability* 1. *If $x \notin Palindrome_p$, then P accepts it with probability ϵ.*

Proof. If $x \in Palindrome_p$, then each of its subwords is a palindrome. Initially, $|\psi_{target}\rangle = |0\rangle$. By the construction of the QFA, after reading each ω_i, the automaton P maps $|\psi_{target}\rangle$ to itself. After reading the right end-marker, P gets into the state $|00\dots0\rangle|0\rangle|0\rangle$. That is the only accepting state.

If $x \notin Palindrome_p$, then at least one subword of the input is a non-palindrome. Let it be ω_g. After reading ω_g and measuring $|\psi_{target}\rangle$, we get $|0\rangle$ with probability

$$\frac{1}{d^2}\left(\sum_{i=0}^{d}\left(\sum_{j=0}^{p/2-1}\cos\frac{2\pi k_i 2^j}{2^p} + \sum_{j=p/2}^{p-1}\cos\frac{2\pi k_i 2^{p-j-1}}{2^p}\right)\right)^2 < \epsilon.$$

Thus, P accepts the wrong word with the probability $< \epsilon$.

If there are m such wrong subwords in the input, and measurements of $|\psi_{target}\rangle$ are independent, then a probability of acceptance of the wrong input is less than $\epsilon^m < \epsilon$.

5 Conclusion

This technique can be applied for a QFA solving a promise problem $Palindrome_p$. This result shows the effectiveness of quantum approach for solving the class of Equality-related problems.

Acknowledgements

We thank Kamil Khadiev and Abuzer Yakaryılmaz for fruitful discussions and help. The reported study was funded by RFBR according to the research project No.19-37-80008.

References

1. Ablayev, F., Ablayev, M., Khadiev, K., Vasiliev, A.: Classical and quantum computations with restricted memory. LNCS 11011, 129–155 (2018)
2. 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 SOFSEM, LNCS 10706, 197–211 (2018)
3. Ablayev, F., Gainutdinova, A.: Complexity of quantum uniform and nonuniform automata. In: Developments in Language Theory. LNCS, vol. 3572, pp. 78–87. Springer (2005)
4. Ablayev, F., Gainutdinova, A., Karpinski, M.: On computational power of quantum branching programs. In: FCT. LNCS, vol. 2138, pp. 59–70. Springer (2001)
5. Ablayev, F., Gainutdinova, A., Karpinski, M., Moore, C., Pollett, C.: On the computational power of probabilistic and quantum branching program. Information and Computation 203(2), 145–162 (2005)
6. Ablayev, F., Gainutdinova, A., Khadiev, K., Yakaryılmaz, A.: Very narrow quantum OBDDs and width hierarchies for classical OBDDs. Lobachevskii Journal of Mathematics 37(6), 670–682 (2016), http://dx.doi.org/10.1134/S199508021606007X
7. Ablayev, F., Gainutdinova, A., Khadiev, K., Yakaryılmaz, A.: Very narrow quantum OBDDs and width hierarchies for classical OBDDs. In: DCFS, LNCS, vol. 8614, pp. 53–64. Springer (2014)
8. Ablayev, F., Khasianov, A., Vasiliev, A.: On complexity of quantum branching programs computing equality-like boolean functions. ECCC (2010)
9. Ablayev, F., Vasilyev, A.: On quantum realisation of boolean functions by the fingerprinting technique. Discrete Mathematics and Applications 19(6), 555–572 (2009)
10. Ablayev, F.: On computational power of classical and quantum branching programs. vol. 6264, pp. 6264 – 6264 – 10 (2006), https://doi.org/10.1117/12.683111
11. Ablayev, F., Moore, C., Pollett, C.: Quantum and stochastic branching programs of bounded width. In: Automata, Languages and Programming. pp. 343–354. Springer Berlin Heidelberg, Berlin, Heidelberg (2002)
12. Ajtai, M., Iwaniec, H., Komlós, J., Pintz, J., Szemerédi, E.: Construction of a thin set with small fourier coefficients. Bulletin of the London Mathematical Society 22(6), 583–590 (1990)

13. Ambainis, A., Freivalds, R.: 1-way quantum finite automata: strengths, weaknesses and generalizations. In: FOCS'98. pp. 332–341. IEEE (1998)
14. Ambainis, A., Nahimovs, N.: Improved constructions of quantum automata. Theoretical Computer Science 410(20), 1916–1922 (2009)
15. Ambainis, A., Yakaryılmaz, A.: Superiority of exact quantum automata for promise problems. Information Processing Letters 112(7), 289–291 (2012)
16. Ambainis, A., Yakaryılmaz, A.: Automata and quantum computing. Tech. Rep. 1507.01988, arXiv (2015)
17. Ambainis, A., Beaudry, M., Golovkins, M., Ķikusts, A., Mercer, M., Thérien, D.: Algebraic results on quantum automata. Theory of Computing Systems 39(1), 165–188 (2006)
18. Ambainis, A., Bonner, R., Freivalds, R., Ķikusts, A.: Probabilities to accept languages by quantum finite automata. In: Computing and Combinatorics. Lecture Notes in Computer Science, vol. 1627, pp. 174–183. Springer Berlin / Heidelberg (1999)
19. Ambainis, A., Ķikusts, A.: Exact results for accepting probabilities of quantum automata. Theoretical Computer Science 295(1-3), 3–25 (2003)
20. Ambainis, A., Ķikusts, A., Valdats, M.: On the class of languages recognizable by 1-way quantum finite automata. In: STACS 2001: 18th Annual Symposium on Theoretical Aspects of Computer Science. pp. 75–86 (2001)
21. Ambainis, A., Nahimovs, N.: Improved constructions of quantum automata. In: TQC. LNCS, vol. 5106, pp. 47–56. Springer (2008)
22. Gainutdinova, A., Yakaryılmaz, A.: Unary probabilistic and quantum automata on promise problems. Quantum Information Processing 17(2), 28 (2018)
23. Ibrahimov, R., Khadiev, K., Prūsis, K., Yakaryılmaz, A.: Error-free affine, unitary, and probabilistic OBDDs. Lecture Notes in Computer Science 10952 LNCS, 175–187 (2018)
24. Khadiev, K., Khadieva, A.: Reordering method and hierarchies for quantum and classical ordered binary decision diagrams. In: CSR 2017, LNCS, vol. 10304, pp. 162–175. Springer (2017)
25. Moore, C., Crutchfield, J.P.: Quantum automata and quantum grammars. Theoretical Computer Science 237(1-2), 275–306 (2000)
26. Motwani, R., Raghavan, P.: Randomized algorithms. Cambridge university press (1995)
27. Say, A.C., Yakaryılmaz, A.: Quantum finite automata: A modern introduction. In: Computing with New Resources, pp. 208–222. Springer (2014)
28. Vasiliev, A., Latypov, M., Ziatdinov, M.: Minimizing collisions for quantum hashing. Journal of Engineering and Applied Sciences 12(4), 877–880 (2017)
29. Wegener, I.: Branching Programs and Binary Decision Diagrams: Theory and Applications. SIAM (2000)
30. Wigderson, A., Xiao, D.: Derandomizing the ahlswede-winter matrix-valued chernoff bound using pessimistic estimators, and applications. Theory of Computing 4, 53–76 (2008)
31. Yakaryılmaz, A., Say, A.C.C.: Languages recognized by nondeterministic quantum finite automata. Quantum Information and Computation 10(9&10), 747–770 (2010)
32. Zheng, S., Qiu, D., Gruska, J., Li, L., Mateus, P.: State succinctness of two-way finite automata with quantum and classical states. Theoretical Computer Science 499, 98–112 (2013)

A Numerical Results

All computation were performed on the following computer:

CPU: Intel Core i7-5930K with 6 cores (12 threads) and 3.50 GHz frequency
RAM: 64 Gb
GPU: NVIDIA GPU GeForce GTX 1080 (GP104-A) with 8 Gb memory
OS: Ubuntu 18.04.1 LTS
Octave version: 4.2.2
GPU drivers: nvidia-drivers-415.23
CUBLAS version: 9.1

Results of numerical experiments are summarized in the following table and figures 1, 2, 3. Figure 4 is a zoomed out version of fig. 3.

The algorithm run for given parameters p and d, and γ was set to $\gamma = 0.9$. Here p is any non-negative integer (not only prime), the program computes the set of parameters $S = \{k_1, k_2 \cdots k_d\}$ such that $\epsilon(S) = \frac{1}{d}\left|\sum_{i=1}^{d} \cos \frac{2\pi k_i}{p}\right| \leq \gamma$. The table also contains a value $\epsilon = \epsilon(S)$ that was computed for the resulting S.

Table 1: Results of numerical experiments. Columns p and d contain input parameters. Column S contains d numbers — the sequence of parameters that was found. Column ϵ contains the value of bias for the sequence S.

p	d	S	ϵ
8	5	1, 2, 3, 0, 4	0.20000
16	7	1, 2, 7, 3, 4, 6, 5	0.14286
16	9	1, 2, 7, 3, 4, 6, 5, 0, 8	0.11111
20	8	1, 4, 5, 3, 6, 9, 8, 2	0.22613
21	8	1, 9, 8, 3, 4, 10, 7, 6	0.21837
22	9	1, 4, 5, 10, 2, 6, 3, 7, 8	0.20458
23	9	1, 10, 5, 11, 8, 4, 3, 9, 2	0.18515
24	9	1, 4, 10, 3, 6, 9, 2, 5, 8	0.22222
25	9	3, 12, 10, 2, 4, 8, 11, 6, 7	0.23534
30	10	1, 4, 8, 12, 3, 9, 7, 6, 5, 13	0.24962
32	25	1, 6, 5, 8, 14, 9, 7, 2, 12, 10, 15, 3, 11, 4, 13, 0, 16, 5, 4, 13, 2, 6, 10, 9, 11	0.10630
64	30	1, 6, 22, 5, 13, 19, 21, 30, 11, 8, 9, 14, 12, 18, 27, 23, 3, 7, 20, 24, 4, 16, 28, 17, 25, 31, 29, 2, 10, 26	0.066026
128	35	1, 44, 30, 9, 6, 50, 31, 32, 41, 14, 28, 24, 35, 7, 18, 46, 49, 21, 40, 52, 22, 5, 19, 12, 45, 48, 60, 39, 63, 36, 43, 20, 13, 10, 3	0.15562
256	40	2, 17, 97, 77, 68, 127, 90, 121, 124, 19, 67, 23, 116, 16, 112, 57, 98, 41, 29, 126, 103, 110, 69, 94, 47, 26, 31, 56, 33, 22, 40, 84, 51, 18, 122, 7, 99, 37, 105, 107	0.17884
384	43	1, 6, 107, 142, 158, 37, 108, 55, 148, 75, 83, 179, 119, 100, 33, 89, 185, 155, 164, 60, 174, 106, 78, 145, 178, 82, 80, 95, 91, 45, 171, 110, 57, 47, 124, 38, 133, 112, 180, 46, 30, 29, 19	0.17676

Table 1: (continued)

p	d	S	ϵ
512	45	1, 172, 186, 226, 162, 197, 245, 239, 45, 37, 129, 217, 111, 7, 146, 224, 219, 87, 220, 190, 130, 24, 127, 208, 185, 101, 57, 150, 125, 84, 74, 193, 93, 114, 153, 62, 144, 3, 243, 115, 206, 122, 42, 152, 136	0.18060
640	47	105, 10, 11, 259, 271, 223, 304, 15, 115, 81, 64, 252, 68, 25, 282, 149, 195, 136, 80, 17, 33, 198, 13, 238, 86, 262, 27, 131, 112, 213, 224, 142, 317, 284, 283, 95, 312, 163, 172, 21, 157, 22, 296, 244, 187, 287, 12	0.18123
768	47	1, 10, 62, 232, 372, 107, 382, 296, 253, 225, 122, 206, 184, 214, 39, 231, 6, 52, 187, 262, 127, 121, 155, 181, 157, 325, 21, 75, 117, 333, 351, 22, 245, 101, 368, 301, 260, 198, 73, 100, 115, 203, 63, 364, 348, 270, 152	0.18809
1024	50	1, 465, 6, 200, 428, 481, 14, 356, 191, 149, 102, 510, 281, 70, 64, 290, 47, 499, 216, 283, 123, 177, 468, 511, 254, 399, 25, 463, 167, 304, 124, 161, 164, 82, 497, 72, 287, 249, 51, 29, 353, 346, 309, 245, 75, 50, 458, 62, 122, 153	0.18722
1280	52	1, 6, 208, 265, 364, 627, 42, 245, 481, 365, 87, 311, 519, 297, 64, 523, 412, 421, 169, 594, 468, 79, 30, 58, 456, 119, 439, 303, 163, 438, 298, 337, 491, 145, 55, 344, 283, 544, 534, 122, 629, 314, 536, 348, 207, 482, 205, 134, 95, 553, 232, 186	0.18946

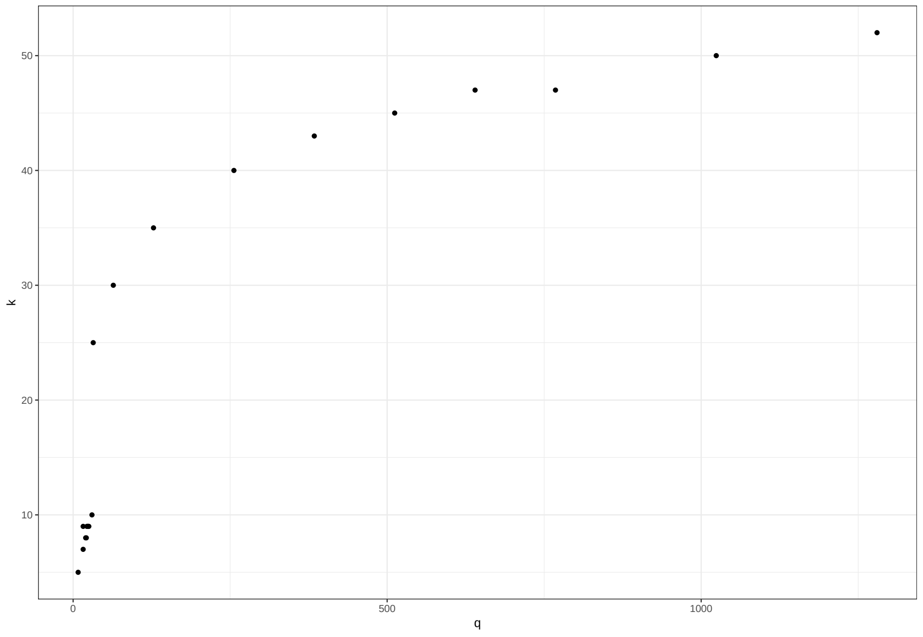

Fig. 1. Values of d that were used for different p

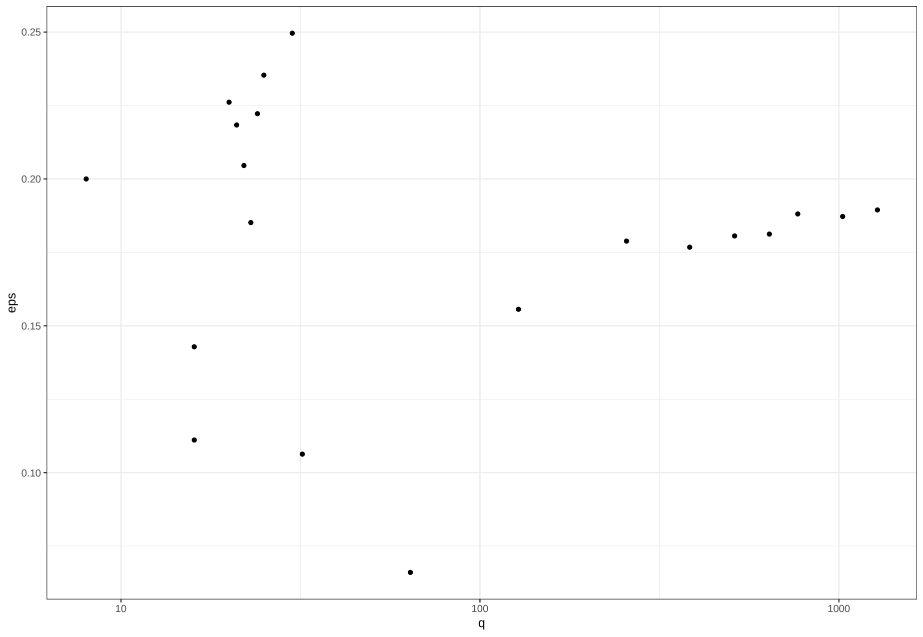

Fig. 2. Computed ϵ values

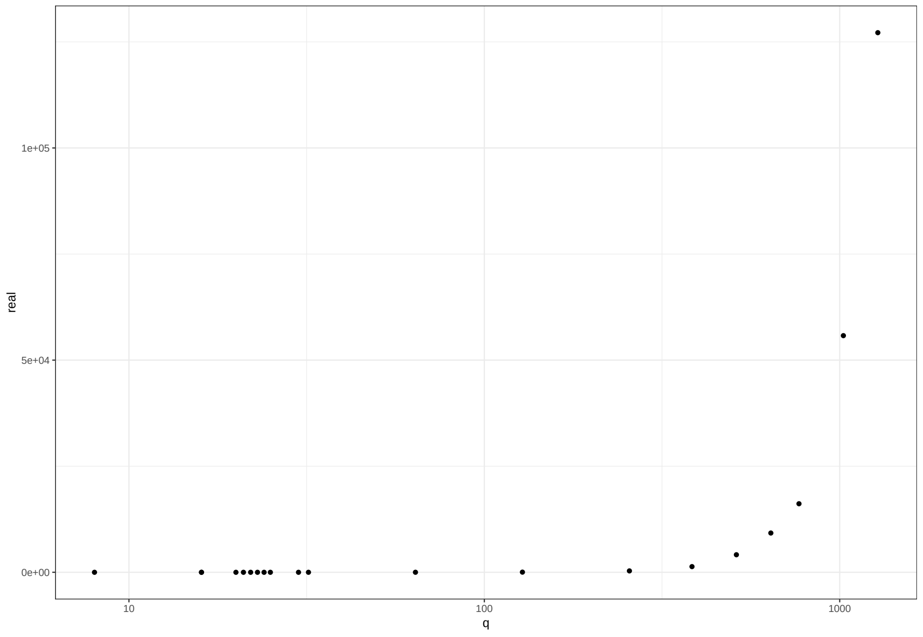

Fig. 3. Computation time

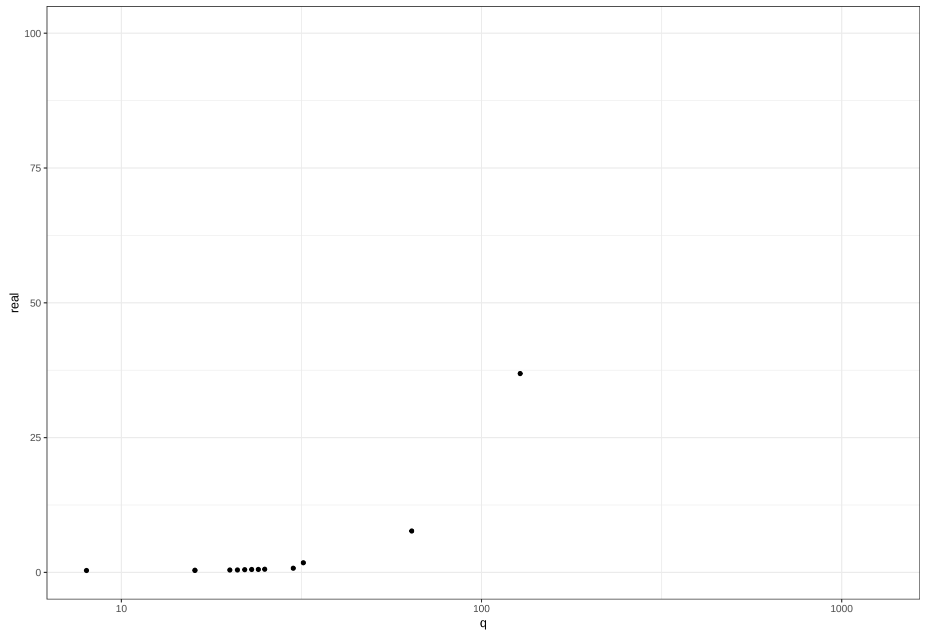

Fig. 4. Computation time for small values of d

ISBN 978-952-12-3840-6
ISSN 1797-8831