Faster quantum concentration via Grover’s search

Cem M. Unsal\textsuperscript{a} and A. Yavuz Oruc\textsuperscript{b}

\textsuperscript{a}Department of Mathematics, University of Maryland, College Park, MD, USA; \textsuperscript{b}Department of Electrical and Computer Engineering, University of Maryland, College Park, MD, USA

ABSTRACT

One of the most important challenges in information networks is to gather data from a larger set of nodes to a smaller set of nodes. This can be done via the use of a concentrator architecture in the connection topology. This paper is a proof-of-concept that demonstrates a quantum-based controller in large interconnection networks can asymptotically perform this task faster. We specifically present quantum algorithms for routing concentration assignments on full capacity fat-and-slim concentrators, bounded fat-and-slim concentrators, and regular fat-and-slim concentrators. Classically, the concentration assignment takes \( O(n) \) time on all these concentrators, where \( n \) is the number of inputs. Powered by Grover’s quantum search algorithm, our algorithms take \( O\left(\sqrt{nc \ln c}\right) \) time, where \( c \) is the capacity of the concentrator. Thus, our quantum algorithms are asymptotically faster than their classical counterparts, when \( c \ln^2 c = o(n) \). In general, \( c = n^\mu \), satisfies \( c \ln^2 c = o(n) \), implying a time complexity of \( O(n^{0.5(1+\mu)} \ln n) \), for any \( \mu, 0 < \mu < 1 \).

1. Introduction

The promise of quantum computing in speeding up computations continues to attract research into exploring quantum algorithms for problems that arise in computer science, mathematics, and other scientific fields of study beyond searching and factorization. Indeed, several quantum algorithms and circuits have been reported for a wide range of classical problems extending from algebraic computations to pattern matching and many problems in graph theory. See for example [1,2] for a survey of quantum algorithms for Abelian and non-Abelian discrete Fourier transform, hidden subgroup problem, computing discrete logarithms, and several other problems in number theory, cryptography, and group theory. Another article by Montanaro provides an overview of quantum algorithms, and in particular surveys the complexity of quantum searching and optimization algorithms [3]. In graph theory, Grover’s search algorithm and quantum walk techniques have been used to solve matching and network flow problems [4–6] and graph traversals [7].

In this paper, we focus our attention on a different direction, namely the application of quantum computing to developing fast quantum algorithms to realize connection requests in concentrators. Concentration is a fundamental operation in interconnection. Thus, we think that developing fast quantum algorithms for realizing connection requests in concentrators will likely have a significant impact on routing in most interconnection networks. It is important to underscore that our goal in this paper is not to develop new techniques in quantum algorithms but rather to prove that for certain network connection models, a quantum controller is asymptotically better than a classical one. As mentioned above, the field of quantum algorithms, using quantum data processing elements to
solve computational problems, is an area that has a rich literature and is an active area of research with enthusiastic interest behind it. However, we believe that the power of quantum algorithms goes beyond that. In this work, we extend and adapt Grover’s seminal work to network controllers. In particular, we focus on identification of active processors in concentration assignments.

2. Sparse crossbar concentrators

Loosely speaking, concentrator routing (concentration) is a one-to-one mapping between two sets of objects, called inputs and outputs, where only inputs can be specified, and it is assumed that there are fewer outputs than inputs. Formally, a graph with a set of vertices, representing inputs and another set of vertices, representing outputs, where each vertex specifies a one-to-one mapping exists between every subset of inputs and some subset of outputs over a set of non-overlapping paths is called a concentrator. Pinsker established that concentrators with \( n \) inputs, \( m \) outputs, and at most \( 29n \) edges exist for all \( m \leq n \) in [8]. Since Pinsker’s seminal result, quite a few concentrator designs have been reported in the literature, see for example [9–18]. The earlier designs in [9–11] provide non-explicit solutions and can be viewed as upper bound results. The first explicit concentrator was reported in [12] using a binomial sparse crossbar design with \( O(n^{1.5}) \) edges. The design given in [13] falls outside the focus of our work as it is based on binary comparators and sorters. In this paper, we are concerned with designing classical and quantum algorithms for a particular set of concentrators that are referred to as fat-and-slim sparse crossbar concentrators. These were introduced in [14] and developed in [15–18]. We note that concentrators are further refined to study a family of bipartite graphs, widely-known as expanders [19–23]. Expanders provide bounded capacity concentration with \( O(n) \) edges. Our work on how to route concentration assignments in such graphs using quantum algorithms will be deferred to another place. In this paper, we give classical algorithms to realize concentration assignments in sparse crossbar concentrators and use Grover’s search to transform them to quantum algorithms, decreasing their time complexity. The rest of the paper is organized as follows. The next section provides the preliminary concepts needed to describe our results. In Section 3, we state our assumptions of classical and quantum models of computation. Sections 4 and 5 present our main results. The paper is concluded in Section 6 with a discussion of our results and possible directions for future research.

An \( (n,m,c) \)-concentrator is a graph with a set of \( n \) vertices, called inputs, a set of \( m \) vertices that are disjoint from the set of inputs, called outputs, and in which a \( k \)-matching, i.e. a set of \( k \) vertex-disjoint paths exists between every \( k \) inputs and some \( k \) outputs, \( 1 \leq k \leq c \leq m \). Such a concentrator is referred to as a bipartite or sparse crossbar concentrator if all paths are of length one, i.e. each path consists of two vertices and an edge that is represented by a crosspoint in the fat-and-slim crossbar model [14]. If \( c < m \) then an \( (n,m,c) \)-concentrator is said to have bounded capacity, and if \( c = m \), it is said to have full capacity. Henceforth, we will refer to the latter as an \( (n,m) \)-concentrator. A concentrator is called explicit if all of its edges (crosspoints) are specified, inexplicit if all of its edges exist with a non-vanishing probability other than 1, and semi-explicit if only a proper subset of its edges is explicitly specified.

Numerous explicit, semi-explicit, and inexplicit concentrators have been described in the literature as mentioned in the introduction. In this paper, we focus on two explicit concentrator designs introduced in [14] and one introduced in [16]. The first among these is a bipartite concentrator, called a full capacity fat-and-slim crossbar in which inputs are divided into two subsets of size \( n-m \) and \( m \). Each of the \( n-m \) inputs is connected to all the outputs, and forms the fat-part of the concentrator, whereas each of the \( m \) inputs is connected to exactly one output and forms its slim part. These concentrators are often diagrammed using a sparse crossbar representation as shown in Figure 1. As described, the left-hand side forms the fat part, and the diagonal line on right hand side forms the slim part of the concentrator. Adding all the crosspoints (edges) together, we find that an fat-and-slim \( (n,m) \)-concentrator consists of \( (n-m)m + m = (n-m+1)m \) crosspoints, and it was established in [14] that every bipartite \( (n,m) \)-concentrator must have at least \( (n-m+1)m \) crosspoints. Therefore, the full-capacity fat-and-slim concentrator is minimal with respect to the number of crosspoints.
The second concentrator is also a sparse crossbar concentrator similar in concept to a full-capacity fat-and-slim concentrator, but with a bounded capacity $c$ as illustrated in Figure 2 for $n = 9$, $m = 7$, $c = 2$. In this construction, the left-hand side forms the slim-part, whereas the right-hand side is added to provide a minimum capacity of $c$. This is ensured by requiring $c \leq m/c$ or $c \leq \sqrt{m}$. The key idea is to provide a sufficient number of crosspoints (edges) to each input in the right part so that, if it gets blocked by as many as $c-1$ inputs from the slim part, it can still find an idle output to match with. Requiring $c \leq m/c$ secures this as each input in the right part is connected to $\lfloor m/c \rfloor$ outputs. The construction also assumes that $n - m \leq c$. This assumption ensures that the main diagonal of crosspoints in the slim section on the left spans the entire set of outputs. It was shown in [14] that the crosspoint complexity of this construction remains with a factor of two of a lower bound of $\lfloor (n-c+1)m/(m-c+1) \rfloor$ crosspoints. We refer the reader to [14] for a more in-depth account of these two concentrator constructions, while we note that the capacity of the second sparse crossbar concentrator is actually larger than $c$ when $m/c > c$. In fact, the capacity of this crossbar is $\lfloor m/c \rfloor$ as each input in the fact section is connected to at least $\lfloor m/c \rfloor$ outputs. We adjust our notation to highlight this observation by replacing $c$ for the width of the inputs in the fat-section by $q$, and letting $n - m \leq q \leq m$, and $c = \lfloor m/q \rfloor$. Therefore, $c \leq \sqrt{m}$ is no longer required as described in [14]. For the rest of this paper, this updated construction will be referred to as ‘bounded capacity fat-and-slim concentrator’. Thus, we revise the capacity of the fat-and-slim crossbar in Figures 2 to 3.

We will also consider a third sparse concentrator that has a more regular structure [16]. This concentrator uses $n = pm$ inputs and $m$ outputs, and it is derived from the full-capacity fat-and-slim concentrator. Each output is connected to $n-m+1$ inputs as in the full-capacity fat-and-slim concentrator. On the other hand, each input is connected to between $m - \lfloor m/p \rfloor$ and $m - \lfloor m/p \rfloor + 1$ outputs, making this $(pm, m)$-concentrator nearly regular in terms of its in-degree (fan-in) as well. In this paper, we will consider the case when $p$ divides $m$.

Additionally, we only consider cases when $p \geq 3$. Figure 3 illustrates this construction for $p = 3$, and $m = 6$. It is seen that the out-degree of the construction falls between 4 and 5. Effectively, this construction is obtained from the full-capacity fat-and-slim concentrator by (i) dividing $n = pm$ columns of crosspoints into $p$ sections, (ii) chopping the diagonal of $m$ crosspoints in the slim part into $p$ groups of $m/p$ columns, and (iii) swapping each of those $m/p$ columns with an equal number of columns in one of the remaining $p-1$ sections on the left. Using the notation in [16], we denote the sets of inputs in the $p$ sections by $l_j, 1 \leq j \leq p$, and let
Figure 2. A bounded capacity fat-and-slim \((9, 7, 2)\)-concentrator, which is also a \((9, 7, 3)\)-concentrator by our extension of its capacity.

\[ I_j = \{x_{j,1}, x_{j,2}, \ldots, x_{j,m}\}. \]

We further let \( V_j = \cap_{k=1}^{m-1} N(x_{j,k}) \), \( U_j = O \setminus V_j \), and \( W_j \) denote the set of inputs that are connected to the outputs in \( U_j \), where \( N(x_{j,k}) \) denotes the neighbour set of input \( x_{j,k}, 1 \leq j \leq p, 1 \leq k \leq m \). In Figure 3, \( V_1 = \{z_3, z_4, z_5, z_6\}, V_2 = \{z_1, z_2, z_5, z_6\}, V_3 = \{z_1, z_2, z_3, z_4\}, U_1 = \{z_1, z_2\}, U_2 = \{z_3, z_4\}, U_3 = \{z_5, z_6\}, W_1 = \{x_{1,1}, x_{1,2}\}, W_2 = \{x_{2,3}, x_{2,4}\}, W_3 = \{x_{3,5}, x_{3,6}\} \).

The crux of this construction is a transformation theorem proved in [16].

As we describe in the latter part of Section 4, the construction of this concentrator makes the design of a classical routing algorithm for it more involved, but we establish that the time complexity of such an algorithm remains \(O(m)\) once active inputs are located.
3. Classical versus quantum routing model

The algorithms presented in this paper will be run on two models of computation: (3.1) classical model and (3.2) quantum model. It is important to highlight the differences between these two models to make a justifiable comparison of their execution times.

3.1. The classical routing model

In the classical model, we assume that quantities of interest are represented using classical bits of 0 and 1. For example, we use $\log n$ bits and $\log m$ bits to identify the inputs and outputs of an $(n, m)$-concentrator, respectively. We further assume that the bits within the representation of each input and/or output can be processed in parallel. For example, we can inspect all of the bits within a representation of any input (output) in parallel to determine if it is a particular input we seek, or we can compare the bits in representations of any two inputs (outputs) or a constant number of inputs (outputs) in parallel to see if they are the same in $O(1)$ time. Such bit-level operations can generally be carried out by logic circuits that consist of elementary logic components such as OR, AND, XOR, XNOR, and NOT gates that we assume have $O(1)$ computation time, and have a constant fan-in and fan-out, i.e. they have a constant number of inputs and outputs. We note that two or a constant number of $\log n$-bit numbers can be added, subtracted or compared using $O(\log n)$ 2-input, 2-output logic gates in $O(\log \log n)$ time using a prefix-adder as described in [24]. This $O(\log \log n)$ time will be suppressed in our time complexity formulas, leaving us with $O(1)$ time. It will also be assumed that $O(\log n)$-bit operands can be written and read in and out of a random access memory in $O(1)$ time. We will be using an array named in to specify if a given input is active. We will assume that in is an array that is stored in a random access memory with $O(1)$ access time. It is possible to relax $O(1)$ access time complexity. However, since such a non-constant access time complexity would be a multiplicative coefficient in the time complexity of the bottleneck step of both classical and quantum algorithms, we will omit this factor from our complexity calculations, effectively assume that the access time of in is $O(1)$.

3.2. The quantum routing model

In the quantum routing model, classical bits are replaced by quantum bits (qubits), and classical logic gates are replaced by those that represent unitary transformations. We assume that unitary gates have one or two inputs. All together, we allow $O(\log n)$ qubits in the quantum routing model. This allows us to work with possibly up to $n$ states in parallel. We further allow any combination of quantum gates to be used on these $O(\log n)$ qubits, but with the restriction that each qubit can source only one quantum gate at a time. This last restriction is a corollary of the no-cloning theorem [25]. This fanout restriction of one in the quantum routing model is not imposed on the bits in the classical routing model. However, we still assume that the fanout of each input is $O(1)$ in the classical model as well. Therefore, the two models can be viewed to be analogous, where any quantum operation on mutually exclusive $O(\log n)$ qubits takes $O(1)$ time much the same way each operation on $O(\log n)$ classical bits takes $O(1)$ time. The difference lies in the amount of parallelism afforded by the two models: in the classical model, we assume that the parallelism is limited to operations on any given pattern of $O(\log n)$ bits, whereas in the quantum routing model, the parallelism transcends any particular or fixed pattern as quantum operations are applied to the totality of the quantum state that includes all $O(n)$ binary patterns of $O(\log n)$ qubits. This vast amount of parallelism inherently present in quantum mechanical systems resulted in Shor’s quantum prime number factorization algorithm [26], which is exponentially faster than the best-known classical algorithm. Another key quantum algorithm, Grover’s quantum search [27] provides a speed-up of $O(\sqrt{n})$ over a sequential algorithm to search an element in an unordered list of $n$ elements. The latter algorithm will be used in Section 5 to reduce the routing time of concentration assignment on fat-and-slim concentrators using quantum algorithms. Before we describe these algorithms, we provide their classical analogs in the next section.
Routing an assignment on a sparse crossbar concentrator amounts to constructing a matching between any given subset of inputs and some subset of outputs of equal cardinality. The particular topologies of the three concentrators given in Section 2 guide the design of a routing algorithm for each concentrator as we describe next.

### 3.3. Full-capacity fat–slim (F–S) concentrator routing

**Algorithm 1: Classical Full-Capacity F–S Concentration**

```plaintext
1: function CLASSICAL FULL FS ROUTE(in, n, m)
   //in: n-bit array that marks up to m active inputs
   //n: number of inputs.
   //m: number of outputs.
2:     L ← list()
3:     for i ← 1 : m do //slim section
4:         if in[m − n + i] == 1 then
5:             pair(yi, zi);
6:         else
7:             L.insert(zi);
8:         end if
9:     end for
10:    for i ← 1 : n − m do //fat section
11:        if in[i] == 1 then
12:            {z = L.remove(); pair(xi, z;)
13:        end if
14:    end for
15: end function
```

### 4. Classical routing on concentrators

Our first algorithm, Algorithm 1 is a restatement of the algorithm that was originally described in [28] for a full-capacity fat-and-slim concentrator. We recall from Section 2 that the set of inputs is partitioned into two sets: those in the fat section: \( X = \{x_1, x_2, \ldots, x_{n-m}\} \) and those in the slim section: \( Y = \{y_1, y_2, \ldots, y_m\} \). The set of outputs of the concentrator is denoted by \( Z = \{z_1, z_2, \ldots, z_m\} \).

As we stated in the earlier section, the active inputs in a routing request, i.e. those to be concentrated are specified by ‘1’ entries in an n-bit array, named in. A routing request with \( k \) active inputs is first completed to a request with \( m \) inputs by combining the leftmost unused \( m-k \) inputs in in with the given \( k \) active inputs. Moreover, the leftmost \( m-k \) unused inputs in in can be determined in \( O(m) \) time by examining the leftmost \( m \) inputs of in.

Effectively, the first for loop assigns each active input in the slim section to the output with the same index value, while also inserting each unused output \( z_i \) into a list \( L \) of \( m \) elements as it checks if input \( y_i, 1 \leq i \leq m \) is active. The second for loop then assigns the active inputs from left to right in the fat section to the unused outputs, i.e. those that are inserted into \( L \) from top to bottom.

The time complexity of Algorithm 1 is easily seen to be \( O(n) \) steps, given that (i) each iteration in the first loop involves checking if a bit in the list in is ‘1’, pairing an input in the slim-section with an output or inserting a value into a list of \( m \) elements as it checks if input \( y_i, 1 \leq i \leq m \) is active. The second for loop then assigns the active inputs from left to right in the fat section to the unused outputs, i.e. those that are inserted into \( L \) from top to bottom.

The time complexity of Algorithm 1 is easily seen to be \( O(n) \) steps, given that (i) each iteration in the first loop involves checking if a bit in the list in is ‘1’, pairing an input in the slim-section with an output or inserting a value into a list of \( m \) elements, and (ii) each iteration in the second loop involves checking a bit, pairing an input in the fat-section with an output, and removing an output from the list \( L \). Checking if in[i] == 1 clearly takes \( O(1) \) time. It is further assumed that pairing an input and output as well as inserting or removing a value in and out of a list also takes \( O(1) \) time. This is a reasonable
assumption, considering that all three operations involve no more than a basic memory read or write operation.

As an example, let \( \text{in} = [0, 1, 0, 1, 0, 0, 1, 0, 1, 0] \), \( n = 11 \) and \( m = 5 \). This implies that set of active inputs are \( \{x_2, x_4, y_1, y_2, y_4\} \). On line 5, pairings are \( \{(y_1, z_1), (y_2, z_2), (y_4, z_4)\} \). This gives us \( L = \{z_3, z_5\} \) at the end of line 9. On line 12, pairings are \( \{(x_2, z_3), (x_4, y_5)\} \).

**Algorithm 2** Classical Bounded Capacity F--S Concentration

```plaintext
1: function CLASSICAL BOUNDED F--S ROUTE(in, n, m, q)
   //in: n-bit array that marks up to c active inputs.
   //n: number of inputs.
   //m: number of outputs.
   //q: the width of the fat section.
   //c = \( \lfloor m/q \rfloor \): capacity.
   //out: m-bit array that marks available outputs (initialized to all 1’s).
2:   for i ← 1 : n − q do //slim section
3:     if in[i] == 1 then
4:       pair(x_i, z_i); out[i] ← 0;
5:     end if
6:   end for
7:   L ← list() //pseudo-fat section
8:   for i ← 1 : q do
9:     if in[i + m] == 1 then
10:       L.insert(i);
11:     end if
12:   end for
13:   while i ← L.remove() do
14:     for j ← 0 : c − 1 do
15:       if out[i + qj] == 1 then
16:         pair(y_i, z_i+j);
17:         break
18:       end if
19:     end for
20:   end while
21: end function
```

### 4.1. Bounded capacity fat–slim concentration routing

Algorithm 2 extends the main idea of Algorithm 1 to routing active inputs in a bounded capacity fat-and-slim concentrator.\(^2\) As in Algorithm 1, the inputs are divided into fat and slim sections, but this time, the slim section is placed on the left and the pseudo-fat section is placed on the right in Figure 2. An \( m \)-bit array, named \( \text{out} \) is added to mark the outputs. The first for loop pairs the active inputs in the slim section with outputs whose indices coincide with those of the active inputs, while marking those outputs by entering ‘0’s into \( \text{out} \) in their index positions. The second for loop finds the active inputs in the pseudo-fat section on the right. The last nested loop pairs the active inputs in the pseudo-fat section on the right by searching for an available output from among the set of outputs to which each active is connected by a crosspoint.

Algorithm 2 has an execution time of \( O(n) = O(m) \).
(1) Lines 2–6 take $O(m)$ time as they involve steps to check if $in[i] = 1$, pair $x_i$ with $z_i$, and clear a bit in the out array.

(2) Lines 7–12 take $O(q)$ time as they involve checking $q$ array elements and adding them to a list.

(3) Lines 13–20 consist of a nested loop that takes $O(c)$ time. To see this, let $c', 0 \leq c' \leq c$, be the number of active inputs found in lines 2–6. In line 17, we exit the interior for loop. This happens when an available output is found for that input. The number of unsuccessful checks for an available output can at most be $c'$ and successful checks for such an output can at most be $c$. Since the number of elements in the list $L$ is also bounded by $c$, this nested loop takes $O(c)$ time.

As an example, let $in = [0, 1, 0, 1, 0, 0, 1]$, $n = 7$, $m = 6$ and $c = 2$. This implies that set of active inputs are $\{x_2, x_4, y_2\}$. On line 4, pairings are $\{(x_2, z_2), (x_4, x_4)\}$. This leaves us with only 1 unpaired active input, $y_2$. $y_2$ is connected to $z_2$, $z_4$ and $z_6$. On line 4 outputs $z_2$ are $z_4$ used. Therefore on line 16 the paring is $\{y_2, z_6\}$.

### 4.2. Regular fat–slim concentration routing

Our third classical algorithm restates the one given in [28] with a tighter time complexity analysis on some steps. This time, we have a sparse crossbar construction with $n = pm$ inputs and $m$ outputs as described in Section 2, where we assume that $p$ divides $m$. A routing request with $k$ active inputs is first completed to a request with $m$ inputs by combining the first unused $m-k$ inputs in $I_1$ with the given $k$ active inputs. This is always possible since if all $k \leq m$ active inputs belong to $I_1$ then its remaining $m-k$ inputs can be combined with the $k$ active inputs to obtain an $m$-request. On the other hand, if only some $k' < k \leq m$ of the $k$ active inputs belong to $I_1$ then $I_1$ must have $m - k' > m - k$ unused inputs, the first $m-k$ of which can be combined with the given $k$ active inputs to obtain an $m$-request. Moreover, the first $m-k$ unused inputs in $I_1$ can be determined in $O(m)$ time by examining all $m$ inputs in $I_1$ in the worst case. Therefore, we will assume that a $k$-request is completed to an $m$-request and specified by an array of $n$-bits named $in$, in which 0 and 1 bits represent the absence and presence of an active input respectively. At the beginning of the algorithm, the locations of these active inputs are searched and active inputs in $I_j = \{x_{j,1}, x_{j,2}, \ldots, x_{j,m}\}$ are placed in a linked list $R_j$, $1 \leq i \leq p$.

Algorithm 3 routes an $m$-request by considering two distinct cases: (a) One of the $R_j$’s has more than $m - m/p$ active inputs. (b) None of $R_j$’s has more than $m - m/p$ active inputs. That there can be no other case for all $p \geq 2$ is shown as follows. Suppose there exists $R_j$ with more than $m - m/p$ active inputs, for some $j$. Then there remain less than $m - (m - m/p) = m/p$ active inputs in the union of all the remaining $p-1$ sets of $m$ outputs. Therefore, another $m$-bit array, i.e. $R_j$ for some $j' \neq j$, $1 \leq j' \leq p$ with more than $m - m/p$ active inputs exists only if $m/p > m - m/p$, which implies $p < 2$ or $p = 1$, contradicting our assumption that $p \geq 2$ as stated in Section 2.

Now, continuing with Algorithm 3, lines 1 through 8 construct the linked lists in $O(n)$ time, assuming that we have $m$ active inputs as described above and determine the number of entries in each linked list by incrementing a size field each time an active input is found. Next, we see that the for statement in line 9 is iterated at most $p$ times, which occurs either when none of $R_j, 1 \leq j \leq p$ has more than $m - m/p$ active inputs or $R_p$ does. During the $j$th iteration, the if statement in line 10 checks if $R_j$ has more than $m - m/p$ 1’s. The size of each set can be queried in $O(1)$ time with the help of the size field that has been computed in line 5.

Case (a): If there exists such an $R_j$ then line 11 pairs all the active inputs in $R_j \cap W_j$, if any, and the outputs in $U_j$ to which those active inputs are connected by crosspoints. Line 12 then pairs as many active inputs in $R_i, 1 \leq i \neq j \leq p$ as possible, if any, with unused outputs in $U_j$. At this point, any active remaining inputs in $R_i, 1 \leq i \neq j \leq p$, are paired with unused outputs in $U_i, i \neq j$ in lines 13 and 14. This pairing is implemented in such a way that one of the $U_i, 1 \leq i \neq j$ is arbitrarily fixed ($U_{j-1}$ in the algorithm) to start the pairing. It is possible that $U_{j-1}$ may not even have an active input, but line 13 serves to initiate pairing of the active inputs in the remaining subsets of inputs other than $I_j$. We note that the number of active inputs in $I_{j-1}$ is less than $m/p = |U_{j+1}|$ and a full crossbar connection exists.
Algorithm 3 Classical Regular F-S Concentration

1: function CLASSICAL_REGULAR_F-S_ROUTE(input, output, m)

//input: n-bit array that marks up to m active inputs.
//output = m * input: number of outputs.
//R_j, 1 ≤ j ≤ p: linked lists of active inputs.
//m: number of outputs.

//The size fields of R_j, 1 ≤ j ≤ p are cleared.
2: for j ← 1: p do

3: for i ← 1: m do //find active elements

4: if input[j * m + i] == 1 then

5: R_j.insert(i); R_j.size++

6: end if

7: end for

8: end for

9: for j ← 1: p do

10: if R_j.size > m - m/p then //case (a)

11: Pair inputs in R_j ∩ W_j with outputs in U_j;

12: Pair inputs in U_j ∪ R_j with unpaired outputs in U_j;

13: Pair unpaired inputs in R_j−1 with outputs in U_j+1;

14: Pair unpaired inputs in U_j ∪ R_j−1 ∪ R_j with outputs in U_j−1;

15: Pair unpaired inputs in R_j with unpaired outputs in V_j;

16: return

17: else //case (b)

18: a[i] ← (m/p ≤ |R_j|); //p-bit array for reindexing

19: end if

20: end for

21: r ← prefixSum(a); s ← prefixSum(¬a);

22: d[i] ← a[i]r[i] + (¬a[i])(a[j]s[j] + r[j]); 1 ≤ j ≤ p; //new indices

23: R′_{d[p]} ← R_j; U′_{d[p]} ← U_j; 1 ≤ j ≤ p;

24: for j ← 2 : p do

25: Take first m/p elements from R′_j and place them in P′_j;

26: Pair inputs in P′_j with outputs in U′_j−1;

27: Q′_j ← R′_j \ P′_j;

28: end for

29: Pair inputs in R′_{d[p] + 1}, R′_{d[p] + 2}, ..., R′_p, P′_1, Q_2, Q_3, ..., Q_{p−1} with outputs in U′_{d[p]}, U′_{d[p] + 1}, ..., U′_p;

30: end function

between I_{j−1} and U_{j+1}. Similarly, in line 14, it is always possible to pair the number of active inputs in \bigcup_{j≠j'≠j−1} k_j with the outputs in U_{j−1} as U_{j−1} has m/p available outputs, which are more numerous than the active inputs in \bigcup_{j≠j'≠j−1} k_j and a full crossbar connection exists between the two sets. Here, the indexing is cyclical, and it is assumed that j, j − 1, and j + 1 are distinct from each other and p ≥ 3. If p = 2 then line 14 is not needed. Finally, any remaining inputs in R_j are paired with the remaining unused outputs in V_j in line 15 as there is a full crossbar connection between the inputs in I_j and outputs in V_j, 1 ≤ j ≤ p.

These steps are illustrated in Figure 4, where the circled numbers identify the steps in the algorithm. In this example, lines 13 and 14 do not result in any pairing. As another example, let m = 12, p = 3, R_1 = \{1, 2, 3, 5, 6, 7, 8, 9, 10\}, R_2 = \{3, 8\}, R_3 = \{1\}. We have U_1 = \{z_1, z_2, z_3, z_4\}, U_2 = \{z_5, z_6, z_7, z_8\}, U_3 = \{z_9, z_{10}, z_{11}, z_{12}\}, and |R_1| = 9 ≥ 8 = m - m/p. Thus, x_{1,1}, x_{1,2}, x_{1,3} are paired with z_1, z_2, z_3 in U_1 in line 11, x_{2,3} is paired with z_4 in U_1 in line 12, x_{3,1} in I_3 is paired with output z_5 in
The remaining active inputs in $R'_j$ are saved into $Q'_j, 2 \leq j \leq r[p]$ in line 27. Finally, the active inputs in $Q'_j$’s are concatenated with the active inputs in $R'_{[p]+1}, R'_{[p]+2}, \ldots, R'_{p}, R'_1$ which are then paired.

Figure 4. Routing in a regular (18, 6)-fat-and-slim concentrator.
together with the outputs in sets $U'_{r[p]+1}, U'_{r[p]+2}, \ldots, U'_p$ in line 29. Note that the number of active inputs in $R_{r[p]+1}', R_{r[p]+2}', \ldots, R'_p, R'_1, Q_2, Q_3, \ldots, Q_{r[p]}$ is given by $(p - r[p] + 1)m/p$ and it must match the number of active outputs in in $U'_{r[p]+1}, U'_{r[p]+2}, \ldots, U'_p$ after the pairing in line 26.

Algorithm 3 has a time complexity of $O(n)$ as shown below.

(1) Lines 2–8: This is a linear search of active inputs out of $n$ inputs, and therefore has a time complexity of $O(n)$.

(2) Lines 9–29: By the time we get to these lines, we already know where the active inputs are located because $R_j$'s consist only of active inputs. At this point the only task left to do is to route these active inputs. We route in two different ways as described in cases (a) and (b) in the algorithm. Both take $O(m)$ time as described below.

(a) Lines 9 and 10: We see that the for statement here is iterated at most $p$ times, which occurs either when none of $R_j, 1 \leq j \leq p$ has more than $m - m/p$ active inputs or $R_p$ does. During the $j$th iteration, the if statement in line 10 checks if $R_j$ has more than $m - m/p$ elements. Since we can query the size of sets in $O(1)$ time and comparison of values in the if statement can also be completed in $O(1)$ time, this line has a time complexity of $O(p)$.

(b) Lines 11–16: In case (a), there are more active inputs than non-diagonals can handle alone, and therefore diagonal crosspoints must be used. These lines execute at most once and take $O(m)$ time as explained below.

(i) Line 11 can be done by iterating through $R_j$ and checking if the current element is in $W_j$. Since $W_j$ is an interval where endpoints are fixed and known, the membership decision amounts to the comparison of the current element against the two endpoints, and this takes $O(1)$ time in our classical bit-parallel processor model. Moreover, the number of pairing operations is clearly bounded by $|W_j| = |U_j| = m/p$, and pairing an active input with the corresponding output in $U_j$ takes $O(1)$ time. Therefore, this line can be done in $O(m)$ time, the size limit of $R_j$.

(ii) Line 12: Let $\hat{R} = \bigcup_{i \neq j} R_i$. Given that all inputs in $\hat{R}$ are connected to outputs in $U_j$ by a full crossbar connection, the active inputs in $\hat{R}$ can be paired with unused outputs in $U_j$ by scanning $\hat{R}$ from left to right and $U_j$ from top to bottom until all unused outputs in $U_j$ are paired. As we represent $R_j, 1 \leq j \leq p$ by linked lists and since there are $m$ active inputs at the most, the pairing can be completed in $O(m)$ time in this case.

(iii) In line 13 we pair two sides of a full crossbar connection, i.e. inputs in $R_{j-1}$ and outputs in $U_{j+1}$. We loop through these sets, removing one element from each to build a complete matching. Given that $|R_{j-1}| \leq m, 2 \leq j \leq m + 1$, and $|U_{j+1}| = m/p$, this line has a time complexity of $O(m)$.

(iv) In line 14, we proceed as in line 12, and note that $|\bigcup_{i \neq j, i \neq j-1} R_i| \leq m$. Therefore this line has a time complexity of $O(m)$.

(v) Line 15 involves connecting the remaining active inputs in $I_j$ and the unused outputs in $V_j$. It is not difficult to verify that this step can be completed in $O(m - m/p)$ time as there is a full crossbar connection between the inputs in $R_j$ and $V_j$, $1 \leq j \leq p$. It follows that the time complexity of the for loop is $O(m - m/p)$.

(c) Lines 17–29: These lines are carried out in case (b) only. In this case, non-diagonal crosspoints can handle the assignment without diagonal crosspoints. Therefore, only the fat sections are used in the routing and these lines can be done in $O(m)$ time as explained below.

(i) Line 18 involves querying size of $R_j$ and comparing it to $m/p$, which can be completed in $O(1)$ time per each iteration of the for loop and in $O(p)$ time for all iterations.

(ii) Line 21 has prefix sum of $p$ elements that can be done in $O(p)$ time by an iterative summation of $p$ bits.
(iii) Line 22 involves an addition of two $O(\log p)$-bit numbers and a $2 \times 1$ multiplexer, both of which take $O(1)$ time and so the loop in this line takes $O(p)$ time.

(iv) Line 23 involves reindexing of $R_i$’s, which clearly takes $O(p)$ time.

(v) Lines 24–27 can be carried out in $O(m/p)$ time per each set of $m/p$ inputs and in $O(r[p] \times m/p) = O(m)$ time for all $r[p]$ sets of $m/p$ inputs.

(vi) Line 29 can be done by pairing the elements between the two sets described in these lines one by one in $O(m)$ time.

Remark 4.1: Algorithms 1, 2 and 3 are all optimal in terms of order of execution time as it takes $\Omega(n)$ time to read a concentration assignment.

Algorithm 4 Quantum Full F- -S Concentration

1: function QUANTUM FULL FS ROUTE($in$, $n$, $m$) //in: $n$-bit array that marks up to $m$ active inputs //n: number of inputs. //m: number of outputs.
2: $L \leftarrow$ list()
3: for $i \leftarrow 1 : m$ do // slim section
4:    if $in[m - n + i] == 1$ then
5:        pair($y_i$, $z_i$);
6:    else
7:        $L$.insert($z_i$);
8:    end if
9: end for
10: while $i \leftarrow$ GroverSearch($in[1 : n - m]$) do //fat section
11:    $z = L$.remove(); pair($x_i$, $z$);
12:    $in[i] = 0$
13: end while
14: end function

5. Quantum routing-Grover’s search

Now that we have established that classical routing algorithms all take $O(n)$ time for full and bounded capacity fat-and-slim concentrators described in Section 2, we turn our attention to quantum routing for them. Since both these concentrators require searching of active inputs out of all inputs, Grover’s search [27] provides a possible approach to identify such inputs faster in the quantum domain. Grover’s search approach has been followed to speed up various matching and network flow problems in [4,6]. These two papers were among the inspirations for our approach. We will use the version of Grover’s search analyzed in Theorem 3 of [29] and refer to it as GroverSearch in this paper. We note that GroverSearch is the only quantum step in the algorithms to be presented in this section and the rest of the paper. The time complexity of GroverSearch is known to be $O(\sqrt{n/k})$ to find one of $k$ marked items out of $n$ possible items. Thus, it takes $O(\sqrt{n/k})$ time to discover all $k$ items by unmarking discovered items one at a time. This is a widely used method even though the original source is unknown. The proof can be found in Lemma 4.1 of [30].

Remark 5.1: As in [4,6], we use Monte Carlo amplification to bound the total error of the entire algorithm resulting in an increase in run-time that is a logarithmic factor in the number of calls to the quantum subroutine, i.e. In $k$. Therefore, the time to discover all marked inputs is $O(\sqrt{n k \ln k})$.
5.1. Quantum full-capacity F–S concentration

The pseudo-code of our routing algorithm for the full-capacity fat-and-slim concentrator is given in Algorithm 4. As in the classical case, this algorithm consists of two parts. In the first part, we route all the active inputs to corresponding outputs in the slim section. We also make a list of idle inputs in this part to identify the corresponding outputs that are not used by the inputs in the slim section. Such outputs are then paired with the active inputs that are found by GroverSearch in the second part of the algorithm. This part of the algorithm is classical and deterministic. The second part of the algorithm provides us with quantum speedup. We use GroverSearch to locate the active inputs in the fat part of the concentrator. Once an active input is found, we pair it with one of the idle outputs that were found in the first part and unmark that input for another round of GroverSearch.

Algorithm 5  Quantum Bounded FS Concentration

1: function QUANTUM SPARSE FS ROUTE(in, n, m, q)
   //in: n-bit array that marks up to \( c \) active inputs.
   //n: number of inputs.
   //m: number of outputs.
   //q: the width of the fat section.
   //c = \( \lfloor m/q \rfloor \): capacity.
   //out: m-bit array that marks available outputs (initialized to all 1’s).
   2: while \( i \leftarrow \text{GroverSearch}(\text{in}[1:n-q]) \) do //slim section
      3:      pair(\( x_i, z_i \));
      4:      in[i] \leftarrow 0;
      5:      out[i] \leftarrow 0;
   6: end while
   7: \( L \leftarrow \text{list}() \); //pseudo-fat section
   8: while \( i \leftarrow \text{GroverSearch}(\text{in}[n-q+1:n]) \) do //slim section
      9:      \( L.\text{insert}(i) \);
     10: end while
   11: \( i \leftarrow L.\text{remove}() \) do
      12:      for \( j \leftarrow 0 : c-1 \) do
         13:         if out[\( i + qj \)] == 1 then
            14:            pair(y_i, z_{i+qj});
            15:            break
         16:         end if
      17: end for
     18: end while
   19: end function

The time complexity of Algorithm 4 is computed as follows. First note that the slim part of the algorithm is classical and deterministic. Since we assume that each query of the array in requires a constant time, identifying the active inputs, assigning them to their respective outputs, and also identifying the unused outputs in the slim part takes \( O(m) \) time. Now, let \( k \) be the number of active inputs in the fat part. It is obvious that \( k \) is upper bounded by \( m \). Hence by Remark 5.1, we find that the total time required for the fat part is \( O(\sqrt{m(n-m)} \ln m) \). Therefore the total time complexity of concentration in a full-capacity, fat-and-slim \((n,m)\)-concentrator is \( O(m + \sqrt{k(n-m)} \ln m) = O(m + \sqrt{nm} \ln m) \), and hence, given that \( n \geq m \), the total time complexity is \( O(\sqrt{nm} \ln m) \), using a quantum processor with \( O(\ln n) \) qubits. For comparison, any classical-bit processor routing algorithm for a full capacity, fat-and-slim \((n,m)\)-concentrator takes \( \Theta(n) \) time as we established in the earlier section (See Remark 4.1).
When \( n = \Theta(m \ln^2 m) \), we have \( O(\sqrt{m} \ln m) = O(n) \). Therefore, our algorithm performs better in quantum domain than any known classical algorithm if \( m \ln^2 m = o(n) \). One such value of \( m \) is clearly \( \Theta(\sqrt{n}) \). In general, it can easily be shown that \( m = n^\mu \), satisfies \( m \ln^2 m = o(n) \), for any \( \mu, 0 < \mu < 1 \), and the time complexity of our algorithm will increase less than linearly with \( n \) for such values of \( m \).

### Algorithm 6 Quantum Regular F--S Concentration

```plaintext
function QUANTUM REGULAR F--S ROUTE(in, p, m)
    // in: n-bit array that marks up to m active inputs.
    // n = m * p: number of inputs.
    // \( R_j, 1 \leq j \leq p \): linked lists of active inputs.
    // m: number of outputs.
    // The size fields of \( R_j, 1 \leq j \leq p \) are cleared.
    while (i, j) ← GroverSearch(in) do
        \( R_j \), insert(i); \( R_j \), size++
    end while
    for j ← 1 : p do
        if \( R_j \), size > m − m/p then //case (a)
            Pair inputs in \( R_j \) ∩ \( W_j \) with outputs in \( U_j \);
            Pair inputs in \( \cup_{i≠j} R_i \) with unpaired outputs in \( U_j \);
            Pair unpaired inputs in \( R_{j-1} \) with outputs in \( U_{j+1} \);
            Pair unpaired inputs in \( \cup_{i≠j,j-1} R_i \) with outputs in \( U_{j-1} \);
            Pair unpaired inputs in \( R_j \) with unpaired outputs in \( V_j \);
        return
    end if
    r ← prefixSum(a);
    \( d[j] ← a[j]r[j] + (\neg a[j]) (s[j] + r[p]) \); 1 ≤ j ≤ p; //new indices
    \( R'_{[0:j]} ← R_j \)
    for j ← 2 : r[p] do
        Take first \( m/p \) elements from \( R'_j \) and place them in \( P'_j \);
        Pair inputs in \( P'_j \) with outputs in \( U'_{j-1} \);
        \( Q'_j ← R'_j \backslash P'_j \)
        end for
        Pair inputs in \( R'_{[p]}+1 \), \( R'_{[p]}+2 \), ..., \( R'_p \), \( R'_1 \), \( Q_2 \), \( Q_3 \), ..., \( Q_r[p] \)
        with outputs in \( U'_{[p]} \), \( U'_{[p]}+1 \), ..., \( U'_{p} \);
    end function
```

#### 5.2. Quantum bounded-capacity F--S concentration

The time complexity of the classical routing algorithm for a bounded capacity fat-and-slim concentrator can also be reduced using quantum search as well, as described in Algorithm 5. The slim section of the algorithm finds at most \( c \) marked elements out of at most \( m \) total elements since \( n − q ≤ m \). Similarly, the fat section of the algorithm also finds \( c \) marked elements out of at most \( m \) total elements since \( q ≤ m \). The total time complexity of both parts is \( O(\sqrt{mc} \ln c) \). Lines 12–19 is the same as 13–20 of Algorithm 2. Therefore, the total time complexity is also \( O(\sqrt{mc} \ln c) = O(\sqrt{n} \ln c) \). Once again, this algorithm perform better than it’s classical counterpart when \( c \ln^2 c = o(n) \).
Remark 5.2: In full capacity F–S concentrator, active inputs for the dense part are located with Grover’s Search while for bounded capacity FS concentrator, all active inputs are located with Grover’s Search.

5.3. Quantum regular FS concentration

Our last quantum algorithm handles concentration assignments for regular fat-and-slim concentrators. It is obtained by replacing the search for active inputs by \texttt{GroverSearch}. From the analysis in Subsection 4.2 we know that once the active inputs are found, the routing takes \( O(m) \). Just like full capacity fat-and-slim concentrator, the search as well as the total time takes \( O(\sqrt{nm} \ln m) \). Therefore the analysis done in Subsection 5.1 also applies here.

6. Conclusions and future work

We have presented three quantum algorithms, all with smaller time complexity as compared to their classical counterparts. For full-capacity fat-and-slim concentrator, our quantum algorithm has a time complexity of \( O(\sqrt{nm} \log m) \) versus the classical routing algorithm complexity of \( O(n) \). Thus, in this case, our quantum algorithm has a smaller time complexity if \( m = n^\mu, 0 < \mu < 1 \). For regular fat-and-slim and bounded-capacity fat-and-slim concentrators, similar speed-up formulas apply as established in the paper. The main take-away from these proofs is that network that has a quantum controller can be asymptotically superior to a network that has a purely classical controller. Thus, a quantum controller utilizes the resources of a classical networks more efficiently.

It should be added that, in both classical and quantum domains, the search process can be speeded up by replicating controller resources. In the case of a classical search algorithm, using \( k \) controllers results in a factor of \( k \) reduction in time complexity from \( O(n) \) to \( O(n/k) \). On the other hand, in the quantum domain, this reduction in time will be limited to a factor \( \sqrt{k} \). Thus, the separation in time complexities between the classical and quantum domains become more pronounced only when \( k \) is small.

As for future research, we will investigate the possibility of quantum accelerations for other concentrator architectures and explore other quantum algorithms such as quantum walk in our research. Another related problem where a similar approach might be helpful is routing on packet switching networks. These and other related problems will be further explored in another paper.

Notes

1. The maximum number of inputs that can be concentrated in unit time independent of the location of the inputs [31].
2. Even though we refer to this construction as a bounded capacity, fat-and-slim concentrator, the fat section on the right is not completely filled with crosspoints as in the case of full-capacity fat-and-slim concentrator.
3. The selection of \( I_1 \) is arbitrary and simplifies our description for the completion of a \( k \)-assignment to an \( m \)-assignment. The algorithm will work regardless of which set of inputs selected.

Disclosure statement

No potential conflict of interest was reported by the author(s).

References

[1] Childs AM, Van Dam W. Quantum algorithms for algebraic problems. Reviews of Modern Physics. 2010;82(1):1–52. doi: 10.1103/RevModPhys.82.1
[2] Mosca M. Quantum algorithms. arXiv preprint arXiv:0808.0369, 2008.
[3] Montanaro A. Quantum algorithms: an overview. npj Quantum Inf. 2016;2(1):1–8. doi: 10.1038/npjqi.2015.23
[4] Ambainis A, Špalek R. Quantum algorithms for matching and network flows. In: Annual symposium on theoretical aspects of computer science. Springer; 2006. p. 172–183.
[5] Ambainis A. Quantum walk algorithm for element distinctness. SIAM J Comput. 2007;37(1):210–239. doi:10.1137/00097539705447311
[6] Dörn S. Quantum algorithms for matching problems. Theory Comput Syst. 2009;45(3):613–628. doi:10.1007/s00224-008-9118-x
[7] Childs AM, Cleve R, Deotto E, et al. Exponential algorithmic speedup by a quantum walk. In: Proceedings of the thirty-fifth annual ACM symposium on theory of computing. 2003. p. 59–68.
[8] Pinsker MS. On the complexity of a concentrator. In: 7th international telegraph conference. Vol. 4. Citeseer; 1973. p. 1–318.
[9] Pippenger N. Super concentrators. SIAM J Comput. 1977;6(2):298–304. doi:10.1137/0206022
[10] Chung F. On concentrators, super concentrators, generalizers, and nonblocking networks. Bell Syst Tech J. 1979;58(8):1765–1777. doi:10.201101/17.1979.58.issue-8
[11] Bassalygo L. Asymptotically optimal switching circuits. Problemy Peredachi Informatsii. 1981;17(3):206–211.
[12] Nakamura S, Masson GM. Lower bounds on crosspoints in concentrators. IEEE Trans Comput. 1982;C-31(12):1173–1179. doi:10.1109/TC.1982.1675941
[13] Chien MV, Oruç AY. High performance concentrators and superconcentrators using multiplexing schemes. IEEE Trans Commun. 1994;42(11):3045–3050. doi:10.1109/26.328986
[14] Oruç AY, Huang H. Crosspoint complexity of sparse crossbar concentrators. IEEE Trans Inf Theory. 1996;42(5):1466–1471. doi:10.1109/18.532886
[15] Gündüzhan E, Oruç AY. Structure and density of sparse crossbar concentrators. In: Du D, Huan F, editors. Advances in switching networks.Citeseer; 1997. p. 169–180.
[16] Guo W, Oruç AY. Regular sparse crossbar concentrators. IEEE Trans Comput. 1998;47(3):363–368. doi:10.1109/12.660174
[17] Ratan R, Oruç A. Performance evaluation of input queued buffered sparse-crossbar packet concentrators. In: Proceedings of conference on information sciences and systems CISS. Vol. 3. Johns Hopkins University, Baltimore, Citeseer; 2003.
[18] Ratan R, Oruç AY. Self-routing quantum sparse crossbar packet concentrators. IEEE Trans Comput. 2010;60(10):1390–1405. doi:10.1109/TC.2010.145
[19] Margulis GA. Explicit constructions of concentrators. Problemy Peredachi Informatsii. 1973;9(4):71–80.
[20] Gabber O, Galil Z. Explicit constructions of linear-sized superconcentrators. J Comput Syst Sci. 1981;22(3):407–420. doi:10.1016/0022-0000(81)90040-4
[21] Alon N. On the number of subgraphs of prescribed type of graphs with a given number of edges. Isr J Math. 1981;38(1–2):116–130. doi:10.1007/BF02761855
[22] Tanner RM. Explicit concentrators from generalized n-gons. SIAM J Algebraic Discrete Methods. 1984;5(3):287–293. doi:10.1137/0605030
[23] Jimbo S, Maruoka A. Expanders obtained from affine transformations. In: Proceedings of the seventeenth annual ACM symposium on theory of computing. 1985. p. 88–97.
[24] Brent RP, Kung HT. A regular layout for parallel adders. IEEE Trans Comput. 1982;C-31(3):260–264. doi:10.1109/TC.1982.1675982
[25] Bužek V, Hillery M. Quantum copying: beyond the no-cloning theorem. Phys Rev A. 1996;54(3):1844–1852. doi:10.1103/PhysRevA.54.1844
[26] Shor PW. Algorithms for quantum computation: discrete logarithms and factoring. In: Proceedings 35th annual symposium on foundations of computer science. IEEE; 1994. p. 124–134.
[27] Grover LK. A fast quantum mechanical algorithm for database search. In: Proceedings of the twenty-eighth annual ACM symposium on theory of computing. 1996. p. 212–219.
[28] Guo W, Oruç AY. Explicit construction of bounded capacity sparse crossbar concentrators. In: Proceedings of the conference on information sciences and systems. Vol. 1. Department of Electrical Engineering, Johns Hopkins University, Baltimore, MD; 1996. p. 233.
[29] Boyer M, Brassard G, Hayer P, et al. Tight bounds on quantum searching. Fortschritte der Physik: Progress of Physics. 1998;46(4–5):493–505. doi:10.1002/(ISSN)1521-3978
[30] Childs AM, Kothari R. Quantum query complexity of minor-closed graph properties. SIAM J Comput. 2012;41(6):1426–1450. doi:10.1137/110833026
[31] Oruç AY. Foundations of interconnection networks. 2023. Under contract, accepted for publication.