

A Quantum Algorithm for Finding k -Minima

Kohei Miyamoto,* Masakazu Iwamura,† and Koichi Kise‡

*Department of Computer Science and Intelligent Systems,
Graduate School of Engineering, Osaka Prefecture University*

(Dated: July 9, 2019)

We propose a new *finding k -minima* algorithm and prove that its query complexity is $\mathcal{O}(\sqrt{kN})$, where N is the number of data indices. Though the complexity is equivalent to that of an existing method, the proposed is simpler. The main idea of the proposed algorithm is to search a good threshold that is near the k -th smallest data. Then, by using the generalization of amplitude amplification, all k data are found out of order and the query complexity is $\mathcal{O}(\sqrt{kN})$. This generalization of amplitude amplification is also not well discussed and we briefly prove the query complexity. Our algorithm can be directly adapted to distance-related problems like k -nearest neighbor search and clustering and classification. There are few quantum algorithms that return multiple answers and they are not well discussed.

I. INTRODUCTION

We propose an $\mathcal{O}(\sqrt{kN})$ quantum algorithm for finding k -minima, where N is the number of data indices. Our algorithm finds the k smallest from N data indices. Dürr and Høyer have originally proposed an $\mathcal{O}(\sqrt{N})$ quantum algorithm for finding one minimum [1]. Then, Dürr, et al. have proposed an $\mathcal{O}(\sqrt{kN})$ quantum algorithm for finding k -minima [2]. Though the proposed algorithm has the same query complexity as the one proposed by Dürr and Heiligman [2], the proposed is simpler. One reason is that they take into account types of data. They want to use this algorithm as a part of graph algorithm, therefore their algorithm is not purely designed as finding k -minima algorithm.

The proposed algorithm is based on following three algorithms: finding minimum (FM) algorithm [1], quantum counting (QC) algorithm [3, 4] and amplitude amplification (AA) [4, 5]. FM algorithm and QC algorithm are used to find a good threshold index, and AA is used to find all k indices whose values are less than the value of the threshold index.

In addition, this paper contributes to the following two. First, we explicitly distinguish gate complexity and query complexity by defining new symbols. Second, we re-formulate FM algorithm and *finding k -minima* algorithm following the manner of AA. Therefore, all of them can be compared more easily and clearly. AA is a quantum database search algorithm and is a generalization of Grover’s algorithm [6]. From N indices, AA searches one of k

indices that satisfy some certain condition with the query complexity of $\mathcal{O}(\sqrt{N/k})$.

Many quantum algorithms do not directly return multiple answers because the measurement of quantum states collapses the state of superposition. Therefore, many trials are required to obtain all the results. This is the disadvantage of quantum algorithms and such trials sometimes increase linearly for the number of results. Hence, $\mathcal{O}(k)$ trials are required for k results. However, our algorithm solves the problem that returns k results with $\mathcal{O}(\sqrt{k})$. We use a generalization of AA that searches all k indices from N indices. We call this algorithm *searching all marked k -indices* algorithm. This problem is solved in $\mathcal{O}(\sqrt{kN})$ query complexity [7–11].

Finding k -minima algorithm can be applied to k -nearest neighbor search like [12] and other quantum machine learning methods such as clustering and classification [11, 13–15].

II. COMPLEXITY MEASURES

Two kinds of complexity measures are used in quantum algorithms. One is *quantum gate complexity* that is based on the number of quantum gates to solve a problem. The other is *query complexity* that is based on the number of queries to solve a problem. For these two complexity measures, the same mathematical expression \mathcal{O} is used. However, in this paper, we explicitly distinguish them by using \mathcal{O}_g for the quantum gate complexity and \mathcal{O}_q for the query complexity. For example, making desired quantum states of a d dimensional vector in $n = \log d$ [qubit] requires $\mathcal{O}_g(n) = \mathcal{O}_g(\log d)$ [14, 16–18], quantum fourier transform of n [qubit] is $\mathcal{O}_g(n \log N)$ [19] and AA’s complexity is $\mathcal{O}_q(\sqrt{N/k})$ [4, 5].

* miyamotokohei@protonmail.com

† masa@cs.osakafu-u.ac.jp

‡ kise@cs.osakafu-u.ac.jp

A. Quantum Gate Complexity

Quantum algorithms based on quantum gates solve the problems by arranging the quantum gates. The Hadamard gate and controlled-NOT gate are the most basic quantum gates. It is known that any kinds of unitary transformation can be approximated by these two gates. This means that the quantum gate complexity can be measured by the number of these two gates.

B. Query Complexity

While the quantum gate complexity is easy to understand, most of the quantum algorithms are evaluated based on the query complexity. This is because many quantum algorithms use oracles.

A query is input to the oracle and query complexity is defined by the number of queries to get an answer. In other words, query complexity measures how many times oracles are called. Oracle is usually treated as a single quantum gate. Therefore, oracles are sometimes called black-box and query model is called black-box model.

To analyze the query complexity, some kinds of methods are proposed. For example, polynomial methods [20, 21] and adversary methods [22, 23]. Polynomial methods generalize a boolean function $f(x) : \{0, 1\}^n \rightarrow \{0, 1\}$ to a polynomial function $p(x) : \mathbb{R}^N \rightarrow \mathbb{R}$. By analyzing the degree of polynomial function $p(x)$, lower bounds can be known. Strong direct product theorem is an important theorem to measure the complexity in both polynomial and adversary methods [8, 9, 24].

III. PRELIMINARIES

In this section, we define mathematical symbols and complexity while presenting how oracle f is defined in AA. AA is one of the most important quantum algorithms and we present FA algorithm following the manner of AA.

All kinds of data are treated as a binary representation in a digital computer. However, quantum computer not only treats binary representation with superposition but also treats analog representation by superposition of quantum bits [25]. Digital representation can have multiple numbers in one time by superposition and analog representation can have one vector that the square of the number is represented by a probability. We will not describe more detail about the representation in this paper.

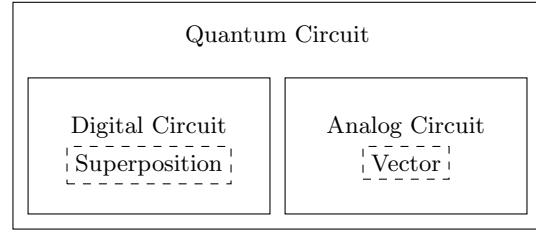


FIG. 1. Quantum circuit is composed of a digital circuit and an analog circuit. Digital circuit uses superposition of binary representation. Analog circuit uses the vector of the number such that the square of the number becomes a probability.

An output that we can measure is binary representation in a quantum computer. Therefore, the last output of the algorithm is binary representation. However, we can use either way in the middle of the quantum circuit. We call circuit that uses binary representation as a digital circuit and algorithm that use analog representation as an analog circuit. This means that the quantum circuit contains a digital circuit and an analog circuit like Figure 1. We have to mention that the input and output of this digital circuit can be superposition in this paper. Many quantum circuits use both a digital circuit and an analog circuit. Moreover, quantum bits sometimes go back and forth between digital representation and analog representation.

We use AA as a base of our explanation. AA searches one index that satisfies a condition. The condition is given by oracle. To use AA, we have to define oracle that fits a problem.

Let D be a set of indices and each of which is tied to value. $|D| = N$ and $x \in D$ is represented by binary.

$$x = \{0, 1\}^{\log N}$$

For simplicity,

$$N = |D| = 2^n, \text{ where } n \in \mathcal{N}$$

We want to find index $x \in D$ that satisfies a condition. For example, index x that has less value than the threshold value.

Let $f(x)$ be a boolean function such that

$$f(x) : \{0, 1\}^{\log N} \rightarrow \{0, 1\}.$$

If $f(x) = 1$, x satisfies a condition f and if $f(x) = 0$, x does not satisfy a condition f .

$$f(x) = \begin{cases} 1, & \text{if } x \text{ satisfies condition } f, \\ 0, & \text{if } x \text{ does not satisfy condition } f. \end{cases}$$

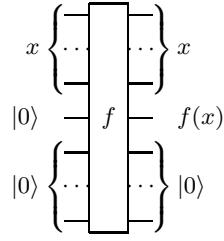


FIG. 2. A quantum circuit that converts index x to $f(x)$. x is a binary index and $f(x)$ is a binary. The bottom input $|0\rangle$ and output $|0\rangle$ is a workspace for quantum computing. Most of the oracles use workspaces like this, however, it is often omitted in a quantum circuit. The output other than the desired output must be the same condition as the input condition. In this case, only the output $f(x)$ is different from the input and others are the same. This is because the measurement of non-desired output should not influence on the desired output. To avoid this, we have to add a gate to inverse the process. However, we omit such gate for simplicity in this paper.

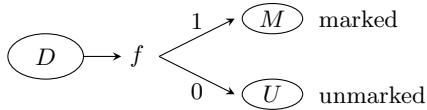


FIG. 3. D is divided into M and U by f

In some paper, $x \in D$ is called marked index if $f(x) = 1$, and $x \in D$ is called unmarked index if $f(x) = 0$.

This kind of boolean function is called oracle and thought as a single quantum gate. If we express f in quantum circuit, it becomes Figure 2. An input x is a binary representation and x is superposed and input to the quantum circuit f in AA.

Thinking f from another point of view, f divide D into two sets. Let M and U be sets of indices such that $f(x) = 1$ (marked) and $f(x) = 0$ (unmarked) respectively.

$$\begin{aligned} M &= \{x \mid x \in D, f(x) = 1\} \\ U &= \{x \mid x \in D, f(x) = 0\} \end{aligned}$$

$M \cup U = D$ and $M \cap U = \emptyset$. D is divided into M and U like Figure 3.

Original AA finds one index $x \in D$ such that $f(x) = 1$. In other words, find x from M . We have to mention that AA does not find all $x \in M$. AA can find only one index in M like figure III.

When we want to find all M from D , we have to update M each time $x \in M$ found. Let $|M| = k$ and all $x \in M$ are found the order of $\{x_1, \dots, x_k\}$. In this case, AA is called k times and x_i means that

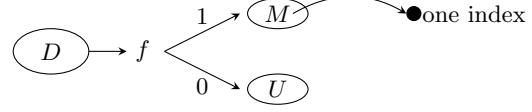


FIG. 4. Overview of AA. AA finds one index from M and M is a subset of D . $M = \{x \mid x \in D, f(x) = 1\}$.

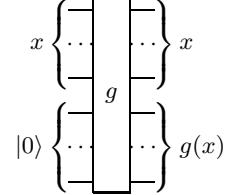


FIG. 5. A quantum circuit G that converts index x to value $g(x)$. x is a binary index. The input index and output index must be the same condition. This is because measurement of index should not influence on the output value $g(x)$.

$x_i \in M$ is found at i times calling of AA. In general, we cannot know the order of finding before applying. In each step, M is updated and one index that found is removed. Let M_i be the i iteration of M and $M_1 = M$.

$$\begin{aligned} M_i &= M_{i-1} \setminus x_{i-1} \\ &= M_1 \setminus \{x_1, \dots, x_{i-1}\} \\ &= \{x_i, \dots, x_k\} \\ |M_i| &= k - i - 1 \end{aligned}$$

See appendix for more detail about finding all marked k -indices.

Finally, we present how to design f by showing one example. AA needs oracle f however it is not given in algorithm and only input and output are assigned. For example, if we want to find one of the indices that has less value than the value of threshold index t .

Let f_t be a required oracle to solve this problem on AA and let g be a function that returns a value by which index $x \in D$. In other words $g(x)$ returns a value that is combined to index $x \in D$. g is also oracle and the quantum circuit is like Figure 5. The gate complexity of this oracle g is $\mathcal{O}_g(\log d)$ as we described in Section II. d is a dimension of $g(x)$. In other words, d is a number of quantum bit. For more detail, see [14, 16–18].

The input x and output $g(x)$ of quantum circuit g is also binary representation and can be superposed. Therefore output can be superposed if input index x is superposed. Figure 6 is an example of how g is working. Then, the oracle f is expressed like this.

| Indices | Values |
|----------|-------------|
| x | $g(x)$ |
| 0 | $g(0) = 34$ |
| 1 | $g(1) = 24$ |
| 2 | $g(2) = 89$ |
| 3 | $g(3) = 39$ |
| 4 | $g(4) = 26$ |
| 5 | $g(5) = 52$ |
| 6 | $g(6) = 95$ |
| \vdots | \vdots |

FIG. 6. Example of the input x and output $g(x)$ of oracle g .

| $x \in D$ | $g(x)$ | $g(t)$ | $f_t(x)$ |
|-----------|-------------|----------------------------------|----------|
| 0 | $g(0) = 97$ | $\geq 55 \rightarrow f_t(0) = 0$ | |
| 1 | $g(1) = 82$ | $\geq 55 \rightarrow f_t(1) = 0$ | |
| 2 | $g(2) = 40$ | $< 55 \rightarrow f_t(2) = 1$ | |
| 3 | $g(3) = 68$ | $\geq 55 \rightarrow f_t(3) = 0$ | |
| 4 | $g(4) = 55$ | $= 55 \rightarrow f_t(4) = 0$ | |
| 5 | $g(5) = 53$ | $< 55 \rightarrow f_t(5) = 1$ | |
| 6 | $g(6) = 96$ | $\geq 55 \rightarrow f_t(6) = 0$ | |
| \vdots | \vdots | \vdots | \vdots |

FIG. 7. Example of indices, value and f_t . When threshold t is 4. Superposed indices x are converted to values $g(x)$ and compared by the threshold value $g(t)$.

$$f_t(x) = \begin{cases} 1, & \text{if } g(x) < g(t), \\ 0, & \text{if } g(x) \geq g(t). \end{cases} \quad (1)$$

Here, t is a threshold index in D . We use this oracle f_t in section IV for FM algorithm. Figure 7 shows an example of how f is working.

Let CMP be a function that compare two values and return 1 or 0.

$$\text{CMP}(v_1, v_2) = \begin{cases} 1, & \text{if } v_1 < v_2, \\ 0, & \text{if } v_1 \geq v_2. \end{cases}$$

The computational complexity of oracle CMP is $\mathcal{O}_g(\log d)$, because the input v_1 and v_2 are binary representation.

From the above, a quantum circuit of f_t is expressed as Figure 8. In this circuit, the upper input x and the middle output $f_t(x)$ are used in AA and other inputs and outputs are not used. However, we

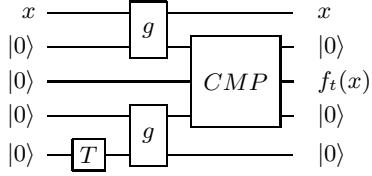


FIG. 8. Quantum circuit of oracle f_t . The index t is created by the gate T in the circuit. In this example, CMP gate is symmetry. The center input is for output $f_t(x)$ and above and below inputs are for two values. If above input is less than the input of below, CMP returns 0. Otherwise, returns 1.

have to keep unchanged of inputs and outputs so as not to influence the states of $f_t(x)$.

IV. FINDING MINIMUM ALGORITHM

We begin with presenting an overview of FM algorithm because it is a simple application of AA and helps understand the succeeding algorithms. An example of the procedure of the algorithm is shown in Figure 9. This algorithm finds the minimum from D with the complexity of $\mathcal{O}_q(\sqrt{N})$ by iteratively updating threshold index t by oracle f_t . Oracle $f_t(x)$ is a function that marks index x such that $g(x) < g(t)$. That is,

$$f_t(x) = \begin{cases} 1, & \text{if } g(x) < g(t), \\ 0, & \text{if } g(x) \geq g(t). \end{cases} \quad (2)$$

The oracle marks the indices that have smaller values than the value of threshold t . Setting a marked index as the new threshold, the value of threshold decreases. Eventually, we obtain the index that has the minimum value.

In summary, FM algorithm is given as follows.

1. Select threshold index t from D uniformly at random.
2. Repeat the following process more than $22.5\sqrt{N} + 1.4\log^2 N$ times.
 - (a) Find index x such that $f_t(x) = 1$.
 - (b) Set the found index x as the threshold index t .
3. Return t as the index that has the minimum value in D .

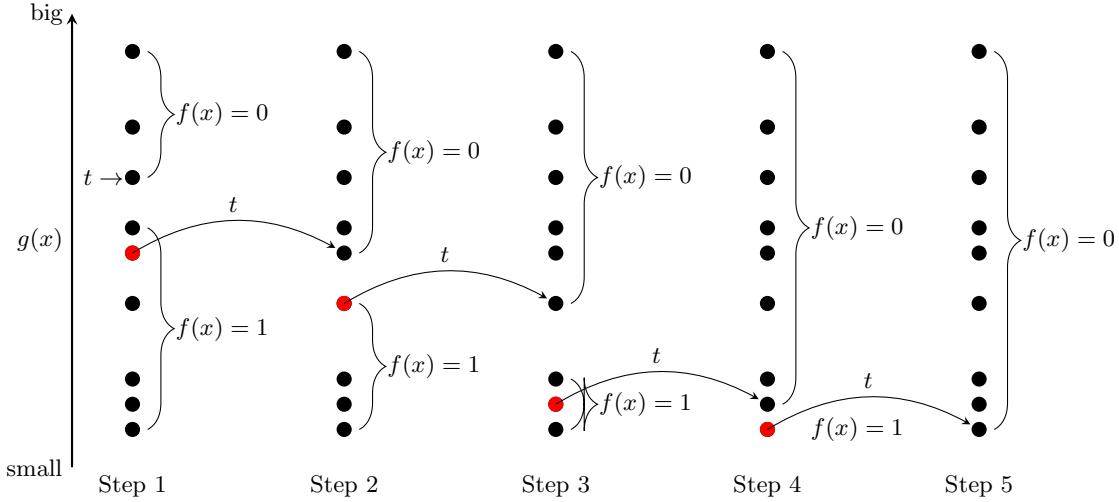


FIG. 9. The updating process of threshold t in FM algorithm. The vertical axis represents the value of indices. When the first threshold t is selected, all points are divided into two. One is $f(x) = 0$ and the other is $f(x) = 1$. AA randomly selects one of the points such that $f(x) = 1$. The selected point, drawn in red, is used as a new threshold index t in the next step. The number of points satisfying $f(x) = 1$ decreases as the step goes. Finally (at the Step 5 in this example), the algorithm ends as no points satisfy $f(x) = 1$.

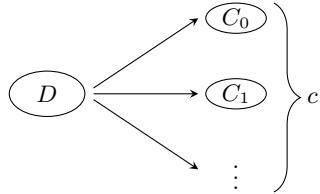


FIG. 10. D is divided into C_i by types.

V. CONVENTIONAL FINDING k -MINIMA ALGORITHM

Dürr and Høyer have proposed an $\mathcal{O}_q(\sqrt{kN})$ algorithm that finds the k smallest indices of *different types* from D [2]. As the algorithm considers *types*, it is complex. Hence, we present it in an easier-to-understand way.

As it treats types, we consider the following conditions (see also Figure 10).

- Each index has a type.
- Let C_i be a set of indices that have type i .
- Let c be the number of types.

Dürr and Høyer's algorithm finds the indices of the c smallest values each of which is the minimum in each type (see Figure 11). This means that so as to use the conventional algorithm for the general *finding*

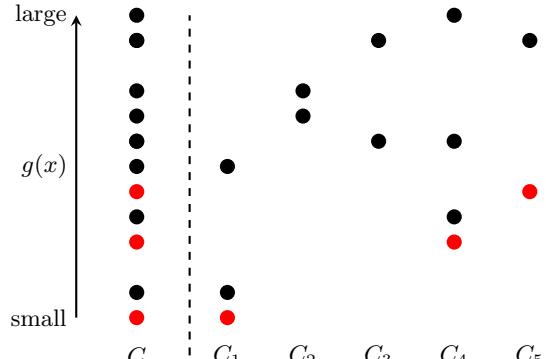


FIG. 11. Overview of Dürr and Høyer's *finding k -minima* algorithm with different types in the case of $c = 5$ and $k = 3$. C is the set of all values of D . C_i is the set of values that belong to type i . Each C_i can contain at most one minimum. This is equal to the case that all of k indices are of different types.

k-minima problem, all indices must be of different types like Figure 12, which corresponds to $c = N = |D|$. Hereafter, we assume all indices are of different types.

Intuitive explanation of the algorithm is that multiple thresholds are maintained while a single threshold is maintained in FM algorithm. Let T be a set of k thresholds and let $f_T(x)$ be an oracle function such that

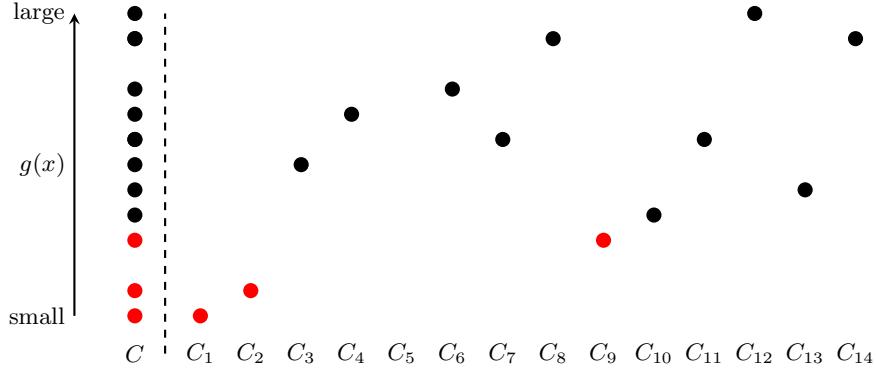


FIG. 12. This is an example case that each C_i contains only one index. The total number of types is the same as that of indices. Therefore, one index exclusively corresponds to one type (one-to-one correspondence).

$$f_T(x) = \begin{cases} 1, & \text{if } g(x) < g(t) \text{ for some } t \in T, \\ 0, & \text{if } g(x) \geq g(t) \text{ for some } t \in T. \end{cases} \quad (3)$$

Similar to Eq. (2) of FM algorithm, Eq. (3) is regarded as AA for *some* threshold t . However, the meaning of *some* is not clearly mentioned in [2]. Though the algorithm does not work well in the worst case, which selects the threshold index t that minimizes $g(t)$, it works in the following cases.

1. t is randomly chosen from T .
2. t is selected so as to maximize $g(t)$.

In the case of 2, which is the best case, such t is obtained by *finding maximum* algorithm [26]. Hence, its computational burden is $\mathcal{O}(\sqrt{k})$. It can be ignored, as it is small enough compared to the complexity of the whole algorithm (i.e., $\mathcal{O}(\sqrt{kN})$).

In addition to that, we have to keep the elements of T without duplication. So as to do that, the algorithm requires duplication check or removing T from search indices set

$$M = \{x \mid f_T(x) = 1, x \in D\}. \quad (4)$$

We assume that it removes T from M because our proposed method also removes already found indices from M .

By ignoring types of data, the *finding k-minima* algorithm based on Dürr and Høyer's algorithm is given as follows.

1. Initialize set T as randomly chosen k indices from D .
2. Repeat the following forever.

- (a) Randomly select a threshold index t from T .
- (b) Find index x such that $f_T(x) = 1$.
- (c) Find t_{\max} such that $t_{\max} = \operatorname{argmax}_{t \in T} g(t)$ [27].
- (d) Replace threshold index t_{\max} with x .

In this algorithm, T is updated in a step-by-step manner and each updating step replaces the index that has the maximum value in T with the found index by AA. This is a kind of a greedy algorithm.

VI. PROPOSED FINDING k -MINIMA ALGORITHM

We propose a new *finding k-minima* algorithm with the complexity of $\mathcal{O}_q(\sqrt{kN})$. Our idea is to search a good threshold (the first phase) and use it for *finding all marked k-indices* algorithm (the second phase). We begin with presenting the second phase. In the second phase, all k' indices, where $k' \geq k$, are found. Suppose that threshold index $t_{k'}$ satisfy

$$M = \{x \mid g(x) < g(t_{k'}), x \in D\}, \quad (5)$$

$$|M| = k'. \quad (6)$$

In the first phase, in order to find the threshold $t_{k'}$, we use FM algorithm and QC algorithm. As shown in Sec. IV, in the process of FM algorithm, $\mathcal{O}(\sqrt{N})$ threshold indices are found in the descending order of values. Therefore, it is easy to find $t_{k'}$ from them by a binary search with QC algorithm.

We present more detail about this binary search with QC algorithm. Let us define the following.

- Let t_i^{FM} be the threshold index that is found in the i -th step of FM algorithm.
- Let T_{FM} be a set of thresholds that are found in the process of FM algorithm, which is given by

$$T_{FM} = \{t_1^{FM}, t_2^{FM}, \dots\}$$

- Let M_i^{FM} be a set of marked indices of the i -th step in FM algorithm.
- Let $h(t)$ be a function that maps index t to the number of indices whose values are less than the value of threshold t in D , which is counted by QC algorithm. That is,

$$h(t) = |M(t)|, \quad (7)$$

where

$$M(t) = \{x \mid g(x) < g(t), x \in D\}. \quad (8)$$

The goal of the binary search is to find i such that $h(t_{i+1}^{FM}) \leq k < h(t_i^{FM})$. Once such i is found, $h(t_i^{FM})$ is used as k' . Fortunately, $t_{k'}$ exists in the last k indices of all found thresholds. Hence, we do not have to search all found $\mathcal{O}(\sqrt{N})$ indices but the last k indices. As QC algorithm requires $\mathcal{O}(\sqrt{N})$ query complexity for N indices [3], a binary search for k indices requires $\mathcal{O}(\log k)$ comparison. As QC algorithm has to run in each step of the binary search, threshold $t_{k'}$ can be found in $\mathcal{O}(\sqrt{N} \log k)$.

In summary, our threshold searching algorithm is shown below.

1. Apply FM algorithm and save the indices of the lastly found k thresholds.
2. Apply the binary search on the k indices of thresholds.

Once we find such a threshold, we can find all marked k' indices by applying *searching all marked k -indices* algorithm. This algorithm searches all elements in the set $\{x \mid x \in D, f(x) = 1\}$. For simplicity, we assume that $\mathcal{O}(k') = \mathcal{O}(k)$. Let T be a set of already found indices in the step of *searching all marked k indices algorithm* and let $f'_{t_k}(x)$ be an oracle such that

$$f'_{t_k}(x) = \begin{cases} 1, & \text{if } g(x) < g(t_k) \text{ and } x \notin T, \\ 0, & \text{otherwise.} \end{cases} \quad (9)$$

Then, searching all marked k -indices in M can be done in $\mathcal{O}(\sqrt{kN})$.

The whole algorithm of the proposed method is below.

1. Apply FM algorithm to D and save the last k indices of FM algorithm step.
2. Search threshold index $t_{k'}$ by a binary search on k indices. The comparison key is $|M^{FM}|$ that is derived by QC algorithm.
3. Apply *finding all marked k -indices* algorithm with threshold $t_{k'}$.

The total query complexity is given as

$$\mathcal{O}(\sqrt{N} \log k) + \mathcal{O}(\sqrt{kN}) = \mathcal{O}(\sqrt{kN}). \quad (10)$$

VII. CONCLUSION

In this paper, we proposed a new finding k -minima algorithm and derived its query complexity. Our algorithm is easier to understand and more elegant than Dürr's algorithm [2]. Finding k -minima algorithm can be applied to many kinds of algorithms or applications. For example, k -nearest neighbor search, k -nearest neighbor clustering and classification.

- [1] Christoph Dürr and Peter Høyer. A quantum algorithm for finding the minimum. *arXiv preprint quant-ph/9607014*, 1996.
- [2] Christoph Dürr, Mark Heiligman, Peter Høyer, and Mehdi Mhalla. Quantum query complexity of some graph problems. *SIAM Journal on Computing*, 35(6):1310–1328, 2006.
- [3] Gilles Brassard, Peter Høyer, and Alain Tapp. Quantum counting. In *International Colloquium on Automata, Languages, and Programming*, pages 820–831. Springer, 1998.
- [4] Gilles Brassard, Peter Høyer, Michele Mosca, and Alain Tapp. Quantum amplitude amplification and estimation. In Jr. Samuel J. Lomonaco, editor, *Quantum Computation and Quantum Information*, volume 305, pages 53–74. American Mathematical Society, 2002.

- [5] Michel Boyer, Gilles Brassard, Peter Høyer, and Alain Tapp. Tight bounds on quantum searching. *Fortschritte der Physik*, 46(4-5):493–505, 1998.
- [6] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 212–219, 1996.
- [7] Andris Ambainis. Quantum search algorithms. *ACM SIGACT News*, 35(2):22–35, 2004.
- [8] Andris Ambainis. A new quantum lower bound method, with an application to strong direct product theorem for quantum search. *arXiv preprint quant-ph/0508200*, 2005.
- [9] Hartmut Klauck, Robert Špalek, and Ronald De Wolf. Quantum and classical strong direct product theorems and optimal time-space tradeoffs. *SIAM Journal on Computing*, 36(5):1472–1493, 2007.
- [10] Sebastian Dörn and Thomas Thierauf. A note on the search for k elements via quantum walk. *Information Processing Letters*, 110(22):975–978, 2010.
- [11] Esma Aïmeur, Gilles Brassard, and Sébastien Gambs. Quantum speed-up for unsupervised learning. *Machine Learning*, 90(2):261–287, 2013.
- [12] Nathan Wiebe, Ashish Kapoor, and Krysta Svore. Quantum algorithms for nearest-neighbor methods for supervised and unsupervised learning. *arXiv preprint arXiv:1401.2142*, 2014.
- [13] Esma Aïmeur, Gilles Brassard, and Sébastien Gambs. Quantum clustering algorithms. In *Proceedings of the 24th international conference on machine learning*, pages 1–8, 2007.
- [14] Seth Lloyd, Masoud Mohseni, and Patrick Rebentrost. Quantum algorithms for supervised and unsupervised machine learning. *arXiv preprint arXiv:1307.0411*, 2013.
- [15] Maria Schuld, Ilya Sinayskiy, and Francesco Petruccione. An introduction to quantum machine learning. *Contemporary Physics*, 56(2):172–185, 2015.
- [16] Lov Grover and Terry Rudolph. Creating superpositions that correspond to efficiently integrable probability distributions. *arXiv preprint quant-ph/0208112*, 2002.
- [17] Phillip Kaye and Michele Mosca. Quantum networks for generating arbitrary quantum states. *arXiv preprint quant-ph/0407102*, 2004.
- [18] Andrei N Soklakov and Rüdiger Schack. Efficient state preparation for a register of quantum bits. *Physical Review A*, 73(1):012307, 2006.
- [19] Lisa Hales and Sean Hallgren. An improved quantum fourier transform algorithm and applications. In *Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on*, pages 515–525, 2000.
- [20] Robert Beals, Harry Buhrman, Richard Cleve, Michele Mosca, and Ronald De Wolf. Quantum lower bounds by polynomials. *Journal of the ACM*, 48(4):778–797, 2001.
- [21] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Pr., 2001.
- [22] Andris Ambainis. Quantum lower bounds by quantum arguments. *Journal of Computer and System Sciences*, 64(4):750–767, 2002.
- [23] Robert Špalek and Mario Szegedy. All quantum adversary methods are equivalent. *arXiv preprint quant-ph/0409116*, 2004.
- [24] Andris Ambainis, Robert Špalek, and Ronald de Wolf. A new quantum lower bound method, with applications to direct product theorems and time-space tradeoffs. *Algorithmica*, 55(3):422–461, 2009.
- [25] Kosuke Mitarai, Masahiro Kitagawa, and Keisuke Fujii. Quantum analog-digital conversion. *Physical Review A*, 99(1):012301, 2019.
- [26] Ashish Ahuja and Sanjiv Kapoor. A quantum algorithm for finding the maximum. *arXiv preprint quant-ph/9911082*, 1999.
- [27] Though this process is not described in [2], we add this because we think it is required for conversion.

Appendix A: Complexity of Searching All Marked k -Indices Algorithm

The problem to search all marked k -indices from D and its query complexity are briefly discussed in [7–11]. Here, we will give the complexity of the problem in an easy-to-understand way.

Let D and M be sets of data indices where $|D| = N$ and marked indices where $|M| = k$, respectively. Since AA searches one of the k indices from D in $\mathcal{O}_q(\sqrt{N/k})$ query complexity [4–6], a way to search all marked k -indices is to repeat the following process until all k -marked indices are found.

1. Find one index m from M by using AA.
2. Remove m from M .

By updating $f(x)$ to $f'(x)$ to remove already found indices.

$$f'(x) = \begin{cases} 1, & \text{if } f(x) = 1 \text{ and } x \notin T, \\ 0, & \text{if } f(x) = 0 \text{ or } x \in T. \end{cases} \quad (\text{A1})$$

This $f'(x)$ needs additional quantum circuit on $f(x)$. However, such quantum circuit is not complex, because we can easily make superposition of the already found indices T in $\mathcal{O}_g(\log N)$ [14, 16–18]. Figure 13 shows an example of the quantum circuit f' .

The following algorithm finds all k -indices.

1. Repeat the following process $\mathcal{O}(k)$ times.
 - (a) Search one index $m \in M$ from D by using AA and $f'(x)$.
 - (b) Add a found index m to T .

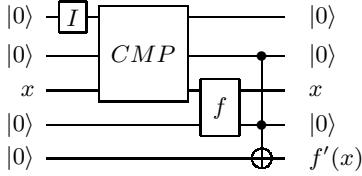


FIG. 13. An example of quantum circuit f' . The gate T creates the superposition of already found indices in T . The gate f is an original oracle of AA that finds one index from D . We can use Fredkin gate instead of Toffoli gate because we use this gate as AND gate [21]. As we said before, we omit inverting gates here. Strictly speaking, the gate CMP do not guarantee that the input x and output are the equivalent superposition. To avoid this problem we have to add an inverse gate after CMP gate. However, we assume that the inverting gate is already built in the CMP gate for simplicity.

- (c) Remove a found index m from M as given as

$$M = M \setminus m.$$

In the t -th iteration of this algorithm, as $|M| = k-t$, the query complexity to search a marked index is $\mathcal{O}_q(\sqrt{\frac{N}{k-t}})$. Therefore, the total query complexity is given as

$$\mathcal{O}_q \left(\sqrt{\frac{N}{k}} + \sqrt{\frac{N}{k-1}} + \dots + \sqrt{N} \right) \quad (\text{A2})$$

$$= \mathcal{O}_q(\sqrt{kN}), \quad (\text{A3})$$

because

$$\sum_{t=1}^k \sqrt{\frac{1}{t}} < 1 + \int_1^k \sqrt{\frac{1}{t}} dt = 2\sqrt{k} - 1. \quad (\text{A4})$$