Quantum Lower and Upper Bounds for 2D-Grid and Dyck Language

Andris Ambainis
Center for Quantum Computer Science, Faculty of Computing, University of Latvia, Riga, Latvia

Kaspars Balodis
Center for Quantum Computer Science, Faculty of Computing, University of Latvia, Riga, Latvia

Jānis Iraids
Center for Quantum Computer Science, Faculty of Computing, University of Latvia, Riga, Latvia

Kamil Khadiev
Kazan Federal University, Russia

Vladislavs Kļevickis
Center for Quantum Computer Science, Faculty of Computing, University of Latvia, Riga, Latvia

Krišjānis Prūsis
Center for Quantum Computer Science, Faculty of Computing, University of Latvia, Riga, Latvia

Yixin Shen
Université de Paris, CNRS, IRIF, F-75006 Paris, France

Juris Smotrovs
Center for Quantum Computer Science, Faculty of Computing, University of Latvia, Riga, Latvia

Jevgēnijs Vihrovs
Center for Quantum Computer Science, Faculty of Computing, University of Latvia, Riga, Latvia

Abstract

We study the quantum query complexity of two problems.

First, we consider the problem of determining if a sequence of parentheses is a properly balanced one (*a Dyck word*), with a depth of at most k. We call this the $\textsc{Dyck}_{k,n}$ problem. We prove a lower bound of $\Omega(c^k\sqrt{n})$, showing that the complexity of this problem increases exponentially in k. Here n is the length of the word. When k is a constant, this is interesting as a representative example of star-free languages for which a surprising $\tilde{O}(\sqrt{n})$ query quantum algorithm was recently constructed by Aaronson et al. [1]. Their proof does not give rise to a general algorithm. When k is not a constant, $\textsc{Dyck}_{k,n}$ is not context-free. We give an algorithm with $O\big(\sqrt{n}(\log n)^{0.5k}\big)$ quantum queries for $\textsc{Dyck}_{k,n}$ for all k. This is better than the trival upper bound n for $k = o\big(\frac{\log(n)}{\log\log n}\big)$.

Second, we consider connectivity problems on grid graphs in 2 dimensions, if some of the edges of the grid may be missing. By embedding the "balanced parentheses" problem into the grid, we show a lower bound of $\Omega(n^{1.5-\epsilon})$ for the directed 2D grid and $\Omega(n^{2-\epsilon})$ for the undirected 2D grid. The directed problem is interesting as a black-box model for a class of classical dynamic programming strategies including the one that is usually used for the well-known edit distance problem. We also show a generalization of this result to more than 2 dimensions.

2012 ACM Subject Classification Theory of computation → Quantum query complexity

Keywords and phrases Quantum query complexity, Quantum algorithms, Dyck language, Grid path

Digital Object Identifier 10.4230/LIPIcs.MFCS.2020.8

Related Version A full version of the paper is available at `https://arxiv.org/pdf/2007.03402.pdf`.

Funding Supported by QuantERA ERA-NET Cofund in Quantum Technologies implemented within the European Union's Horizon 2020 Programme (QuantAlgo project) and ERDF project 1.1.1.5/18/A/020 "Quantum algorithms: from complexity theory to experiment". The research was funded by the subsidy allocated to Kazan Federal University for the state assignment in the sphere of scientific activities.

Acknowledgements The authors would like to thank the anonymous reviewers for their constructive comments and suggestions.

1 Introduction

We study the quantum query complexity of two problems:

Quantum complexity of regular languages. Consider the problem of recognizing whether an n-bit string belongs to a given regular language. This models a variety of computational tasks that can be described by regular languages. In the quantum case, the most commonly used model for studying the complexity of various problems is the query model. For this setting, Aaronson, Grier and Schaeffer [1] recently showed that any regular language L has one of three possible quantum query complexities on inputs of length n: $\Theta(1)$ if the language can be decided by looking at $O(1)$ first or last symbols of the word; $\tilde{\Theta}(\sqrt{n})$ if the best way to decide L is Grover's search (for example, for the language consisting of all words containing at least one letter a); $\Theta(n)$ for languages in which we can embed counting modulo some number p which has quantum query complexity $\Theta(n)$.

As shown in [1], a regular language being of complexity $\tilde{O}(\sqrt{n})$ (which includes the first two cases above) is equivalent to it being star-free. Star-free languages are defined as the languages which have regular expressions not containing the Kleene star (if it is allowed to use the complement operation). Star-free languages are one of the most commonly studied subclasses of regular languages and there are many equivalent characterizations of them. One class of the star-free languages mentioned in [1] is the Dyck languages (with one type of parenthesis) with constant height k. Dyck language with height k consists of words with balanced number of parentheses such that in no prefix the number of opening parentheses exceeds the number of closing parentheses by more than k; we denote the problem of determining if an input of length n belongs to this language by $\textsc{Dyck}_{k,n}$. In case of unbounded height $k = \frac{n}{2}$, the language is a fundamental example of a context-free language that is not regular. When more types of parenthesis are allowed, the famous Chomsky–Schützenberger representation theorem shows that any context-free language is the homomorphic image of the intersection of a Dyck language and a regular language.

Our results. We show that an exponential dependence of the complexity on k is unavoidable. Namely, for the balanced parentheses language, we have

- there exists $c > 1$ such that, for all $k \leq \log n$, the quantum query complexity is $\Omega(c^k\sqrt{n})$;
- If $k = c \log n$ for an appropriate constant c, the quantum query complexity is $\Omega(n^{1-\epsilon})$.

Thus, the exponential dependence on k is unavoidable and distinguishing sequences of balanced parentheses of length n and depth $\log n$ is almost as hard as distinguishing sequences of length n and arbitrary depth.

Similar lower bounds have recently been independently proven by Buhrman et al. [8].

Additionally, we give an explicit algorithm (see Theorem 3) for the decision problem $\textsc{Dyck}_{k,n}$ with $O\big(\sqrt{n}(\log n)^{0.5k}\big)$ quantum queries. The algorithm also works when k is not a constant and is better than the trivial upper bound of n when $k = o\Big(\frac{\log(n)}{\log\log n}\Big)$.

Finding paths on a grid. The second problem that we consider is graph connectivity on subgraphs of the 2D grid. Consider a 2D grid with vertices (i, j), $i \in \{0, 1, \ldots, n\}, j \in \{0, 1, \ldots, k\}$ and edges from (i, j) to $(i + 1, j)$ and $(i, j + 1)$. The grid can be either directed (with edges in the directions of increasing coordinates) or undirected. We are given an unknown subgraph G of the 2D grid and we can perform queries to variables x_u (where u is an edge of the grid) defined by $x_u = 1$ if u belongs to G and 0 otherwise. The task is to determine whether G contains a path from $(0, 0)$ to (n, k).

Our interest in this problem is driven by the edit distance problem. In the edit distance problem, we are given two strings x and y and have to determine the smallest number of operations (replacing one symbol by another, removing a symbol or inserting a new symbol) with which one can transform x to y. If $|x| \leq n, |y| \leq k$, the edit distance is solvable in time $O(nk)$ by dynamic programming [13]. If $n = k$ then, under the strong exponential time hypothesis (SETH), there is no classical algorithm computing edit distance in time $O(n^{2-\epsilon})$ for $\epsilon > 0$ [5] and the dynamic programming algorithm is essentially optimal.

However, SETH does not apply to quantum algorithms. Namely, SETH asserts that there is no algorithm for general instances of SAT that is substantially better than naive search. Quantumly, a simple use of Grover's search gives a quadratic advantage over naive search. This leads to the question: can this quadratic advantage be extended to edit distance (and other problems that have lower bounds based on SETH)?

Since edit distance is quite important in classical algorithms, the question about its quantum complexity has attracted a substantial interest from various researchers. Boroujeni et al. [7] invented a better-than-classical quantum algorithm for approximating the edit distance which was later superseded by a better classical algorithm of [9]. However, there has been no quantum algorithms computing the edit distance exactly (which is the most important case).

The main idea of the classical algorithm for edit distance is as follows:

- We construct a weighted version of the directed 2D grid (with edge weights 0 and 1) that encodes the edit distance problem for strings x and y, with the edit distance being equal to the length of the shortest directed path from $(0, 0)$ to (n, k).
- We solve the shortest path problem on this graph and obtain the edit distance.

As a first step, we can study the question of whether the shortest path is of length 0 or more than 0. Then, we can view edges of length 0 as present and edges of length 1 as absent. The question "Is there a path of length of 0?" then becomes "Is there a path from $(0, 0)$ to (n, k) in which all edges are present?". A lower bound for this problem would imply a similar lower bound for the shortest path problem and a quantum algorithm for it may contain ideas that would be useful for a shortest path quantum algorithm.

Our results. We use our lower bound on the balanced parentheses language to show an $\Omega(n^{1.5-\epsilon})$ lower bound for the connectivity problem on the directed 2D grid. This shows a limit on quantum algorithms for finding edit distance through the reduction to shortest paths. More generally, for an $n \times k$ grid ($n > k$), our proof gives a lower bound of $\Omega((\sqrt{n}k)^{1-\epsilon})$.

The trivial upper bound is $O(nk)$ queries, since there are $O(nk)$ variables. There is no nontrivial quantum algorithm, except for the case when k is very small. Then, we show that the connectivity problem can be solved with $O(\sqrt{n}\log^{k/2} n)$ quantum queries[1] but this bound becomes trivial already for $k = \Omega(\frac{\log n}{\log\log n})$.

For the undirected 2D grid, we show a lower bound of $\Omega((nk)^{1-\epsilon})$, whenever $k \geq \log n$. Thus, the naive algorithm is almost optimal in this case. We also extend both of these results to higher dimensions, obtaining a lower bound of $\Omega((n_1 n_2 \dots n_d)^{1-\epsilon})$ for an undirected $n_1 \times n_2 \times \dots \times n_d$ grid in d dimensions and a lower bound of $\Omega(n^{(d+1)/2-\epsilon})$ for a directed $n \times n \times \dots \times n$ grid in d dimensions.

In a recent work, an $\Omega(n^{1.5})$ lower bound for edit distance was shown by Buhrman et al. [8], assuming a quantum version of the Strong Exponential Time hypothesis (QSETH). As part of this result they give an $\Omega(n^{1.5})$ query lower bound for a different path problem on a

[1] Aaronson et al. [1] also give a bound of $O(\sqrt{n}\log^{m-1} n)$ but in this case m is the rank of the syntactic monoid which can be exponentially larger than k.

2D grid. Then QSETH is invoked to prove that no quantum algorithm can be faster than the best algorithm for this shortest path problem. Neither of the two results follow directly one from another, as different shortest path problems are used.

2 Definitions

For a word $x \in \Sigma^*$ and a symbol $a \in \Sigma$, let $|x|_a$ be the number of occurrences of a in x.

For two (possibly partial) Boolean functions $g : G \to \{0,1\}$, where $G \subseteq \{0,1\}^n$, and $h : H \to \{0,1\}$, where $H \subseteq \{0,1\}^m$, we define the composed function $g \circ h : D \to \{0,1\}$, with $D \subseteq \{0,1\}^{nm}$, as $(g \circ h)(x) = g\big(h(x_1,\ldots,x_m),\ldots,h(x_{(n-1)m+1},\ldots,x_{nm})\big)$. Given a Boolean function f and a nonnegative integer d, we define f^d recursively as f iterated d times: $f^d = f \circ f^{d-1}$ with $f^1 = f$.

For a matrix Γ, $\|\Gamma\|$ denotes the spectral norm of Γ: $\|\Gamma\| = \max_{\overrightarrow{x}\neq 0} \frac{\|\Gamma\overrightarrow{x}\|}{\|\overrightarrow{x}\|}$ where $\|\overrightarrow{x}\|$ is the 2-norm of a vector.

Quantum query model. We use the standard form of the quantum query model. Let $f : D \to \{0,1\}, D \subseteq \{0,1\}^n$ be an n variable function we wish to compute on an input $x \in D$. We have an oracle access to the input x – it is realized by a specific unitary transformation usually defined as $|i\rangle|z\rangle|w\rangle \to |i\rangle|z + x_i \pmod 2\rangle|w\rangle$ where the $|i\rangle$ register indicates the index of the variable we are querying, $|z\rangle$ is the output register, and $|w\rangle$ is some auxiliary work-space. An algorithm in the query model consists of alternating applications of arbitrary unitaries independent of the input and the query unitary, and a measurement in the end. The smallest number of queries for an algorithm that outputs $f(x)$ with probability $\geq \frac{2}{3}$ on all x is called the quantum query complexity of the function f and is denoted by $Q(f)$.

Let a symmetric matrix Γ be called an adversary matrix for f if the rows and columns of Γ are indexed by inputs $x \in D$ and $\Gamma_{xy} = 0$ if $f(x) = f(y)$. Let $\Gamma^{(i)}$ be a similarly sized matrix such that $\Gamma^{(i)}_{xy} = \begin{cases} \Gamma_{xy} & \text{if } x_i \neq y_i \\ 0 & \text{otherwise} \end{cases}$. Then let $Adv^{\pm}(f) = \max\limits_{\substack{\Gamma \text{ - an adversary} \\ \text{matrix for } f}} \frac{\|\Gamma\|}{\max_i \|\Gamma^{(i)}\|}$ be called the adversary bound and let $Adv(f) = \max\limits_{\substack{\Gamma \text{ - an adversary matrix for } f \\ \Gamma \text{ - nonnegative}}} \frac{\|\Gamma\|}{\max_i \|\Gamma^{(i)}\|}$ be called the positive adversary bound. The following facts will be relevant for us: $Adv(f) \leq Adv^{\pm}(f)$; $Q(f) = \Theta(Adv^{\pm}(f))$ [12]; $Adv^{\pm}$ composes exactly even for partial Boolean functions f and g, meaning, $Adv^{\pm}(f \circ g) = Adv^{\pm}(f) \cdot Adv^{\pm}(g)$ [11, Lemma 6].

Reductions. We will say that a Boolean function f is reducible to g and denote it by $f \leqslant g$ if there exists an algorithm that given an oracle O_x for an input of f transforms it into an oracle O_y for g using at most $O(1)$ calls of oracle O_x such that $f(x)$ can be computed from $g(y)$. Therefore, from $f \leqslant g$ we conclude that $Q(f) \leq Q(g)$ because one can compute $f(x)$ using the algorithm for $g(y)$ and the reduction algorithm that maps x to y.

Dyck languages of bounded depth. Let Σ be an alphabet consisting of two symbols: (and). The Dyck language L consists of all $x \in \Sigma^*$ that represent a correct sequence of opening and closing parentheses. We consider languages L_k consisting of all words $x \in L$ where the number of opening parentheses that are not closed yet never exceeds k. The language L_k corresponds to a query problem $\textsc{Dyck}_{k,n}(x_1,...,x_n)$ where $x_1,\ldots,x_n \in \{0,1\}$ describe a word of length n in the natural way: the i^{th} symbol of x is (if $x_i = 0$ and) if $x_i = 1$. $\textsc{Dyck}_{k,n}(x) = 1$ iff the word x belongs to L_k. For all $x \in \{0,1\}^n$, we define $f(x) = |x|_0 - |x|_1$,

we call it the **balance**. We define a $+k$-substring (resp. $-k$-substring) as a substring whose balance is equal to k (resp. equal to $-k$). A $\pm k-$substring is a substring whose balance is equal to k in absolute value. For all $0 \leq i \leq j \leq n-1$, we define $x[i,j] = x_i, x_{i+1}, \cdots, x_j$. Finally, we define $h(x) = \max_{0 \leq i \leq n-1} f(x[0,i])$ and $h^-(x) = \min_{0 \leq i \leq n-1} f(x[0,i])$. A substring $x[i,j]$ is *minimal* if it does not contain a substring $x[i',j']$ such that $(i,j) \neq (i',j')$, and $f(x[i',j']) = f(x[i,j])$.

Connectivity on a directed 2D grid. Let $G_{n,k}$ be a directed version of an $n \times k$ grid in two dimensions, with vertices $(i,j), i \in \{0,1,\ldots,n\}, j \in \{0,1,\ldots,k\}$ and directed edges from (i,j) to $(i+1,j)$ (if $i < n$) and from (i,j) to $(i,j+1)$ (if $j < k$). If G is a subgraph of $G_{n,k}$, we can describe it by variables x_e corresponding to edges e of $G_{n,k}$: $x_e = 1$ if the edge e belongs to G and $x_e = 0$ otherwise. We consider a problem 2D-DConnectivity in which one has to determine if G contains a path from $(0,0)$ to (n,k): 2D-DConnectivity$_{n,k}(x_1,\ldots,x_m) = 1$ (where m is the number of edges in $G_{n,k}$) iff such a path exists.

Connectivity on an undirected 2D grid. Let $G_{n,k}$ be an undirected $n \times k$ grid and let G be a subgraph of $G_{n,k}$. We describe G by variables x_e in a similar way and define 2D-Connectivity$_{n,k}(x_1,\ldots,x_m) = 1$ iff G contains a path from $(0,0)$ to (n,k). We also consider d dimensional versions of these two problems, on $n_1 \times n_2 \times \ldots n_d$ grids. In the directed version (dD-DConnectivity), we have a subgraph G of a directed grid (with edges directed in the directions from $(0,\ldots,0)$ to $(n_1,\ldots,n_d)$) and dD-DConnectivity$(x_1,\ldots,x_m) = 1$ iff G contains a directed path from $(0,\ldots,0)$ to $(n_1,\ldots,n_d)$. The undirected version is defined similarly, with an undirected grid instead of a directed one.

3 A quantum algorithm for membership testing of Dyck$_{k,n}$

In this section, we give a quantum algorithm for Dyck$_{k,n}(x)$, where k can be a function of n. The general idea is that Dyck$_{k,n}(x) = 0$ if and only if one of the following conditions holds: (i) x contains a $+(k+1)$-substring; (ii) x contains a substring $x[0,i]$ such that the balance $f(x[0,i]) = -1$; (iii) the balance of the entire word $f(x) \neq 0$.

The main algorithm is presented in Section 3.2. It based on a subroutine presented in Section 3.1.

3.1 $\pm k$-Substring Search algorithm

The goal of this section is to describe a quantum algorithm which searches for a substring $x[i,j]$ that has a balance $f(x[i,j]) \in \{+k,-k\}$ for some integer k. Throughout this section, we find and consider only **minimal** substrings. A substring is minimal if it does not contain a proper substring with the same balance. Throughout this section we use the following easily verifiable facts:

- For any two minimal $\pm k$-substrings $x[i,j]$ and $x[k,l]$: $i < k \implies j < l$. This induces a natural linear order among all $\pm k$-substrings according to their starting (or, equivalently, ending) positions.
- Minimal $+k$-substrings do not intersect with minimal $-k$-substrings.
- If $x[l_1,r_1]$ and $x[l_2,r_2]$ with $l_1 < l_2$ are two **consecutive** minimal $(k-1)$-substrings and their signs are the same, then $x[l_1,r_2]$ is a k-substring with this sign.

This algorithm is the basis of our algorithms for Dyck$_{k,n}$. The algorithm works in a recursive way. It searches for two consecutive minimal $\pm(k-1)$-substrings $x[l_1,r_1]$ and $x[l_2,r_2]$ such that they either overlap or there are no $\pm(k-1)$-substrings between them. If both substrings

$x[l_1, r_1]$ and $x[l_2, r_2]$ are $+(k-1)$-substrings, then we get a minimal $+k$-substring in total. If both substrings are $-(k-1)$-substrings, then we get a minimal $-k$-substring in total.

Our algorithm utilizes three subroutines. The first one is $\textsc{FindAtLeftmost}_k(l, r, t, d, s)$ which accepts as inputs: the borders l and r, where l and r are integers such that $0 \leq l \leq r \leq n-1$; a position $t \in \{l, \ldots, r\}$; a maximal length d for the substring, where d is an integer such that $0 < d \leq r - l + 1$; the sign of the balance $s \subseteq \{+1, -1\}$. $+1$ is used for searching for a $+k$-substring, -1 is used for searching for a $-k$-substring, $\{+1, -1\}$ is used for searching for both. It outputs a triple (i, j, σ) such that $l \leq i \leq t \leq j \leq r$, $j - i + 1 \leq d$, $f(x[i, j]) \in \{+k, -k\}$ and $\sigma = \text{sign}(f(x[i, j])) \in s$. The substring should be the leftmost one that contains t, i.e. there is no other minimal $x[i', j']$ such that $i' < i$, $t \in [i', j']$, $f(x[i', j']) = f(x[i, j])$. If no such substrings have been found, the algorithm returns NULL.

The second one is $\textsc{FindAtRightmost}_k$. It is similar to the $\textsc{FindAtLeftmost}_k$, but finds the rightmost $\pm k$-substring, i.e. there is no other minimal $x[i', j']$ such that $j' > j$, $t \in [i', j']$, $f(x[i', j']) = f(x[i, j])$

The third one is $\textsc{FindFirst}_k(l, r, s, \mathit{direction})$ and accepts as inputs: the borders l and r, where l and r are integers such that $0 \leq l \leq r \leq n-1$; the sign of the balance $s \subseteq \{+1, -1\}$. a $\mathit{direction} \in \{\mathit{left}, \mathit{right}\}$. When the direction is right (respectively left), $\textsc{FindFirst}_k$ finds the first $\pm k$-substring from the left to the right (respectively from the right to the left) in $[l, r]$ of sign s.

These three subroutines are interdependent since $\textsc{FindAtLeftmost}_k$ uses $\textsc{FindFirst}_{k-1}$ and $\textsc{FindAtRightmost}_{k-1}$ as subroutines, $\textsc{FindFirst}_k$ uses $\textsc{FindAtLeftmost}_k$ and $\textsc{FindAtRightmost}_k$ as subroutines. A description of $\textsc{FindAtLeftmost}_k(l, r, t, d, s)$ follows. The algorithm is presented in [4, Appendix A]. The description of the subroutine $\textsc{FindAtRightmost}_k(l, r, t, d, s)$ is similar and is omitted.

When $k = 2$, the procedure $\textsc{FindAtLeftmost}_2(l, r, t, d, s)$ checks that $x_t = x_{t-1}$ and $\text{sign}(f(x[t-1, t])) \in s$. If yes, it has found the substring. Otherwise, it checks if $x_t = x_{t+1}$ and $\text{sign}(f(x[t, t+1])) \in s$. If both checks fail, the procedure returns NULL. For $k > 2$ the procedure is the following.

Step 1. Check whether t is inside a $\pm(k-1)$-substring of length at most $d-1$, i.e. $v = (i, j, \sigma) \leftarrow \textsc{FindAtLeftmost}_{k-1}(l, r, t, d-1, \{+1, -1\})$. If $v \neq$ NULL, then $(i_1, j_1, \sigma_1) \leftarrow (i, j, \sigma)$ and the algorithm goes to Step 2. Otherwise, the algorithm goes to Step 6.

Step 2. Check whether $i_1 - 1$ is inside a $\pm(k-1)$-substring of length at most $d-1$ and choose the rightmost one: $v = (i, j, \sigma) \leftarrow \textsc{FindAtRightmost}_{k-1}(l, r, i_1 - 1, d-1, \{+1, -1\})$. If $v =$ NULL, then the algorithm goes to Step 3. If $v \neq$ NULL and $\sigma = \sigma_1$, then $(i_2, j_2, \sigma_2) \leftarrow (i, j, \sigma)$ and go to Step 8. Otherwise, go to Step 4.

Step 3. Search for the first $\pm(k-1)$-substring on the left from $i_1 - 1$ at distance at most d, i.e. $v = (i, j, \sigma) \leftarrow \textsc{FindFirst}_{k-1}(\min(l, j_1 - d + 1), i_1 - 1), \{+1, -1\}, \mathit{left})$. If $v \neq$ NULL and $\sigma_1 = \sigma$, then $(i_2, j_2, \sigma_2) \leftarrow (i, j, \sigma)$ and go to Step 8. Otherwise, go to Step 4.

Step 4. Check whether $j_1 + 1$ is inside a $\pm(k-1)$-substring of length at most $d-1$, i.e. $v = (i, j, \sigma) \leftarrow \textsc{FindAtLeftmost}_{k-1}(l, r, j_1 + 1, d-1, \{+1, -1\})$. If $v \neq$ NULL, then $(i_2, j_2, \sigma_2) \leftarrow (i, j, \sigma)$ and go to Step 8. Otherwise, go to Step 5.

Step 5. Search for the first $\pm(k-1)$-substring on the right from $j_1 + 1$ at distance at most d, i.e. $v = (i, j, \sigma) \leftarrow \textsc{FindFirst}_{k-1}(j_1 + 1, \min(i_1 + d - 1, r), \{+1, -1\}, \mathit{right})$. If $v \neq$ NULL, then $(i_2, j_2, \sigma_2) \leftarrow (i, j, \sigma)$, then go to Step 8. Otherwise, return NULL.

Step 6. Search for the first $\pm(k-1)$-substring on the right at distance at most d from t, i.e. $v = (i, j, \sigma) \leftarrow \textsc{FindFirst}_{k-1}(t, \min(t + d - 1, r), \{+1, -1\}, \mathit{right})$ If $v \neq$ NULL, then $(i_1, j_1, \sigma_1) \leftarrow (i, j, \sigma)$ and go to Step 7. Otherwise, returns NULL.

Step 7. Search for the first $\pm(k-1)$-substring on the left from t at distance at most d, i.e. $v = (i, j, \sigma) \leftarrow \textsc{FindFirst}_{k-1}(\max(l, t-d+1), t), \{+1, -1\}, left)$
If $v \neq$ NULL, then $(i_2, j_2, \sigma_2) \leftarrow (i, j, \sigma)$ and go to Step 8. Otherwise, returns NULL.

Step 8. If $\sigma_1 = \sigma_2$, $\sigma_1 \in s$ and $\max(j_1, j_2) - \min(i_1, i_2) + 1 \leq d$, the subroutine returns $(\min(i_1, i_2), \max(j_1, j_2), \sigma_1)$, otherwise returns NULL.

By construction and induction on k, the two $\pm(k-1)$-substrings $x[i_1, j_1]$ and $x[i_2, j_2]$ (if they exist) involved in the procedure $\textsc{FindAtLeftmost}_k$ are always consecutive and minimal. $\textsc{FindAtLeftmost}_k$ thus returns a $\pm k$-substring, if both substrings have the same sign.

Using this basic procedure, we then search for a $\pm k-$substring by searching for a t and d such that $\textsc{FindAtLeftmost}_k(l, r, t, d, s)$ returns a non-NULL value. Unfortunately, our algorithms have two-sided bounded error: they can, with small probability, return NULL even if a substring exists or return a wrong substring instead of NULL. In this setting, Grover's search algorithm is not directly applicable and we need to use a more sophisticated search [10]. Furthermore, simply applying the search algorithm naively does not give the right complexity. Indeed, if we search for a substring of length roughly d (say between d and $2d$), we can find one with expected running time $O(\sqrt{(r-l)/d})$ because at least d values of t will work. On the other hand, if there are no such substrings, the expected running time will be $O(\sqrt{r-l})$. Intuitively, we can do better because if there is a substring of length at least d then there are at least d values of t that work. Hence, we only need to distinguish between no solutions, or at least d. This allows to stop the Grover iteration early and make $O(\sqrt{(r-l)/d})$ queries in all cases.

▶ **Lemma 1** (Modified from [10], [4, Appendix B]). *Given n algorithms, quantum or classical, each computing some bit-value with bounded error probability, and some $T \geqslant 1$, there is a quantum algorithm that uses $O(\sqrt{n/T})$ queries and with constant probability: returns the index of a "1", if there are at least T "1s" among the n values; returns* NULL *if there are no "1"; returns anything otherwise.*

The algorithm that uses above ideas is presented in Algorithm 1.

Algorithm 1 $\textsc{FindFixedLen}_k(l, r, d, s)$. Search for any $\pm k$-substring of length $\in [d/2, d]$.

Find t such that $v_t \leftarrow \textsc{FindAtLeftmost}_k(l, r, t, d, s) \neq$ NULL using Lemma 1 with $T = d/2$.
return v_t or NULL if none.

We can then write an algorithm $\textsc{FindAny}_k(l, r, s)$ that searches for any $\pm k$-substring. We consider a randomized algorithm that uniformly chooses a of power 2 from $[2^{\lceil \log_2 k \rceil}, (r-l)]$, i.e. $d \in \{2^{\lceil \log_2 k \rceil}, 2^{\lceil \log_2 k \rceil + 1}, \ldots, 2^{\lceil \log_2 (r-l) \rceil}\}$. For the chosen d, we run Algorithm 1. So, the algorithm will succeed with probability at least $O(1/\log(r-l))$. We can apply Amplitude amplification and ideas from Lemma 1 to this and get an algorithm that uses $O(\sqrt{\log(r-l)})$ iterations.

Algorithm 2 $\textsc{FindAny}_k(l, r, s)$. Search for any $\pm k$-substring.

Find $d \in \{2^{\lceil \log_2 k \rceil}, 2^{\lceil \log_2 k \rceil + 1}, \ldots, 2^{\lceil \log_2 (r-l) \rceil}\}$ such that:
$v_d \leftarrow \textsc{FindFixedLen}_k(l, r, d, s) \neq$ NULL using amplitude amplification.
return v_d or NULL if none.

Finally, we present the algorithm that finds the first $\pm k$-substring – FindFirst$_k$. Let us consider the case $direction = right$. We first find the smallest segment from the left to the right such that its length w is a power of 2 and it contains a $\pm k$-substring. We do so by doubling the length of the segment until we find a $\pm k$-substring. We now have a segment that contains a $\pm k$-substring and we want to find the leftmost one. We do so by the following variant of binary search. At each step let $mid = \lfloor (lBorder + rBorder)/2 \rfloor$ be the middle of the search segment $[lBorder, rBorder]$. There are three cases:

- There is a k-substring in $[lBorder, mid]$, then the leftmost k-substring is in this segment.
- There are no k-substrings in $[lBorder, mid]$, but mid is inside a k-substring. Then the leftmost k-substring that contains mid is the required substring.
- There are no k-substrings in $[lBorder, mid]$ and mid is not inside a k-substring. Then the required substring is in $[mid + 1, rBorder]$.

Each iteration of the loop the algorithm halves the search space or finds the first k-substring itself if it contains mid. If $direction = left$, we replace FindAtLeftmost$_k$ by FindAtRightmost$_k$ that finds the rightmost $\pm k$-substring that containts mid. A detailed description of this algorithm is presented in [4, Appendix C].

▶ **Proposition 2.** *For any $\varepsilon > 0$ and k, algorithms FindAtLeftmost$_k$, FindFixedLen$_k$, FindAny$_k$ and FindFirst$_k$ have two-sided error probability $\varepsilon < 0.5$ and return, when correct:*

- *If t is inside a $\pm k-$substring of sign s of length up to d in $x[l, r]$, then FindAtLeftmost$_k$ will return such a substring, otherwise it returns* NULL. *The running time is $O(\sqrt{d}(\log(r-l))^{0.5(k-2)})$.*
- *FindFixedLen$_k$ either returns a $\pm k-$substring of sign s and length at most d in $x[l, r]$, or* NULL. *It is only guaranteed to return a substring if there exists $\pm k-$substring of length at least $d/2$, otherwise it can return* NULL. *The running time is $O(\sqrt{r-l}(\log(r-l))^{0.5(k-2)})$.*
- *FindAny$_k$ returns any $\pm k-$substring of sign s in $x[l, r]$, otherwise it returns* NULL. *The running time is $O(\sqrt{r-l}(\log(r-l))^{0.5(k-1)})$.*
- *FindFirst$_k$ returns the first $\pm k-$substring of sign s in $x[l, r]$ in the specified direction, otherwise it returns* NULL. *The running time is $O(\sqrt{r-l}(\log(r-l))^{0.5(k-1)})$.*

Proof. We prove the result by induction on k. The base case of $k = 2$ is obvious because of simplicity of FindAtLeftmost$_2$ and FindAtRightmost$_2$ procedures. We first prove the correctness of all the algorithms, assuming there are no errors. At the end we explain how to deal with the errors.

We start with FindAtLeftmost$_k$: there are different cases to be considered when searching for a $+k$-substring $x[i, j]$ of length $\leq d$.

1. Assume that there are j_1 and i_2 such that $i < j_1 < i_2 < j$, $|f(x[i, j_1])| = |f(x[i_2, j])| = k - 1$ and $\text{sign}(f(x[i, j_1])) = \text{sign}(f(x[i_2, j])) \in s$. If $t \in \{i_2, \ldots, j\}$, then the algorithm finds $x[i_2, j]$ in Step 1 and the first invocation of FindFirst$_{k-1}$ in Step 3 finds $x[i, j_1]$. If $t \in \{i, \ldots, j_1\}$, then the algorithm finds $x[i, j_1]$ in Step 1 and the second invocation of FindFirst$_{k-1}$ in Step 5 finds $x[i_2, j]$. If $j_1 < t < i_2$, then the third invocation of FindFirst$_{k-1}$ in Step 6 finds $x[i_2, j]$ and the forth invocation of FindFirst$_{k-1}$ in Step 7 finds $x[i, j_1]$.
2. Assume that there are j_1 and i_2 such that $i < i_2 < j_1 < j$, $|f(x[i, j_1])| = |f(x[i_2, j])| = k - 1$ and $\text{sign}(f(x[i, j_1])) = \text{sign}(f(x[i_2, j])) \in s$. If $t \in \{i, \ldots, j_1\}$, then the algorithm finds $x[i, j_1]$ in Step 1. After that, it finds $x[i_2, j]$ in Step 4. If $t \in \{j_1 + 1, \ldots, j\}$, then the algorithm finds $x[i_2, j]$ in Step 1. After that, it finds $x[i, j_1]$ in Step 2.

By induction, the running time of each FindAtLeftmost$_{k-1}$ invocation is $O(\sqrt{d}(\log(r-l))^{0.5(k-3)})$, and the running time of each FindFirst$_{k-1}$ invocation is $O(\sqrt{d}(\log(r-l))^{0.5(k-2)})$.

We now look at FindFixedLen$_k$: by construction and definition of FindAtLeftmost$_k$, if the algorithm returns a value, it is a valid substring (with high probability). If there exists a substring of length at least $d/2$, then any query to FindAtLeftmost$_k$ with a value of t in this interval will succeed, hence there are at least $d/2$ solutions. Therefore, by Lemma 1, the algorithm will find one with high probability and make $O\left(\sqrt{\frac{r-l}{d/2}}\right)$ queries. Each query has complexity $O(\sqrt{d}(\log(r-l))^{0.5(k-2)})$ by the previous paragraph, hence the running time is bounded by $O(\sqrt{r-l}(\log(r-l)^{0.5(k-2)})$.

We can now analyze FindAny$_k$: Assume that the shortest $\pm k$-substring $x[i,j]$ is of length $g = j - i + 1$. Therefore, there is a d such that $d \leq g \leq 2d$ and the FindFixedLen$_k$ procedure returns a substring for this d with constant success probability. So, the success probability of the randomized algorithm is at least $O(1/\log(l-r))$. Therefore, the amplitude amplification does $O(\sqrt{\log(r-l)})$ iterations. The running time of FindFixedLen$_k$ is $O(\sqrt{r-l}(\log(r-l))^{0.5(k-2)})$ by induction, hence the total running time is $O(\sqrt{r-l}(\log(r-l))^{0.5(k-2)}\sqrt{\log(l-r)}) = O(\sqrt{r-l}(\log(r-l))^{0.5(k-1)})$.

Finally, we analyze FindFirst$_k$: See [4, Appendix C].

We now turn to error analysis. The case of FindAtLeftmost$_k$ is easy: the algorithm makes at most 5 recursive calls, each having a success probability of $1-\varepsilon$. Hence it will succeed with probability $(1-\varepsilon)^5$. We can boost this probability to $1-\varepsilon$ by repeating this algorithm a constant number of times. Note that this constant depends on ε.

The analysis of FindFixedLen$_k$ follows from [10] and Lemma 1: since FindAtLeftmost$_k$ has two-sided error ε, there exists a search algorithm with two-sided error ε. ◀

3.2 The Algorithm for Dyck$_{k,n}$

To solve Dyck$_{k,n}$, we modify the input x. As the new input we use $x' = 1^k x 0^k$. Dyck$_{k,n}(x) = 1$ iff there are no $\pm(k+1)$-substrings in x'. This idea is presented in Algorithm 3.

Algorithm 3 Dyck$_{k,n}(x)$. The Quantum Algorithm for Dyck$_{k,n}$.

$x \leftarrow 1^k x 0^k$
$v = \text{FindAny}_{(k+1)}(0, n+2k-1, \{+1,-1\})$
return $v == \text{NULL}$

▶ **Theorem 3** ([4, Appendix D]). *Algorithm 3 solves Dyck$_{k,n}$ and the running time of Algorithm 3 is $O(\sqrt{n}(\log n)^{0.5k})$. The algorithm has two-side error probability $\varepsilon < 0.5$.*

4 Lower bounds for Dyck languages

▶ **Theorem 4.** *There exist constants $c_1, c_2 > 0$ such that $Q\left(Dyck_{c_1 \ell m, c_2 (2m)^\ell}\right) = \Omega\left(m^\ell\right)$.*

Proof. We will use the partial Boolean function $\text{Ex}_m^{a|b} = \begin{cases} 1, & \text{if } |x|_0 = a \\ 0, & \text{if } |x|_0 = b. \end{cases}$

We prove the theorem by a reduction $\left(\text{Ex}_{2m}^{m|m+1}\right)^\ell \leqslant \text{Dyck}_{c_1 \ell m, c_2 (2m)^\ell}$, with the reduction described in [4, Appendix E]. It is known that $Adv^{\pm}\left(\text{Ex}_{2m}^{m|m+1}\right) \geq Adv\left(\text{Ex}_{2m}^{m|m+1}\right) > m$

[2, Theorem 5.4]. The Adversary bound composes even for partial Boolean functions [11, Lemma 1], therefore $Q\left(\left(\mathrm{Ex}_{2m}^{m|m+1}\right)^{\ell}\right) = \Omega\left(m^{\ell}\right)$. Via the reduction the same bound applies to $\textsc{Dyck}_{c_1\ell m, c_2(2m)^{\ell}}$. ◀

▶ **Theorem 5.** *For any $\epsilon > 0$, there exists $c > 0$ such that $Q(\textit{DYCK}_{c\log n,n}) = \Omega\left(n^{1-\epsilon}\right)$.*

Proof. For any $\epsilon > 0$, there exists an m such that $Adv^{\pm}\left(\mathrm{Ex}_{2m}^{m|m+1}\right) \geq (2m)^{1-\epsilon}$. Without loss of generality we may assume that $(2m)^{\ell} = n$. From Theorem 4 with $\ell = \log_{2m} n$ we obtain $c_2(2m)^{\ell} = c_2 n$ and height $c_1 m\ell = \Theta(\log n)$. The query complexity is at least $\left((2m)^{1-\epsilon}\right)^{\ell} = \left((2m)^{\ell}\right)^{1-\epsilon} = n^{1-\epsilon}$. Therefore $Q(\textsc{Dyck}_{c\log n,n}) = \Omega\left(n^{1-\epsilon}\right)$. ◀

For constant depths the following bound can be derived:

▶ **Theorem 6.** *There exists a constant $c_1 > 0$ such that $Q(\textit{DYCK}_{c_1\ell,n}) = \Omega(2^{\frac{\ell}{2}}\sqrt{n})$.*

Proof. Let $m = 4$ in the Theorem 4. Then, $Q\left(\textsc{Dyck}_{c_1\ell,c_2 8^{\ell}}\right) = \Omega\left(4^{\ell}\right)$ for some constants $c_1, c_2 > 0$. Consider the function $\textsc{And}_{\frac{n}{c_2 8^{\ell}}} \circ \textsc{Dyck}_{c_1\ell,c_2 8^{\ell}}$ with a promise that $\textsc{And}_k$ has as an input either k or $k-1$ ones. Then,

$$Q\left(\textsc{And}_{\frac{n}{c_2 8^{\ell}}} \circ \textsc{Dyck}_{c_1\ell,c_2 8^{\ell}}\right) = \Theta\left(Adv^{\pm}\left(\textsc{And}_{\frac{n}{c_2 8^{\ell}}} \circ \textsc{Dyck}_{c_1\ell,c_2 8^{\ell}}\right)\right) \text{ and}$$

$$Adv^{\pm}\left(\textsc{And}_{\frac{n}{c_2 8^{\ell}}} \circ \textsc{Dyck}_{c_1\ell,c_2 8^{\ell}}\right) \geq Adv^{\pm}\left(\textsc{And}_{\frac{n}{c_2 8^{\ell}}}\right) Adv^{\pm}\left(\textsc{Dyck}_{c_1\ell,c_2 8^{\ell}}\right) = \Omega\left(2^{\frac{\ell}{2}}\sqrt{n}\right),$$

with the second step following from the composition of $Adv^{\pm}$ for partial functions [11]. This implies the same lower bound on $\textsc{Dyck}_{c_1\ell,n}$ because the computation of the composition $\textsc{And}_{\frac{n}{c_2 8^{\ell}}} \circ \textsc{Dyck}_{c_1\ell,c_2 8^{\ell}}$ can be straightforwardly reduced to $\textsc{Dyck}_{c_1\ell,n}$ by a simple concatenation of $\textsc{Dyck}_{c_1\ell,c_2 8^{\ell}}$ instances. ◀

5 Quantum complexity of st-Connectivity in grids

5.1 Quantum complexity of 2D-DConnectivity$_{n,k}$

▶ **Theorem 7.** *For any $n \geq k$ and $\epsilon > 0$, $Q(\textit{2D-DCONNECTIVITY}_{n,k}) = \Omega\left((\sqrt{n}k)^{1-\epsilon}\right)$.*

In particular, if we have a square grid then

▶ **Corollary 8.** *For any $\epsilon > 0$, $Q(\textit{2D-DCONNECTIVITY}_{n,n}) = \Omega\left(n^{1.5-\epsilon}\right)$.*

Proof of Theorem 7. For any sequence w of m opening and closing parentheses it is possible to plot the changes of depth, i.e., the number of opening parentheses minus the number of closing parentheses, for all prefixes of the sequence, see Figure 1.

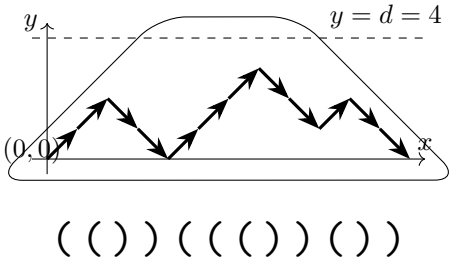

Figure 1 Representation of the Dyck word "(())((()))()".

We can connect neighboring points by vectors $(1,1)$ and $(1,-1)$ corresponding to opening and closing parentheses respectively. Clearly $w \in L_d$ if and only if the path starting at the origin $(0,0)$ ends at $(m,0)$ and never crosses $y=0$ and $y=d$. Consequently a path corresponding to $w \in L_d$ always remains within the trapezoid bounded by $y=0$, $y=d$, $y=x$, $y=-x+m$. This suggests a way of mapping $\textsc{Dyck}_{d,m}$ to the 2D-$\textsc{DConnectivity}_{n,k}$ problem:

1. An opening parenthesis in position i corresponds to a "column" of upwards sloping available edges $(i-1,l) \to (i,l+1)$ for all $l \in \{0,1,\ldots,d-1\}$ such that $i-1+l$ is even. A closing parenthesis in position i corresponds to downwards sloping available edges $(i-1,l) \to (i,l-1)$ for all $l \in \{1,\ldots,d\}$ such that $i-1+l$ is even. See Figure 2.
2. The edges outside the trapezoid adjacent to the trapezoid are forbidden (see Figure 3), i.e., it is sufficient to "insulate" the trapezoid by a single layer of forbidden edges. The only exception are the edges adjacent to the $(0,0)$ and $(m,0)$ vertex as those will be used in the construction (step 4).

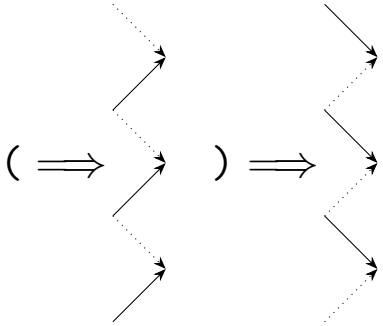

Figure 2 Mapping of $\textsc{Dyck}_{d,m}$ variables to 2D-$\textsc{DConnectivity}$.

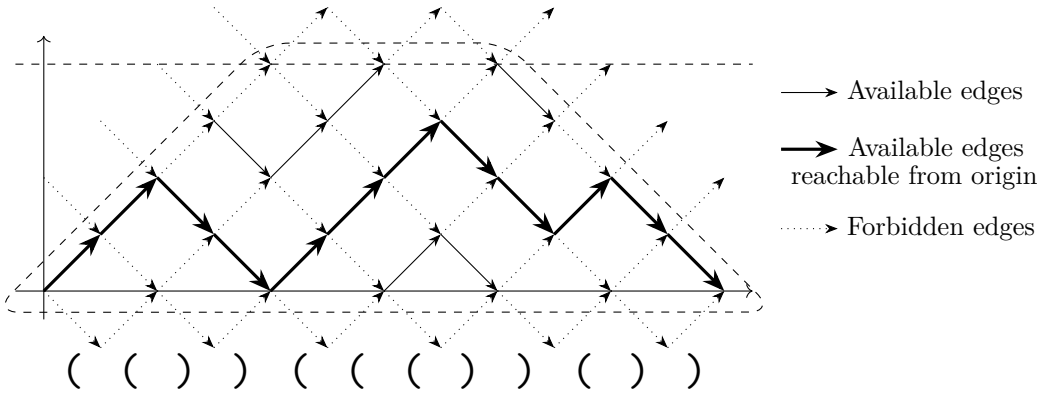

Figure 3 Mapping of a complete input corresponding to Dyck word "`(())((())())`" to 2D-$\textsc{DConnectivity}$.

3. Rotate the trapezoid by 45 degrees counterclockwise. This isolated trapezoid can be embedded in a directed grid and its starting and ending vertices are connected by a path if and only if the corresponding input word is valid.
4. Finally we can lay multiple independent trapezoids side by side and connect them in parallel forming an $\textsc{Or}_t$ of $\textsc{Dyck}_{d,m}$ instances; see Figure 4.

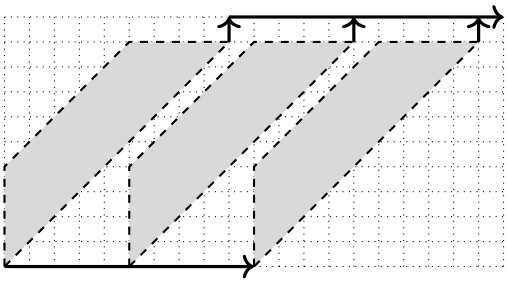

Figure 4 Reduction $\text{OR}_t \circ \text{DYCK} \leqslant$ 2D-DCONNECTIVITY

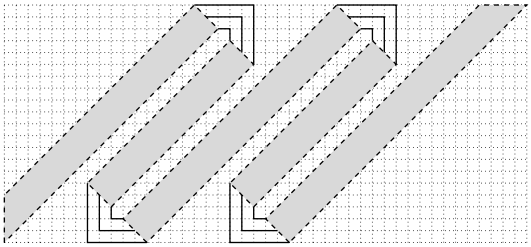

Figure 5 Folding of a long DYCK instance in an undirected grid

This concludes the reduction $\text{OR}_t \circ \text{DYCK}_{d,m} \leqslant$ 2D-DCONNECTIVITY$_{n,k}$, where $n = (d+1)(t-1) + \frac{m}{2} + 1$ and $k = \frac{m}{2} + 1$. By the well known composition result of Reichardt [12] we know that $Q(\text{OR}_t \circ \text{DYCK}_{d,m}) = \Theta(Q(\text{OR}_t) \cdot Q(\text{DYCK}_{d,m}))$. All that remains is to pick suitable t, d and m for the proof to be complete. Let k be the vertical dimension of the grid and $k \leq n$. Then we take $m = \Theta(k)$, $d = \log m$ and $t = \frac{n}{d}$. ◀

Constructing a non-trivial quantum algorithm appears to be difficult and we conjecture that the actual complexity may be $\Omega(nk)$, except for the case when k is small, compared to n. For very small k (up to $k = \Theta(\frac{\log n}{\log\log n})$), a better quantum algorithm is possible.

▶ **Theorem 9** ([4, Appendix F]). *$Q(2D\text{-}DC\text{ONNECTIVITY}_{n,k}) = O\left(\sqrt{n}\log_2^{k/2} n\right)$. Moreover, there is a time-efficient quantum query algorithm that solves 2D-DCONNECTIVITY$_{n,k}$ in time $O\left(\sqrt{n}\log_2^{k/2+O(1)} n\right)$.*

5.2 Lower bounds for 2D-Connectivity$_{n,k}$

Even though it is possible to use the construction from Section 5.1 to give a lower bound of $\Omega\big((\sqrt{n}k)^{1-\epsilon}\big)$ for the undirected case because the paths for each instance of DYCK never bifurcate or merge, this lower bound can be further improved to a nearly tight estimate.

▶ **Theorem 10.** *For any $n \geq k$, $k = \Omega(\log n)$, $\epsilon > 0$, $Q(2D\text{-}C\text{ONNECTIVITY}_{n,k}) = \Omega\big((nk)^{1-\epsilon}\big)$.*

Proof. We start off by representing an input as a path in a trapezoid, see Figure 3. But now instead of connecting multiple instances of DYCK in parallel we will embed one long instance by folding it when it hits the boundary of the graph. To implement a fold we will use simple gadgets depicted in Figure 5.

This way a DYCK instance of length m and depth $\log m$ can be embedded in an $n \times k$ grid such that $\frac{nk}{\log m} = \Theta(m)$. Using Theorem 5 we conclude that solving 2D-CONNECTIVITY$_{n,k}$ requires at least $\Omega\big((nk)^{1-\epsilon}\big)$ quantum queries. ◀

5.3 Lower bounds for d-dimensional grids

For undirected d-dimensional grids we give a tight bound on the number of queries required to solve connectivity.

▶ **Theorem 11.** *For any $\epsilon > 0$, for undirected d-dimensional grids of size $n_1 \times n_2 \times \ldots \times n_d$ that are not "almost-one-dimensional", i.e., there exists $i \in [d]$ such that $\frac{\prod_{j=1}^{d} n_j}{n_i} = \Omega(\log n_i)$:*

$$Q(\text{d}D\text{-}C\text{ONNECTIVITY}_{n_1,n_2,\ldots,n_d}) = \Omega((n_1 \cdot n_2 \cdot \ldots \cdot n_d)^{1-\epsilon}).$$

Proof. For the purposes of this theorem, it is more convenient to refer to $n_1 \times \ldots \times n_d$ sized grids as $n'_1 \times \ldots \times n'_d$ sized where $n'_i = n_i + 1$. Then the theorem follows from the 2D case by iteratively using the fact that a d-dimensional grid of size $n'_1 \times n'_2 \times \ldots \times n'_{d-1} \times n'_d$ contains as a subgraph a $(d-1)$-dimensional grid of size $n'_1 \times n'_2 \times \ldots \times n'_{d-2} \times n'_{d-1}n'_d$. One way to see this is to consider a bijective mapping of the vertices $(x_1, \ldots, x_{d-1}, x_d)$ to $(x_1, \ldots, x_{d-2}, x_d n'_{d-1} + x_{d-1})$ if x_d is even and to $(x_1, \ldots, x_{d-2}, x_d n'_{d-1} + n'_{d-1} - 1 - x_{d-1})$ if x_d is odd. It is a bijection because x_d and x_{d-1} can be recovered from $x_d n'_{d-1} + n'_{d-1} - 1 - x_{d-1}$ by computing the quotient and remainder on division by n'_{d-1}. One can view this procedure as "folding" where we take layers (vertices corresponding to some $x_d = l$) and fold them into the $(d-1)$-st dimension alternating the direction of the layers depending on the parity of the layer l. For this procedure to place the starting and ending vertices the furthest apart, it requires that n'_d is an odd number. Otherwise we embed a smaller subgraph $n'_1 \times \ldots \times n'_{d-1} \times (n'_d - 1)$ and add an edge $(n_1, \ldots, n_{d-1}, n_d - 1)$ to $(n_1, \ldots, n_{d-1}, n_d)$. In the end we obtain a lower bound of $\Omega(((((((n'_d - 1)n'_{d-1} - 1)n'_{d-2} - 1)\ldots)n'_2 - 1)n'_1)^{1-\epsilon}) = \Omega((n_1 \cdot n_2 \cdot \ldots \cdot n_d)^{1-\epsilon})$. ◀

For directed d-dimensional grids we can only slightly improve over the $n^{\frac{d}{2}}$ trivial lower bound.

▶ **Theorem 12.** *For directed d-dimensional grids of size $n_1 \times n_2 \times \ldots \times n_d$ such that $n_1 \leq n_2 \leq \ldots \leq n_d$ and $\epsilon > 0$, $Q(\text{d}D\text{-DC}\textsc{onnectivity}_{n_1,n_2,\ldots,n_d}) = \Omega((n_{d-1}\prod_{i=1}^{d} n_i)^{\frac{1}{2}-\epsilon})$.*

▶ **Corollary 13.** *For directed d-dimensional grids of size $n \times n \times \ldots \times n$ and $\epsilon > 0$, $Q(\text{d}D\text{-DC}\textsc{onnectivity}_{n,n,\ldots,n}) = \Omega(n^{\frac{d+1}{2}-\epsilon})$.*

Proof of Theorem 12. For each $I \in \{0, 1, \ldots, n_1\} \times \{0, 1, \ldots, n_1\} \times \ldots \times \{0, 1, \ldots, n_{d-2}\}$ we take take a 2-dimensional hard instance G_I of 2D-DC$\textsc{onnectivity}_{n_{d-1},n_d}$ having query complexity $\Omega(n_{d-1}^{1-\epsilon} n_d^{\frac{1}{2}-\epsilon})$. We then connect them in parallel like so:

- Include the entire $(d-2)$-dimensional subgrid from $(0, \ldots, 0)$ to $(n_1, n_2, \ldots, n_{d-2}, 0, 0)$ and similarly the subgrid from $(0, 0, \ldots, 0, n_{d-1}, n_d)$ to $(n_1, n_2, \ldots, n_{d-2}, n_{d-1}, n_d)$;
- For each $I \in \{0, 1, \ldots, n_1\} \times \{0, 1, \ldots, n_1\} \times \ldots \times \{0, 1, \ldots, n_{d-2}\}$ embed the instance G_I in the subgrid $(I, 0, 0)$ to (I, n_{d-1}, n_d);
- Forbid all other edges.

This construction computes $\textsc{Or}_{\prod_{i=1}^{d-2}(n_i+1)} \circ$ 2D-DC$\textsc{onnectivity}_{n_{d-1},n_d}$ whose complexity is at least $\Omega(\sqrt{\prod_{i=1}^{d-2}(n_i+1)}\, n_{d-1}^{1-\epsilon} n_d^{\frac{1}{2}-\epsilon}) = \Omega((n_{d-1}\prod_{i=1}^{d} n_i)^{\frac{1}{2}-\epsilon})$. ◀

6 Directions for future works

Some directions for future work are:

1. **Better algorithm/lower bound for the directed 2D grid?** Can we find an $o(n^2)$ query quantum algorithm or improve our lower bound? A nontrivial quantum algorithm would be particularly interesting, as it may imply a quantum algorithm for edit distance.
2. **Quantum algorithms for directed connectivity?** More generally, can we come up with better quantum algorithms for directed connectivity? The span program method used by Belovs and Reichardt [6] for the undirected connectivity does not work in the directed case. As a result, the quantum algorithms for directed connectivity are typically based on Grover's search in various forms, from simply speeding up depth-first/breadth-first search to more sophisticated approaches [3]. Developing other methods for directed connectivity would be very interesting.

3. **Quantum speedups for dynamic programming.** Dynamic programming is a widely used algorithmic method for classical algorithms and it would be very interesting to speed it up quantumly. This has been the motivating question for both the connectivity problem on the directed 2D grid studied in this paper and a similar problem for the Boolean hypercube in [3] motivated by algoritms for Travelling Salesman Problem. There are many more dynamic programming algorithms and exploring quantum speedups of them would be quite interesting.

References

1 Scott Aaronson, Daniel Grier, and Luke Schaeffer. A quantum query complexity trichotomy for regular languages. *Electronic Colloquium on Computational Complexity (ECCC)*, 26:61, 2018.

2 Andris Ambainis. Quantum lower bounds by quantum arguments. *Journal of Computer and System Sciences*, 64(4):750–767, 2002.

3 Andris Ambainis, Kaspars Balodis, Janis Iraids, Martins Kokainis, Krisjanis Prusis, and Jevgenijs Vihrovs. Quantum speedups for exponential-time dynamic programming algorithms. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1783–1793, 2019. doi:10.1137/1.9781611975482.107.

4 Andris Ambainis, Kaspars Balodis, Jānis Iraids, Kamil Khadiev, Vladislavs Kļevickis, Krišjānis Prūsis, Yixin Shen, Juris Smotrovs, and Jevgēnijs Vihrovs. Quantum lower and upper bounds for 2d-grid and dyck language, 2020. arXiv:2007.03402.

5 Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 51–58. ACM, 2015.

6 Aleksandrs Belovs and Ben W. Reichardt. Span programs and quantum algorithms for st-connectivity and claw detection. In *Algorithms - ESA 2012 - 20th Annual European Symposium, Ljubljana, Slovenia, September 10-12, 2012. Proceedings*, pages 193–204, 2012. doi:10.1007/978-3-642-33090-2_18.

7 Mahdi Boroujeni, Soheil Ehsani, Mohammad Ghodsi, MohammadTaghi HajiAghayi, and Saeed Seddighin. Approximating edit distance in truly subquadratic time: Quantum and MapReduce. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1170–1189. SIAM, 2018.

8 Harry Buhrman, Subhasree Patro, and Florian Speelman. The quantum strong exponential-time hypothesis, 2019. arXiv:1911.05686.

9 Diptarka Chakraborty, Debarati Das, Elazar Goldenberg, Michal Koucký, and Michael E. Saks. Approximating edit distance within constant factor in truly sub-quadratic time. In *59th Annual IEEE Symposium on Foundations of Computer Science (FOCS), Paris, France, Oct 7-9, 2018*, pages 979–990, 2018. arXiv:1810.03664.

10 Peter Høyer, Michele Mosca, and Ronald de Wolf. Quantum search on bounded-error inputs. In Jos C. M. Baeten, Jan Karel Lenstra, Joachim Parrow, and Gerhard J. Woeginger, editors, *Automata, Languages and Programming*, pages 291–299, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

11 Shelby Kimmel. Quantum adversary (upper) bound. In *International Colloquium on Automata, Languages, and Programming*, pages 557–568. Springer, 2012.

12 Ben W. Reichardt. Reflections for quantum query algorithms. In *Proceedings of the Twenty-second Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '11, pages 560–569, Philadelphia, PA, USA, 2011. Society for Industrial and Applied Mathematics. URL: http://dl.acm.org/citation.cfm?id=2133036.2133080.

13 Robert A Wagner and Michael J Fischer. The string-to-string correction problem. *Journal of the ACM (JACM)*, 21(1):168–173, 1974.