# Approximating Edit Distance in Truly Subquadratic Time: Quantum and MapReduce

MAHDI BOROUJENI, Sharif University of Technology, Iran
SOHEIL EHSANI, University of Maryland, USA
MOHAMMAD GHODSI, Sharif University of Technology, Iran, and Institute for Research in Fundamental Sciences (IPM), Iran
MOHAMMADTAGHI HAJIAGHAYI, University of Maryland, USA
SAEED SEDDIGHIN, Toyota Technological Institute at Chicago, USA

The *edit distance* between two strings is defined as the smallest number of *insertions*, *deletions*, and *substitutions* that need to be made to transform one of the strings to another one. Approximating edit distance in subquadratic time is "one of the biggest unsolved problems in the field of combinatorial pattern matching" [37]. Our main result is a quantum constant approximation algorithm for computing the edit distance in truly subquadratic time. More precisely, we give an $O(n^{1.810})$ quantum algorithm that approximates the edit distance within a factor of 3. We further extend this result to an $O(n^{1.708})$ quantum algorithm that approximates the edit distance within a larger constant factor.

Our solutions are based on a framework for approximating edit distance in parallel settings. This framework requires as black box an algorithm that computes the distances of several smaller strings all at once. For a quantum algorithm, we reduce the black box to *metric estimation* and provide efficient algorithms for approximating it. We further show that this framework enables us to approximate edit distance in distributed settings. To this end, we provide a MapReduce algorithm to approximate edit distance within a factor of $1+\epsilon$, with sublinearly many machines and sublinear memory. Also, our algorithm runs in a logarithmic number of rounds.

CCS Concepts: • **Computing methodologies** → *MapReduce algorithms*; *Massively parallel algorithms*; • **Theory of computation** → *Quantum computation theory*; **Approximation algorithms analysis**; **Pattern matching**; Dynamic programming;

Additional Key Words and Phrases: Edit distance, approximation algorithm, subquadratic time algorithm, quantum algorithm, parallel algorithm, MapReduce



Portions of this research were completed while the Soheil Ehsani, MohammadTaghi HajiAghayi, and Saeed Seddighin were visitors at the Simons Institute for the Theory of Computing.
A preliminary version of this article was presented at *SODA 2018* [15].
Supported in part by NSF CAREER award CCF-1053605, NSF BIGDATA grant IIS-1546108, NSF AF:Medium grant CCF-1161365, DARPA GRAPHS/AFOSR grant FA9550-12-1-0423, and another DARPA SIMPLEX grant.
Authors' addresses: M. Boroujeni, Sharif University of Technology, Azadi St., Tehran, Iran; email: safarnejad@ce.sharif.edu; S. Ehsani and M. HajiAghayi, University of Maryland, College Park, Maryland; emails: {ehsani, hajiagha}@cs.umd.edu; M. Ghodsi, Sharif University of Technology, Azadi St., Tehran, Iran, Institute for Research in Fundamental Sciences (IPM), Rocquencourt, Tehran, Iran; email: ghodsi@sharif.edu; S. Seddighin, Toyota Technological Institute at Chicago, Chicago, Illinois; email: saeedreza.seddighin@ttic.edu.

0004-5411/2021/05-ART19 $15.00
https://doi.org/10.1145/3456807

**ACM Reference format:**
Mahdi Boroujeni, Soheil Ehsani, Mohammad Ghodsi, MohammadTaghi HajiAghayi, and Saeed Seddighin. 2021. Approximating Edit Distance in Truly Subquadratic Time: Quantum and MapReduce. *J. ACM* 68, 3, Article 19 (May 2021), 41 pages.
https://doi.org/10.1145/3456807

## 1 INTRODUCTION

The *edit distance* (a.k.a *Levenshtein distance*) is a well-known metric to measure the similarity of two strings. This metric has been extensively used in several fields such as computational biology, natural language processing, and information theory. The algorithmic aspect of the problem is even more fundamental; the problem of computing the edit distance is a textbook example for dynamic programming. Therefore, edit distance has been subject to a plethora of studies in the past few decades (see e.g. [2–5, 8–12, 24, 35, 37, 39, 43, 45, 48, 53, 57]).

The edit distance between two strings is defined as the smallest number of *insertions*, *deletions*, and *substitutions* that need to be made on one of the strings to transform it to another one. For two strings $s_1$ and $s_2$ with $n$ characters in total ($|s_1| + |s_2| = n$), a classic dynamic program finds the edit distance between them in time $O(n^2)$. The idea is to define auxiliary variables $d_{i,j}$s, which denote the edit distance between the first $i$ characters of $s_1$ and the first $j$ characters of $s_2$. Next, we iteratively determine the values of the auxiliary variables based on the following formula:

$$d_{i,j} = \begin{cases} d_{i-1,j-1} & \text{if } s_1[i] = s_2[j], \\ 1 + \min\{d_{i-1,j-1}, d_{i,j-1}, d_{i-1,j}\} & \text{if } s_1[i] \neq s_2[j]. \end{cases}$$

Despite the simplicity of the above solution, it has remained one of the most efficient algorithms from a theoretical standpoint to this day. Since the 1970s, several researchers aimed to improve the quadratic running time of the problem; however, thus far, the best-known algorithm runs in time $O(n^2/\log^2 n)$ [48]. The shortcoming of these studies is partly addressed by the work of Backurs and Indyk [10], wherein the authors show that a truly subquadratic time algorithm is impossible to achieve unless a widely believed conjecture (SETH[1]) fails.

Unfortunately, the quadratic dependency of the running time on the size of the input makes it impossible to use such algorithms for large inputs in practice. For example, a human genome consists of almost 3 billion base pairs that need to be incorporated in similarity measurements. Therefore, several studies were focused on improving the running time of the algorithm by considering approximation solutions. A trivial $\sqrt{n}$ approximation algorithm follows from an $O(n + d^2)$ exact algorithm of Landau et al. [45], where $d$ is the edit distance between the two strings. Subsequent research improved this to $n^{3/7}$ [11], to $n^{1/3+o(1)}$ [12], to $2^{\widetilde{O}(\sqrt{\log n})}$ [8], and to the latest of which provides a polylogarithmic approximation guarantee in subquadratic time [5]. Note that although the running times of these algorithms are almost linear, even if one favors the approximation factor over the running time, slowing down the algorithms to barely subquadratic doesn't yield an asymptotically better approximation guarantee. Despite persistent studies, finding a subquadratic algorithm with a constant approximation factor, which is the "holy grail" here, is still open (see Section 6 of Indyk [37]). You can see an overview of recent works in this field in Section 1.1.

Quantum computation provides a strong framework to substantially improve the running time of many algorithmic problems. This includes a long list of problems from algebraic computational problems, to measuring graph properties, to string matching, to searching, to optimizing programs,

[1]The *strong exponential time hypothesis* states that no algorithm can solve the satisfiability problem in time $2^{n(1-\epsilon)}$.

etc. [13, 14, 28, 35, 38, 44, 47, 56]. However, quantum techniques can only be applied to limited structures. For instance, many classic problems such as sorting or even counting the number of 1s in a 0-1 array are still as time-consuming even with quantum computation. Indeed, existing quantum techniques offer no immediate improvement to the running time of edit distance. Likewise, no improved quantum algorithm is known for many classic DP-type problems such as finding the lcs (longest common subsequence) or dtw (dynamic time warping) of two strings, or determining the Fréchet distance between two polylines. To the best of our knowledge, no exact or approximation algorithm is known for edit distance in subquadratic time via quantum computation.

In this work, we provide a framework to approximate the edit distance between two strings within a constant factor. This framework requires as black box a procedure that takes several smaller strings as input and approximates their distances all at once. For quantum computers, we reduce this black box to finding the distances of a metric, namely *metric estimation*. In this problem, we are given a metric space where any distance is available by a query from a distance oracle. We show that metric estimation cannot be approximated within a factor better than 3 with a subquadratic number of quantum queries. On the contrary, we provide positive results for approximation factor 3 and also larger constant factors. We show our bounds are tight up to constant factors by proving lower bounds on the query complexity of metric estimation. Our metric estimation quantum algorithms are general tools and may find their applications in other distance-related problems as well. Combining this black box with our framework yields subquadratic quantum algorithms for approximation edit distance within a constant factor. Our work is similar in spirit to the work of Le Gall [47] and Dürr et al. [25], where combinatorial techniques are used to obtain efficient quantum algorithms. We believe that our work opens an avenue to further investigation of edit distance in the quantum setting and perhaps achieving a near-linear-time quantum algorithm for edit distance.

As another application of our framework, we design a MapReduce algorithm for approximating edit distance within an approximation factor of $1 + \epsilon$. MapReduce is one of the most recent developments in the area of parallel computing. It has the benefits of both sequential and parallel computation. Many tech companies such as Google, Facebook, Amazon, and Yahoo designed MapReduce frameworks and have used them to implement fast algorithms to analyze their data. In this article, we focus on the well-known MapReduce theoretical framework initiated by Karloff et al. [40] (and later further refined by Andoni et al. [6]). Our algorithm runs in a logarithmic number of rounds with a sublinear number of machines and sublinear memory of each machine. Moreover, the running time of each machine is subquadratic.

To the best of our knowledge, both our quantum algorithms and our MapReduce algorithm are the first to improve upon the trivial $O(n^2)$ classical algorithm beyond subpolynomial factors for approximating edit distance[2] in these settings. We believe that our framework can be useful to better understand edit distance in other models, such as the streaming and the semi-streaming models.

### 1.1 Related Work

*Prior Work.* The closest works to our results are [4] and [8]. In particular, they use a space embedding approach from [53] with dividing the string into blocks of smaller size, but our main observations and structural lemmas are completely different from their approach. We note that to the best of our knowledge, the ideas of our framework are novel and have not been used in any of the previous work. Moreover, our combinatorial ideas have an advantage over previous approaches

[2]within a constant factor.

including [8], since these approaches have at least $\log n$ levels of recursion, and therefore having a logarithmic approximation factor is their inherent bottleneck.

A similar approach is taken in the work of Nayebi and Williams [52], wherein the authors study the computational complexity of APSP on quantum computers. They give an APSP algorithm for graph instances with small integer weights. They also give a fine-grained reduction from APSP to negative triangle via quantum computing.

In [9], the authors give a parallel algorithm for determining the edit distance between two strings. Their algorithm uses $\widetilde{O}(n^2)$ processors and a shared memory of $O(n^2)$. Note that their algorithm cannot be used in MapReduce models, since the number of machines and memory of each machine in a MapReduce algorithm should be sublinear, and the number of rounds should be $O(\text{polylog}(n))$ [40]. The major advantage of our MapReduce algorithm over the algorithm of [9] is that both the number of machines and the memory of each machine is sublinear in our algorithm. Moreover, the number of rounds in our algorithm is $O(\log(n))$.

Designing MapReduce algorithms for simulating sequential dynamic programs for important problems was recently initiated by Im et al. [36]. They study DP-type problems with two key properties, monotonicity and decomposability. Their framework does not apply here since edit distance is neither monotone nor decomposable.

*Subsequent Work.* Following the publication of the conference version of this article [15], a number of follow-up works pursued our approach.

Many subsequent works use the framework of this article to approximate edit distance and other string similarity problems. The most notable work is done by Chakraborty et al. [22, 23], in which they present the first constant approximation algorithm with truly subquadratic running time for edit distance. They use the same framework as our article, cover input strings with smaller windows, and then use triangle inequality to approximate the edit distance between many pairs of windows. Hence, two out of three steps of their algorithm are similar to ours [55]. However, they replace the third step of our framework, which is done by Grover's search here, by a novel randomized approach, which makes the algorithm entirely classical. A series of works then improve the running time of the algorithm. Koucký and Saks [42] and Brakensiek and Rubinstein [20] independently provide a near-linear-time constant-factor approximation algorithm for edit distance where the input strings are far from each other. Finally, Andoni and Nosatzki [7] present a similar near-linear-time algorithm that does not impose any condition on input strings. Note that the approximation factors of these near-linear-time algorithms are (exponentially or doubly exponentially) large constants. The best algorithm in terms of running time that approximates edit distance within a factor of $3 + \epsilon$ is presented by Goldenberg et al. [30] with a running time of $\widetilde{O}(n^{1.6+o(1)})$.

Furthermore, several works customize the framework of this article to solve other problems [19, 26, 29, 32, 41, 49]. Notably, Rubinstein et al. [54] modify the framework of this article and use an extension of triangle inequality that is applicable to non-metric settings. By doing so, they provide approximation algorithms for LCS and LIS. Moreover, Boroujeni et al. [16] replaced the triangle inequality by a randomized technique and give $1+o(1)$ approximation algorithms for edit distance and longest common subsequence when one of the input strings is randomly perturbed.

Moreover, HajiAghayi et al. [34] substantially improve our massively parallel (MapReduce) algorithm for edit distance in terms of approximation factor, round complexity, number of machines, and total running time. Roughly speaking, we construct a set of (not necessarily disjoint) windows for both strings and show that by computing the edit distances of the windows, one can approximate the solution. Our algorithm uses matrix multiplication in the (max, +) setting, which is time-consuming and requires $O(\log n)$ rounds of computation. In contrast, HajiAghayi et al. [34] take a rather different approach and break only one of the strings into disjoint blocks of size $n^{1-x}$. Then,

based on the properties of edit distance, they seek to find appropriate intervals of the other string that are potential matches for each block. Once the candidate intervals are selected, they show that one can solve the problem in a single round. Thus, their algorithm only needs two rounds of computation. Furthermore, they present a massively parallel algorithm for LCS. In addition, Boroujeni and Seddighin [17] improve the running time and the memory of this algorithm and give the first parallel constant-factor approximation algorithm for edit distance that has a truly subquadratic total running time. Their main contribution is parallelizing the framework of this article.

## 2 OUR RESULTS AND TECHNIQUES

In this section, we explain the ideas and techniques of our framework and show how we obtain a subquadratic algorithm for approximating the edit distance on quantum computers. The basis of our MapReduce algorithm is similar to what we explain here, though some details are modified to run the algorithm in a logarithmic number of MapReduce rounds. More details about the MapReduce algorithm can be found in Section 6. Our quantum algorithm is based on several known techniques of quantum computing, algorithm design, and approximation algorithms. On the quantum side, we take advantage of *Grover's search* [31] and *amplitude amplification* [21] to improve the lookup time on an unordered set. On the algorithmic side, we benefit from classical algorithmic tools such as dynamic programming techniques, divide and conquer, and randomized techniques. In addition to this, we leverage *the bootstrapping technique* to further improve the running time of our algorithm, by allowing the approximation guarantee to grow to larger constant numbers.

Recall that the edit distance between two strings is defined as the smallest number of insertions, deletions, and substitutions that one needs to perform on one of the strings to obtain the other one. For two strings $s_1$ and $s_2$, we denote their edit distance by $\text{edit}(s_1, s_2)$. By definition, edit distance meets all of the *identity of indiscernibles,*[3] *symmetry,*[4] and *triangle inequality*[5] properties, and thus for any set of strings $\mathcal{M}$, $\langle \mathcal{M}, \text{edit} \rangle$ forms a metric space.[6] Following this intuition, our algorithm is closely related to the study of the metric spaces.

In the following, we outline our algorithm in three steps. First, we define an auxiliary problem, namely *metric estimation,* and present efficient approximation algorithms for this problem accompanied by tight bounds on its quantum complexity. Roughly speaking, in this problem, we are given a metric space with $n$ points and oracle access to the distances, and the goal is to output an $n \times n$ matrix that is an estimate to the distances between the points. One may think of the oracle as an ordinary computer program that we then convert to the corresponding quantum code and unitary operator using a quantum compiler [27]. We give two approximation algorithms that solve the metric estimation problem with approximation factors $3 + \epsilon$ and $\mathsf{e}_{\mathsf{m}}(\epsilon) = O(1/\epsilon)$ with $\widetilde{O}(n^{5/3}\text{poly}(1/\epsilon))$ and $\widetilde{O}(n^{3/2+\epsilon}\text{poly}(1/\epsilon))$ oracle queries, respectively. Notice that the running times of the algorithms are $O(n^2\text{poly}(1/\epsilon))$, but the query complexities are subquadratic. This allows us to approximate metric spaces with sublinear points for which answering an oracle query is time-consuming. We also consider $\Delta$-neighboring metric estimation where we output the distance of two points only if their indices differ by at most $\Delta$, where $\Delta < n$ is a positive integer. Similar to to metric estimation, we give two approximation algorithms that solve $\Delta$-neighboring metric estimation with the same approximation factors $3+\epsilon$ and $\mathsf{e}_{\mathsf{m}}(\epsilon) = O(1/\epsilon)$ with $\widetilde{O}(n\Delta^{2/3}\text{poly}(1/\epsilon))$ and $\widetilde{O}(n\Delta^{1/2+\epsilon}\text{poly}(1/\epsilon))$ oracle queries, respectively. Moreover, the running times of the algorithms are $O(n\Delta\text{poly}(1/\epsilon))$. We emphasize that our metric estimation results are general and can be used

[3] $\text{edit}(s_1, s_2) = 0 \Leftrightarrow s_1 = s_2$.
[4] $\text{edit}(s_1, s_2) = \text{edit}(s_2, s_1)$.
[5] $\text{edit}(s_1, s_2) + \text{edit}(s_2, s_3) \geq \text{edit}(s_1, s_3)$.
[6] A set of points $\mathcal{M}$ and a distance function $d$ form a metric space $\langle \mathcal{M}, d \rangle$, if $d$ meets all of the aforementioned properties.

for any metric. In the second step, we show that any algorithm that solves the metric estimation problem within an approximation factor $\alpha$ can be used as a black box to obtain an $\alpha + \epsilon$ approximation solution for edit distance. As we show, the reduction takes a subquadratic time and thus, using our $3 + \epsilon$ approximation algorithm for metric estimation, we obtain a $3 + \epsilon$ approximation algorithm for edit distance. Finally, in Section 5, we devise a bootstrapping technique to further improve the running time of the algorithm by taking a hit on the approximation guarantee. In what follows, we explain each of the steps in more detail. Before we delve into the algorithm, we would like to note some comments.

- The only step of the algorithm where quantum computation plays a role is the first step where we discuss metric estimation. Nevertheless, everywhere we use the term *algorithm*, we mean a quantum algorithm unless otherwise stated.
- In this section, we explain the abstract ideas and steps of the algorithm. Therefore, sometimes we do not provide formal proofs for some of the arguments that we make. The reader can find a detailed discussion of all statements and proofs in Sections 3, 4, 5, and 6.
- Everywhere we use the word *operation*, we refer to insertion, deletion, or substitution.

### 2.1 Metric Estimation

As mentioned earlier, in the metric estimation problem, we are given a metric space $\langle \mathcal{M}, d \rangle$ and an oracle $O$ that reports $d(x, y)$ for two points $x$ and $y$ in an invocation. The goal of the problem is to estimate the distance matrix of the points with as few oracle calls as possible. Due to the impossibility results for exact or even solutions with small approximation factors for this problem (see the rest for more details), our aim is to find an approximation solution.

| Metric Estimation |
| --- |
| Input: a metric space $\langle \mathcal{M}, d \rangle$ with $n$ points where $\mathcal{M} = \{p_1, p_2, \ldots, p_n\}$ and an oracle function $O$ to access the distances. |
| Guarantee: all the distances are integer numbers in the interval $[0, u]$. We assume $u$ is $O(\text{poly}(n))$. |
| An output (with approximation factor $\alpha > 1$): an $n \times n$ matrix $A$, where $d(p_i, p_j) \leq A[i][j] \leq \alpha d(p_i, p_j)$ holds for every $1 \leq i, j \leq n$. |

Before we state the main ideas and results, we briefly explain two key tools that we borrow from previous work and use as black boxes in our algorithms. The first tool is the seminal work of Grover [31] for making fast searches in an unordered database. Suppose we are given a function $f : [n] \to \{0, 1\}$, where $[n] = \{1, 2, 3, \ldots, n\}$, and we wish to list up to $m$ distinct indices for which the value of the function is equal to 1. We refer to this problem as *element listing*.

| Element Listing |
| --- |
| Input: integers $n$ and $0 < m \leq n$, and access to an oracle that, upon receiving an integer $i$, reports the value of $f(i)$. $f$ is defined over $[n]$ and maps each index to either 0 or 1. |
| Output: a list of up to $m$ indices for which the value of $f$ is equal to 1. If the total number of such indices is not more than $m$, the output should contain all of them. |

The pioneering work of Grover [31] implies that the element listing problem can be solved with only $O(\sqrt{nm})$ oracle calls via quantum computation. We subsequently make use of this algorithm in this section.

Theorem 2.1 (Proven in [18]). *The listing problem can be solved with $O(\sqrt{nm})$ oracle queries via quantum computation.*

The second quantum technique that we use in this article is a tool for proving lower bounds on the quantum complexity of the problems. Let $f : [n] \to \{-1, 1\}$ be a function defined over the numbers $1, 2, \ldots, n$ that maps each index to either $-1$ or 1 and $\text{par}(f) = \prod_{i\in[n]} f(i)$. In the parity problem, we are given oracle access to $f$ and the goal is to determine $\text{par}(f)$ with as few oracle calls as possible.

| Parity |
| --- |
| Input: an integer $n$, and access to an oracle $O$ that upon receiving an integer $i$ reports the value of $f(i)$. $f$ is defined over $[n]$ and maps each index to either $-1$ or 1. |
| Output: $\text{par}(f) = \prod_{i\in[n]} f(i)$. |

Of course, if the numbers of $-1$s or 1s are substantially smaller than $n$ (i.e., $o(n)$), one can use Grover's search to list all of such indices and compute the parity with fewer than $\Omega(n)$ oracle calls. However, if this is not the case for either $-1$ or 1, such an approach fails. The seminal work of Farhi et al. [27] showed that at least $\Omega(n)$ queries are necessary for solving the parity problem and thus quantum computation offers no speedup in this case.

Theorem 2.2 (Proven in [27]). *The parity problem cannot be solved with fewer than $\Omega(n)$ queries with quantum computation.*

Based on the result of Farhi et al. [27], we begin with showing an impossibility result. Our first result for metric estimation is a hardness of approximation for factors smaller than 3 using a subquadratic number of queries. More precisely, in Section 3, we show that any quantum algorithm that approximates metric estimation within a factor smaller than 3 needs to make at least $\Omega(n^2)$ oracle queries.

Theorem 3.1 [Restated]. *Any quantum algorithm for solving the metric estimation problem with an approximation factor smaller than 3 needs to make at least $\Omega(n^2)$ oracle calls.*

The idea is to show a reduction from parity to metric estimation. Suppose we are given an instance I of the parity problem. Roughly speaking, we construct an instance Cor(I) of the metric estimation and prove that Cor(I) has a valid metric as input. Next, we show that any algorithm that approximates metric estimation within a factor smaller than 3 with $o(n^2)$ queries can be turned into a quantum algorithm for solving parity with $o(n)$ queries, which is impossible due to Farhi et al. [27].

Despite this hardness of approximation for factors better than 3, we show the problem is significantly more tractable when we allow the approximation guarantee to be slightly more than 3. In Section 3, we show that for any $\epsilon > 0$, a $3 + \epsilon$ approximation of metric estimation is possible via $\widetilde{O}(n^{5/3}\text{poly}(1/\epsilon))$ queries.

Theorem 3.3 [Restated]. *For any $\epsilon > 0$, there exists a quantum algorithm that solves metric estimation with $\widetilde{O}(n^{5/3}\text{poly}(1/\epsilon))$ queries within an approximation factor of $3 + \epsilon$. Moreover, the running time of the algorithm is $\widetilde{O}(n^2\text{poly}(1/\epsilon))$.*

Our first take on the solution is to discretize the problem at the expense of imposing an additional $1+\epsilon$ factor to our guarantee. Notice that all of the distances of the metric are non-negative integers no more than $u$. Therefore, one can divide the distances into $\log_{1+\epsilon/3} u = \widetilde{O}(\text{poly}(1/\epsilon))$ disjoint intervals where the distances within each interval differ in at most a multiplicative factor of $1+\epsilon/3$. For every interval $[x, (1 + \epsilon/3)x]$ we can set a threshold $t = (1 + \epsilon/3)x$ and find all pairs within

a distance of at most $t$ with an approximation factor of 3. Then, based on all these solutions, one can find a $3+\epsilon$ approximation distance for every pair of the points.

Now the problem boils down to the following: given a threshold $t$, find all pairs $(p_i, p_j)$ such that $d(p_i, p_j) \leq t$. Of course, an exact solution for this problem is hopeless due to our impossibility result. Therefore, we allow some false positive in our solution as well. More precisely, we restrict our solution to contain all pairs $(p_i, p_j)$ such that $(p_i, p_j) \leq d$, but additional pairs are also allowed to appear, if $(p_i, p_j) \leq 3d$. It is easy to show that any solution that solves the above problem via $\widetilde{O}(n^{5/3}\text{poly}(1/\epsilon))$ queries yields a $3+\epsilon$ approximation factor algorithm for metric estimation that uses at most $\widetilde{O}(n^{5/3}\text{poly}(1/\epsilon))$ oracle calls.

In what follows, we describe the ideas to solve the problem for a fixed threshold $t$. The algorithm is explained in detail in Section 3; therefore, here we just mention the tools and techniques. For convenience, we construct a graph $G$ with $n$ nodes and correspond every point $p_i$ of the metric to a vertex $v_i$ of the graph. For a pair of points $(p_i, p_j)$, we add an undirected edge $(v_i, v_j)$ to the graph, if $d(p_i, p_j) \leq t$. Notice that the oracle function $O$ provides us the exact value of $d(p_i, p_j)$ for any $p_i$ and $p_j$; therefore, we can examine whether an edge exists between two vertices $v_i, v_j$ with a single oracle call. Recall that Grover's search allows us to find as many as $m$ elements with value 1 of a function of size $n$ via $O(\sqrt{nm})$ oracle calls. Therefore, if the number of the edges of the graph is $O(n^{4/3})$, we can use Grover's search (Theorem 2.1) to list all of the edges with $O(\sqrt{n^2 \cdot n^{4/3}}) = O(n^{5/3})$ queries and solve the problem. Therefore, the non-trivial part of the problem is the case where the graph is dense. In this case, the average degree of the vertices is at least $\Omega(n^{1/3})$. Now, suppose we select a vertex $v_i$ whose degree is at least $n^{1/3}$, and with $n-1$ query calls, find the distances of its corresponding point $p_i$ from all other points of the metric. Let set $D^t$ be the set of all points that have a distance of at most $t$ from $p_i$ and $D^{2t}$ be the points with a distance of at most $2t$ from $p_i$. Trivially, $D^t \subseteq D^{2t}$. Due to the triangle inequality, all of the edges incident to the vertices corresponding to set $D^t$ are from the vertices corresponding to $D^{2t}$. Moreover, the distances of all points of $D^t$ from points of $D^{2t}$ are bounded by $3t$. Therefore, one can report all such pairs in the solution and proceed by removing $D^t$ from the graph (however, some vertices of $D^{2t}$ remain in the graph). Thus, all that remains is to solve the problem for an instance with at most $n - n^{1/3}$ nodes recursively. Since we make at most $O(n)$ query calls for every $n^{1/3}$ vertices (an amortized of $n^{2/3}$ per vertex), the total number of queries is $O(n^{5/3})$. More details about this can be found in Section 3.

In addition to Theorem 3.3, we show in Section 3 that with a deeper analysis, one can use the same ideas to further improve the query complexity to $\widetilde{O}(n^{3/2+\epsilon}\text{poly}(1/\epsilon))$ by allowing the approximation guarantee to grow up to $\mathsf{e}_{\mathsf{m}}(\epsilon) = O(1/\epsilon)$.

Theorem 3.5 [Restated]. *For any $\epsilon > 0$, there exists a quantum algorithm that solves metric estimation with $\widetilde{O}(n^{3/2+\epsilon}\text{poly}(1/\epsilon))$ queries within an approximation factor of $\mathsf{e}_{\mathsf{m}}(\epsilon) = O(1/\epsilon)$. Moreover, the running time of the algorithm is $\widetilde{O}(n^2\text{poly}(1/\epsilon))$.*

Moreover, we show that the query complexity of any algorithm that approximates metric estimation within a constant factor is at least $\Omega(n^{3/2})$. We prove this lower bound using a technique of Ambainis [1].

Theorem 3.8 [Restated]. *Any quantum algorithm that estimates distances of a metric space of $n$ points with a constant approximation factor has a query complexity of at least $\Omega(n^{3/2})$.*

Furthermore, we define $\Delta$-neighboring metric estimation. In this problem, for a given $\Delta$, we output an estimate of $d(p_i, p_j)$ only if $|i-j| \leq \Delta$. Therefore, instead of outputting the whole distance matrix, we only output a diagonal band of the distance matrix (i.e., the main diagonal and $\Delta$ diagonals on either side). A matrix $A$ where $A[i][j] = 0$ if $|i-j| > \Delta$ is called a band matrix with a bandwidth of $\Delta$.

Table 1. Quality of the Approximation Algorithms for Metric Estimation

| Approx. Factor | $\alpha < 3$ | $\alpha = 3 + \epsilon$ | $\alpha = \mathrm{e_m}(\epsilon)$ | $\alpha =$ Any Constant |
|---|---|---|---|---|
| # queries without $\Delta$ | $\Omega(n^2)$ (Theorem 3.1) | $\widetilde{O}(n^{5/3}\mathrm{poly}(1/\epsilon))$ (Theorem 3.3) | $\widetilde{O}(n^{3/2+\epsilon}\mathrm{poly}(1/\epsilon))$ (Theorem 3.5) | $\Omega(n^{3/2})$ (Theorem 3.8) |
| # queries with $\Delta$ | $\Omega(n\Delta)$ (Corollary 3.9) | $\widetilde{O}(n\Delta^{2/3}\mathrm{poly}(1/\epsilon))$ (Corollary 3.11) | $\widetilde{O}(n\Delta^{1/2+\epsilon}\mathrm{poly}(1/\epsilon))$ (Corollary 3.13) | $\Omega(n\Delta^{1/2})$ (Corollary 3.14) |

$\Delta$-Neighboring Metric Estimation

Input: a metric space $\langle \mathcal{M}, d\rangle$ with $n$ points where $\mathcal{M} = \{p_1, p_2, \ldots, p_n\}$, an oracle function $O$ to access the distances, and a positive integer $\Delta \leq n$.

Guarantee: all the distances are integer numbers in the interval $[0, u]$. We assume $u$ is $O(\mathrm{poly}(n))$.

An output (with approximation factor $\alpha > 1$): an $n \times n$ band matrix $A$ with bandwidth $\Delta$, where $d(p_i, p_j) \leq A[i][j] \leq \alpha d(p_i, p_j)$ holds for every $1 \leq i, j \leq n$ where $|i - j| \leq \Delta$.

We prove the following four results for $\Delta$-neighboring metric estimation using similar techniques to that of metric estimation.

Corollary 3.9 [Restated]. *Any quantum algorithm for solving the $\Delta$ metric estimation problem with an approximation factor smaller than 3 needs to make at least $\Omega(n\Delta)$ oracle calls.*

Corollary 3.11 [Restated]. *For any $\epsilon > 0$, there exists a quantum algorithm that solves parameterized metric estimation with $\widetilde{O}(n\Delta^{2/3}\mathrm{poly}(1/\epsilon))$ queries within an approximation factor of $3 + \epsilon$. Moreover, the running time of the algorithm is $\widetilde{O}(n\Delta\mathrm{poly}(1/\epsilon))$.*

Corollary 3.13 [Restated]. *For any $\epsilon > 0$, there exists a quantum algorithm that solves metric estimation with $\widetilde{O}(n\Delta^{1/2+\epsilon}\mathrm{poly}(1/\epsilon))$ queries within an approximation factor of $\mathrm{e_m}(\epsilon) = O(1/\epsilon)$. Moreover, the running time of the algorithm is $\widetilde{O}(n\Delta\mathrm{poly}(1/\epsilon))$.*

Corollary 3.14 [Restated]. *Any quantum algorithm that estimates distances of a metric space of $n$ points with a constant approximation factor has a query complexity of at least $\Omega(n\Delta^{1/2})$.*

You can find a summary of the results explained in this section in Table 1.

### 2.2 Approximating Edit Distance within a Factor $3 + \epsilon$

In the second step, we provide an algorithm to approximate the edit distance between two strings in subquadratic time, based on a reduction to metric estimation. Our approach here is twofold. Suppose we are given a *guess* $d$ on the actual edit distance between the strings, and we want to find an approximation proof to the guess. More precisely, we wish to find out whether $d$ is smaller than the actual distance of the strings or report a transformation of the strings with at most $\alpha d$ operations,[7] where $\alpha$ is given as an approximation factor. If $d$ is substantially smaller than $n$, then the $O(n + d^2)$ exact algorithm of Landau et al. [45] solves the problem in subquadratic time. Therefore, the only hard instances of the problem are when $d$ is asymptotically close to $n$. Therefore, we define a subtask of the edit distance problem, in which we are given two strings $s_1$

[7] insertion, deletion, or substitution.

and $s_2$ and guaranteed that the edit distance between the strings is at most $\delta(|s_1| + |s_2|)$, where $\delta$ is not too small. The goal is to find a transformation of the strings with at most $(\delta \cdot \alpha)(|s_1| + |s_2|)$ operations, where $\alpha$ is the approximation factor of the algorithm. We refer to this subtask of edit distance as *the $\delta$-bounded edit distance* problem.

| $\delta$-bounded edit distance |
|---|
| Input: two strings $s_1$ and $s_2$, and a real number $0 \leq \delta \leq 1$. |
| Guarantee: $\text{edit}(s_1, s_2) \leq \delta(|s_1| + |s_2|)$. |
| Output (with an approximation factor $\alpha > 1$): a sequence of operations with size at most $(\delta \cdot \alpha)(|s_1| + |s_2|)$ that transforms $s_1$ into $s_2$. |

We combine a divide-and-conquer technique with dynamic programming in order to approximate $\delta$-bounded edit distance. In addition to this, we subsequently make use of the quantum techniques mentioned earlier in our solution. Recall that the total number of characters in the input is equal to $n$, i.e., $|s_1| + |s_2| = n$. For clarity, we define three parameters $0 < \beta < 1$, $\mu > 0$, and $\gamma > 1$. $\gamma$ is an integer number, but $\beta$ and $\mu$ are real numbers. We use $\beta$, $\mu$, and $\gamma$ as three parameters of our algorithm, and after the analysis, we show which values for $\beta$, $\mu$, and $\gamma$ give us the best guarantee.

We begin by defining the notion of a *window* and construct a set of windows for each string. Let $l = \lfloor n^{1-\beta} \rfloor$ be the *maximum window size* and define a window of $s_1$ as a substring of $s_1$. Moreover, define $g = \lfloor l/\gamma \rfloor = O(n^{1-\beta}/\gamma)$ as *the gap size* and construct a collection $W_1$ of windows for $s_1$ as follows: First, for every $0 \leq i \leq \lfloor \frac{|s_1|-l}{g} \rfloor$, put a window $s_1[ig + 1, ig + l]$ (i.e., a window from index $ig + 1$ to index $ig + l$ of $s_1$) in $W_1$. In other words, $W_1$ contains tentatively $\gamma(|s_1|/l) = O(\gamma n^{\beta})$ windows of length $l$ where the gap between the neighboring windows is equal to $g$. Second, for every $0 \leq i \leq \lfloor \frac{|s_1|-l}{g} \rfloor$ and every $0 < j < \log_{1+\mu} l$, we add a window $s_1[ig + 1, ig + l - (1 + \mu)^j]$. The total number of windows in $W_1$ is $(1 + \log_{1+\mu} l) \cdot O(\gamma n^{\beta}) = \widetilde{O}((1/\mu)\gamma n^{\beta})$. Figure 1 illustrates how the windows of $W_1$ span over the characters of $s_1$. Notice that some of the windows overlap.

Similar to this, we construct a collection $W_2$ of windows for $s_2$ using the same parameters $l$ and $g$. We define a *transformation* of $s_1$ into $s_2$ as a sequence of insertions, deletions, and substitutions that turns $s_1$ into $s_2$. After a transformation of $s_1$ into $s_2$, we call a character of $s_2$ *old* if it either is substituted by a character of $s_1$ or remained intact during the transformation. In other words, if a character is not inserted during a transformation, it is called old. Based on this, we define the notion of *a window-compatible transformation* as follows:

*Definition 2.3.* Let $S = \langle w_1, w_2, \ldots, w_k \rangle$ and $S' = \langle w'_1, w'_2, \ldots, w'_k \rangle$ be two sequences of size $k$ of non-overlapping windows from $W_1$ and $W_2$, respectively. We call a transformation of $s_1$ into $s_2$ window compatible with respect to $S$ and $S'$ if (1) all old characters of $s_2$ are in the windows of $S'$ and (2) every old character of $s_2$ that is in a window $w'_i$ was placed in window $w_i$ of $s_1$ prior to the transformation. We call a transformation window compatible if it is window compatible with respect to at least a pair of sequences of non-overlapping windows **from** $W_1$ and $W_2$, respectively.

Intuitively, a window-compatible transformation with respect to two sequences of windows $S$ and $S'$ does not allow the characters to move in between the windows; if a character is initially placed in a window $w_i$, it should be either deleted or placed in window $w'_i$ of $s_2$ and vice versa. We emphasize that in order for a transformation to be window compatible, the corresponding windows should be selected from $W_1$ and $W_2$, respectively. A few examples of window-compatible and window-incompatible transformations are illustrated in Figure 2.

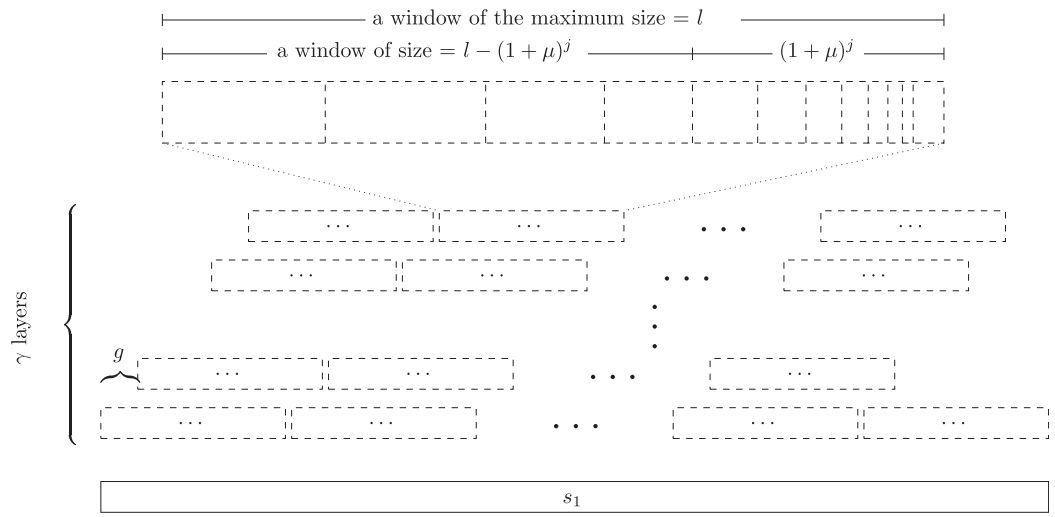
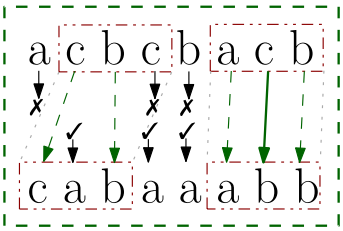
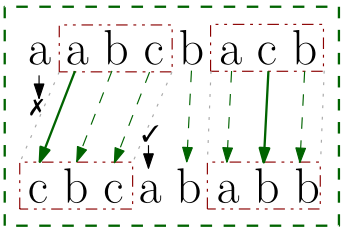


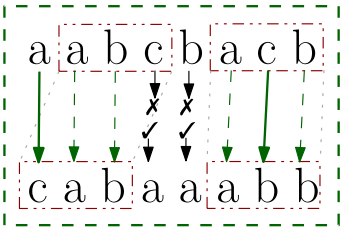

Fig. 1. $s_1$ is shown with a solid rectangle and $l$-sized windows of $W_1$ are depicted via dashed rectangles. Next to each window of size $l$, we have $\log_{1+\epsilon} l$ windows of smaller sizes, which is shown for one window at the top of the figure.



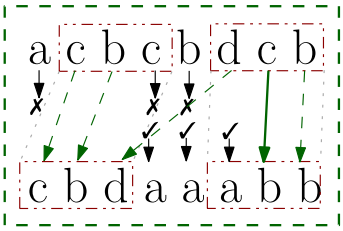

(a) An example of a window-compatible transformation.




(b) The transformation is not window compatible since character 5 of the second string is old, but doesn't lie in any windows.




(c) The transformation is not window compatible since character 1 of the second string is old, but prior to the transformation, it was not placed in any windows.




(d) The transformation is not window compatible since character 3 of the second string is old, but prior to the transformation, it was not placed in the corresponding window.

Fig. 2. Figure 2(a), (b), (c), and (d) show a few examples of window-compatible and window-incompatible transformations. Solid arrows show substitutions, dashed arrows show the characters that remain in the string, and other characters are either inserted or deleted.

As we show in the following, window-compatible transformations are well structured. In fact, we show in Section 4 that if the edit distances of the windows are accessible in time $O(1)$, a dynamic program can find an optimal[8] window-compatible transformation of $s_1$ into $s_2$ in time $O(n + \Delta \cdot \min(|W_1|, |W_2|))$. Here, $\Delta$ is the maximum difference between the indices of two windows that match to each other.

Lemma 4.1 [Restated]. *Given a matrix of edit distances between the substrings corresponding to every pair of windows of $W_1$ and $W_2$, and a positive integer $\Delta$, one can compute an optimal window-compatible transformation of $s_1$ into $s_2$ where a window in $s_1$ can map to a window in $s_2$ only if their indices do not differ by more than $\Delta$ in time $O(n + \Delta \cdot \min(|W_1|, |W_2|))$.*

Lemma 4.1 shows that window-compatible transformations are easy to find. It also follows from Lemma 4.1 that any $\alpha$ approximation matrix for the edit distances of the windows suffices to find an approximately optimal window-compatible transformation (with the same approximation factor) in time $O(n+\Delta\cdot\min(|W_1|, |W_2|))$. This makes the connection of edit distance and metric estimation more clear.

We complement this observation by a structural proof. In Section 4, we show that the length of the shortest window-compatible transformation of $s_1$ into $s_2$ is not far from $\delta(|s_1| + |s_2|)$. This enables us to use the previously mentioned algorithms to find an approximately optimal window-compatible transformation and show this is in fact a constant approximation away from $\delta(|s_1| + |s_2|)$. Moreover, in cases in which the solution is small, a window in $s_1$ can map to a window in $s_2$ only if their indices do not differ by more than $\widetilde{O}((1/\mu)\delta n/g)$.

Lemma 4.3 [Restated]. *Given that $\mathrm{edit}(s_1, s_2) \leq \delta n$, there exists a window-compatible transformation of $s_1$ into $s_2$ with at most $((1 + 2\mu)\delta + 1/\gamma)n + 2l$ operations. Moreover, for each window $w_i \in W_1$ that matches to a window $w_2 \in W_2$, their indices do not differ by more than $\Delta = \widetilde{O}((1/\mu)\delta n/g)$.*

Now we can put things in perspective. Lemma 4.1, in light of the results of metric estimation, provides us a nice tool for finding an approximately optimal window-compatible transformation, and Lemma 4.3 argues that such a transformation is to some extent optimal. Based on this, we outline our algorithm for $\delta$-bounded edit distance as follows:

(1) Construct the windows of $W_1$ and $W_2$ for both $s_1$ and $s_2$.
(2) Construct a metric $\langle \mathcal{M}, \mathrm{edit}\rangle$, where $\mathcal{M} = W_1 \cup W_2$ and the distance of two points in $\mathcal{M}$ is equal to the edit distance between their corresponding windows. We use the classic algorithm of edit distance to answer every oracle invocation for reporting the edit distance between two windows. Using the quantum approximation algorithm of $\Delta$-neighboring metric estimation, find a $3 + \epsilon$ approximation solution to the edit distances for every pair of windows (Corollary 3.11).
(3) Based on the estimated distances, find a $3 + \epsilon$ approximately optimal window-compatible transformation (Lemma 4.1).
(4) Report the transformation as an approximation proof for the $\delta$-bounded edit distance problem.

We show in Section 4 that by setting $\beta = 6/7$, $\mu = \epsilon/5$, and $\gamma = 1/\epsilon\delta$, the above algorithm runs in time $\widetilde{O}(n^{2-4/21}\ \mathrm{poly}(1/\epsilon))$ and has an approximation factor of $3 + \epsilon$.

Lemma 4.4 [Restated]. *There exists a quantum algorithm that solves the $\delta$-bounded edit distance problem within an approximation factor of $3 + \epsilon$ in time $\widetilde{O}(n^{2-4/21}\mathrm{poly}(1/\epsilon))$.*

[8] a transformation with the smallest number of operations.

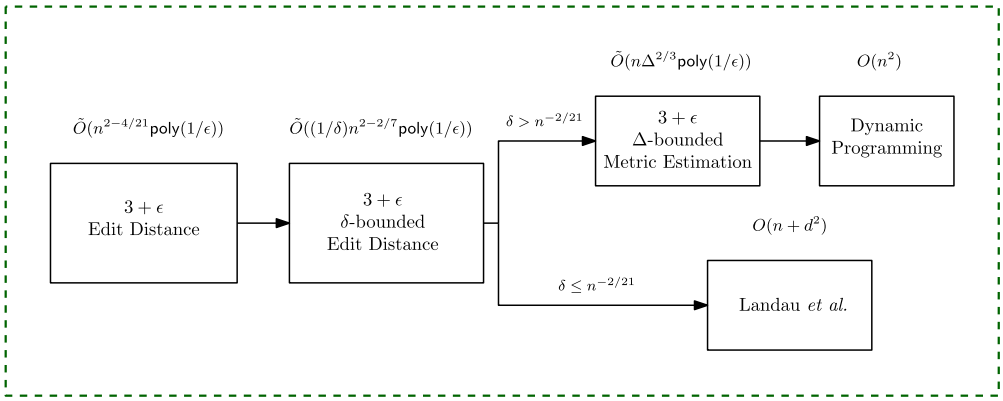



Fig. 3. The diagram depicts the components of the $3+\epsilon$ algorithm for edit distance. $x \rightarrow y$ shows that component $x$ uses component $y$ as a black box.

By Lemma 4.4, we can approximate the $\delta$-bounded edit distance problem in truly subquadratic time in case the guarantee holds. Of course, if this algorithm provides a larger or invalid transformation, one can immediately imply that the guarantee $\text{edit}(s_1, s_2) \leq \delta(|s_1| + |s_2|)$ is violated. The rest of the solution for edit distance follows from a simple multiplicative method. In order to solve edit distance, we first check whether the two strings are equal, and in that case, we report that their distance is equal to 0. Otherwise, $\text{edit}(s_1, s_2) \geq 1$. Now, we start with $\rho = 1/n$ and every time run our solution for $\delta$-bounded edit distance with parameter $\delta = \rho$ to find an approximation proof for $\text{edit}(s_1, s_2) = \rho n$. If our algorithm finds a proper transformation with at most $(3\rho + \epsilon)n$ operations, then we report that solution. Otherwise, we know that $\text{edit}(s_1, s_2) > \rho n$ and thus multiply $\rho$ by a factor $1+\epsilon$. Of course, this comes at the expense of an additional multiplicative factor of $1+\epsilon$ to the approximation factor; however, the running time remains $\widetilde{O}(n^{2-4/21}\text{poly}(1/\epsilon))$. We later refer to this technique as *guess and multiply*.

Theorem 4.5 [Restated]. *There exists a quantum algorithm that solves edit distance within an approximation factor of* $3+\epsilon$ *in time* $\widetilde{O}(n^{2-4/21}\text{poly}(1/\epsilon))$.

### 2.3 Improving the Running Time via Bootstrapping

So far, we have discussed how to use divide and conquer and metric estimation to approximate edit distance in subquadratic time. In this section, we explain the ideas to improve the running time of the algorithm by taking a hit on its approximation factor.

Recall that, in order to approximate the edit distance, we first construct a set of windows. Next, we use metric estimation to estimate the edit distances of the windows, and finally, we use a dynamic programming algorithm to find an almost optimal window-compatible transformation. As discussed before, such a solution approximates the edit distance within a constant factor. The components of this algorithm are illustrated in Figure 3.

Now, we show that we can improve the algorithm at two points. First, instead of using the $3+\epsilon$ approximation algorithm for metric estimation, we can lose a factor of $\mathsf{e}_{\mathsf{m}}(\epsilon)$ in the approximation and estimate the distances in time $\widetilde{O}(n\Delta^{1/2+\epsilon}\text{poly}(1/\epsilon))$ (Corollary 3.13). In addition to this, as an oracle function for metric estimation, we do not really need to compute the exact edit distances of the windows; a constant estimation to the distances suffices. Therefore, one can use our algorithm for approximating edit distance to implement the oracle in subquadratic time. Of course, this again comes at the expense of deteriorating the approximation guarantee, but the running time improves.

In this section, we show how we combine these ideas to achieve an $\widetilde{O}(n^{2-(5-\sqrt{17})/3+\epsilon}\text{poly}(1/\epsilon)) \simeq \widetilde{O}(n^{1.708})$-time algorithm. As to why the exponent converges to $2-(5-\sqrt{17})/3$, we refer the reader to a discussion in Section 5.

To formalize the above ideas, suppose we are given two strings $s_1$ and $s_2$ and would like to approximate the edit distance between the strings in time $\widetilde{O}(n^{2-(5-\sqrt{17})/3+\epsilon}\text{poly}(1/\epsilon))$. We call our algorithm for this problem $\mathcal{A}(\epsilon)$ and refer to its time complexity and approximation factor with $\text{t}_\text{e}(\epsilon)$ and $\text{e}_\text{e}(\epsilon)$, respectively. We inductively show that

$$\text{t}_\text{e}(\epsilon) = \widetilde{O}\left(n^{2-(5-\sqrt{17})/3+\epsilon}\text{poly}(1/\epsilon)\right)$$

and $\text{e}_\text{e}(\epsilon) = O(1/\epsilon)^{O(\log 1/\epsilon)}$. Notice that if $2-(5-\sqrt{17})/3+\epsilon \geq 2$, $\mathcal{A}(\epsilon)$ can be trivially implemented with the classic $O(n^2)$ algorithm and the approximation factor $\text{e}_\text{e}(\epsilon) = 1$. Now, assume that $2-(5-\sqrt{17})/3+\epsilon < 2$.

An $\widetilde{O}((1/\delta)n^{2-(5-\sqrt{17})/2+3\epsilon/2}\text{poly}(1/\epsilon))$-time algorithm for $\delta$-bounded edit distance suffices to design $\mathcal{A}(\epsilon)$. If $\delta \leq n^{-(5-\sqrt{17})/6+\epsilon/2}$, we run the $O(n+\delta^2 n^2)$ algorithm of Landau et al. [45]; otherwise, the running time of our algorithm is $\widetilde{O}(n^{2-(5-\sqrt{17})/3+\epsilon}\text{poly}(1/\epsilon))$. Moreover, a similar guess-and-multiply method explained in Section 2.2 extends this solution to edit distance. Therefore, all we need is to approximate the $\delta$-bounded edit distance problem in time $\widetilde{O}((1/\delta)n^{2-(5-\sqrt{17})/2+3\epsilon/2}\text{poly}(1/\epsilon))$. To this end, we again define three parameters $\beta$, $\mu$, and $\gamma$ and set the window size equal to $\lfloor n^{1-\beta}\rfloor$, $\mu$ to $\epsilon/5$, and the gap size equal to $g = \lfloor l/\gamma\rfloor$. Similar to what we explained before, we construct two sets of windows $W_1$ and $W_2$ for $s_1$ and $s_2$ based on the windows size and gap size. Now, we use the same algorithm for finding the edit distance between $s_1$ and $s_2$, with two modifications.

(1) Construct the windows of $W_1$ and $W_2$ for both $s_1$ and $s_2$.
(2) Construct a metric $\langle \mathcal{M}, \text{edit}\rangle$, where $\mathcal{M} = W_1 \cup W_2$ and the distance of two points in $\mathcal{M}$ is equal to the edit distance between their corresponding windows. We use $\mathcal{A}(2\epsilon)$ (a slightly slower version of our algorithm) for estimating the edit distances of the windows in time $\text{t}_\text{e}(2\epsilon) = \widetilde{O}(n^{2-(5-\sqrt{17})/3+2\epsilon}\text{poly}(1/\epsilon))$ as on oracle function. Using the approximation algorithm of metric estimation, find an $\text{e}_\text{m}(\epsilon)\text{e}_\text{e}(2\epsilon)$ approximation solution to the edit distances for every pair of windows (Theorem 3.5).
(3) Based on the estimated distances, find an $\text{e}_\text{m}(\epsilon)\text{e}_\text{e}(2\epsilon)$ approximately optimal window-compatible transformation (Lemma 4.1).
(4) Report the transformation as an approximation proof for the $\delta$-bounded edit distance problem.

Notice that there are two modifications to the previous algorithm. First, instead of using the $3+\epsilon$ factor algorithm for $\Delta$-neighboring metric estimation, here, we use an $\text{e}_\text{m}(\epsilon)$ approximation factor algorithm that runs in time $\widetilde{O}(n\Delta^{1/2+\epsilon}\text{poly}(1/\epsilon))$. Moreover, instead of implementing the oracle function via the classic $O(n^2)$ algorithm, we use $\mathcal{A}(2\epsilon)$ for approximating the edit distances. In Section 5, we show that by setting the right values for parameters $\beta$, $\mu$, and $\gamma$, the running time and approximation factor of algorithm $\mathcal{A}(\epsilon)$ would be $\widetilde{O}(n^{2-(5-\sqrt{17})/3+\epsilon}\text{poly}(1/\epsilon))$ and $\text{e}_\text{e}(\epsilon) = O(1/\epsilon)^{O(\log 1/\epsilon)}$, respectively.

Theorem 5.1 [Restated]. There exists an $\widetilde{O}(n^{2-(5-\sqrt{17})/3+\epsilon})$-time quantum algorithm that approximates edit distance within a factor $\text{e}_\text{e}(\epsilon) = O(1/\epsilon)^{O(\log 1/\epsilon)}$.

Figure 4 shows the components of $\mathcal{A}(\epsilon)$.

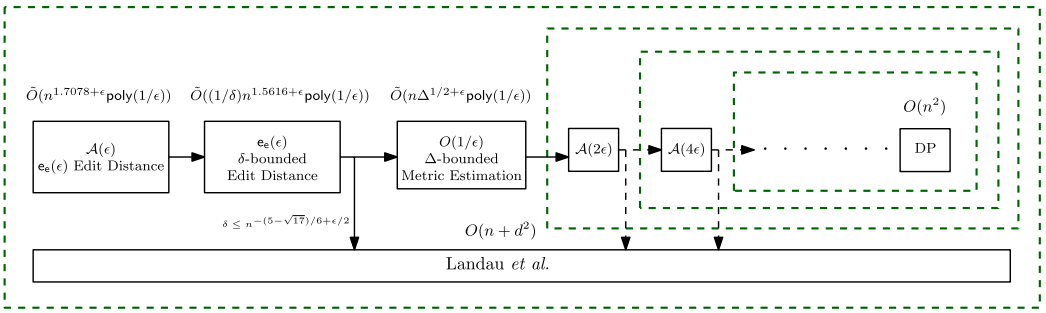



Fig. 4. The diagram illustrates the bootstrapping technique to achieve an $\widetilde{O}(n^{1.708})$-time quantum algorithm for approximating edit distance. $x \to y$ shows that component $x$ uses component $y$ as a black box.

## 3 METRIC ESTIMATION

In this section, we discuss the metric estimation problem. Although the results of this section are only auxiliary observations to be later used for edit distance, these results are of independent interest and may apply to future work. As defined previously, in this problem, we wish to estimate the distance matrix of a metric space $\langle \mathcal{M}, d\rangle$ with $n$ points. Notice that an estimation of a distance $d(p_i, p_j)$ with approximation factor $\alpha$ lies in the range $[d(p_i, p_j), \alpha d(p_i, p_j)]$; therefore, the estimated value cannot be less than the actual distance. However, it can be more than the actual distance by a multiplicative factor of $\alpha$. We tend to minimize the query complexity and the approximation factor; however, our algorithm is allowed to run in time $\widetilde{O}(n^2)$. Throughout this section, we show a tradeoff between the approximation factor and the quantum query complexity of metric estimation. First, we present an impossibility result that shows the approximation factor cannot be less than 3 unless we make a quadratic number of queries. Next, in Section 3.2, we present our desired $3+\epsilon$ approximation algorithm for metric estimation with a subquadratic query complexity. Afterward, we adjust our algorithm to make as few as $\widetilde{O}(n^{3/2+\epsilon}\text{poly}(1/\epsilon))$ oracle calls for a larger constant approximation $\mathsf{e}_\mathsf{m}(\epsilon) = O(1/\epsilon)$. Next, we show that $\Omega(n^{3/2})$ quantum queries are necessary for any constant factor approximation algorithm. Last but not least, we give similar results for $\Delta$-neighboring metric estimation, where we output the distance between two points only if their indices do not differ by more than $\Delta$.

### 3.1 Hardness of Approximation for $\alpha < 3$

As aforementioned, the purpose of this section is to show an impossibility result for approximating metric estimation within a factor smaller than 3 with subquadratic query complexity. To this end, we give a reduction from the well-known parity problem to the metric estimation problem. Parity is one of the problems for which quantum computers cannot perform better than classical computers. Recall the definition of the parity problem from Section 2.1.

| Parity |
| --- |
| Input: an integer $n$, and access to an oracle $O$ that upon receiving an integer $i$ reports the value of $f(i)$. $f$ is defined over $[n]$ and maps each index to either $-1$ or 1. |
| Output: $\text{par}(f) = \prod_{i\in[n]} f(i)$. |

Note that $\text{par}(f)$ is either $+1$ or $-1$ for every function $f$. Farhi et al. [27] proved that at least $\Omega(n)$ oracle queries are necessary to find $\text{par}(f)$. A classic method to show lower bounds on the

time/query complexity of problems is via a reduction from parity. This method has been used to show lower bounds on the quantum query complexity of many problems [25, 50].

The idea is to construct a metric space from a given function $f$ and show that any estimation of the metric with an approximation factor smaller than 3 can be used to compute the parity of $f$. A metric space should satisfy three properties: identity of indiscernibles, symmetry, and triangle inequality. Keep in mind that our construction should be in such a way that the metric meets all of the mentioned properties. For a function $f : [n^2] \to \{-1, 1\}$, we construct a metric $\mathcal{M} = \{a_1, a_2, \ldots, a_n, b_1, b_2, \ldots, b_n\}$ with $2n$ points. We divide the points into two groups, namely $a_i$s and $b_i$s, where the distances of the points within each group are all equal to 1. Moreover, for every pair of points $(a_i, b_i)$, the distance of $a_i$ from $b_i$ is either $1/2$ or $3/2$, depending on function $f$. We show that, given an $\alpha < 3$ approximation estimation for the distances of $\mathcal{M}$, one can determine $\mathrm{par}(f)$ uniquely.

Theorem 3.1. *Any quantum algorithm that approximates the metric estimation problem with an approximation factor smaller than* 3 *needs to make at least* $\Omega(n^2)$ *oracle calls.*

Proof. As promised, we prove this theorem by reducing the parity problem to the metric estimation problem. Suppose we are given an instance I of the parity problem consisting of $f : [m] \to \{0, 1\}$ and an oracle $O$ to access $f$. We assume w.l.o.g that $m = n^2$ and construct an instance $\mathrm{Cor}(\mathrm{I})$ of metric estimation as follows: let $\langle \mathcal{M}, d \rangle$ be a set of $2n$ points, where the distance of the points $p_i$ and $p_j$ is denoted by $d(p_i, p_j)$. We divide the points of the metric into two groups, $\{a_1, a_2, \ldots, a_n\}$ and $\{b_1, b_2, \ldots, b_n\}$. As mentioned before, the distances within the points of each group are equal to 1. Moreover, for every pair of points $a_i$ and $b_j$, we set $d(a_i, b_j)$ as follows:

$$d(a_i, b_j) = \begin{cases} 3/2 & f((i-1)n + j) = 1, \\ 1/2 & \text{otherwise.} \end{cases}$$

The identity and symmetry conditions are met by definition. We show that the triangle inequality also holds. If all three points of a triangle are in the same group (either $a_i$s or $b_i$s), then their distances are all 1. If they are in different groups, the distances are one of these cases, $\langle 1, 1/2, 1/2 \rangle$, $\langle 1, 1/2, 3/2 \rangle$, or $\langle 1, 3/2, 3/2 \rangle$, all of which meet the triangle inequality. Thus, $\langle \mathcal{M}, d \rangle$ is a valid metric space. One can trivially construct an oracle $Q$ for $\mathrm{Cor}(\mathrm{I})$ that reports the distance of a pair of points with a single query to $O$.

Now, suppose for the sake of contradiction that there exists a quantum algorithm that estimates the distances within a factor smaller than 3 with $o(n^2)$ query calls of $Q$. We show we can use this algorithm to find $\mathrm{par}(f)$ as follows. We first run the algorithm to approximate all of the distances via $o(n^2)$ query calls to $Q$. This costs us a total of $o(n^2)$ queries to $O$ since every query of $Q$ makes a call to $O$. Next, for every pair of points $(a_i, b_i)$ we determine $f((i-1)n + j)$ as follows:

$$f((i-1)n + j) = \begin{cases} 1 & d^*(a_i, b_j) \geq 3/2, \\ -1 & \text{otherwise,} \end{cases}$$

where $d^*(a_i, b_j)$ is the estimated distance of point $a_i$ from point $b_j$. The correctness of our reduction follows from the fact that the approximation factor of the algorithm for metric estimation is smaller than 3 and thus if $d^*(a_i, b_j) \geq 3/2$, the actual distance $d(a_i, b_j)$ is more than $1/2$. Finally, we take the multiplication of all determined values for $f$ and compute $\mathrm{par}(f)$ with $o(n^2) = o(m)$ queries. This contradicts the observation of Farhi et al. [27]. □

### 3.2 A $3+\epsilon$ Approximation Algorithm with $\widetilde{O}(n^{5/3}\text{poly}(1/\epsilon))$ Queries

In this section, we present a quantum algorithm to estimate the distances of a metric space within an approximation factor of $3+\epsilon$. Our algorithm makes $\widetilde{O}(n^{5/3}\text{poly}(1/\epsilon))$ oracle calls.

The first idea of our algorithm is to discretize the distances. Recall that the distances of the metric are non-negative integers in the interval $[0,u]$. We separate the numbers into disjoint intervals. We put a separate interval $[0,0]$ for 0 and continue on with the numbers in $[1,u]$. Every time, we find the smallest number $1 \le x \le u$ that is not covered in the previous intervals and add a new interval $[x,(1+\epsilon)x]$ to the list. Since $u = \text{poly}(n)$, the number of intervals is $\text{polylog}n\text{poly}(1/\epsilon) = \widetilde{O}(\text{poly}(1/\epsilon))$. Now, by losing a factor $1+\epsilon$ in the approximation, we can round up all of the numbers within an interval to its highest value and solve the problem for each interval separately. Therefore, the problem boils down to the following: given a threshold $t$, find all pairs of the points with a distance of at most $t$. We call this problem threshold estimation. Note that since we wish to find a 3 approximation solution for threshold estimation, a false positive is also allowed in the solution. More precisely, the solution should contain all pairs of points within a distance of at most $t$, but pairs within distances up to $3t$ are also allowed to be included.

In order to approximate threshold estimation, we subsequently make use of Grover's search algorithm [18]. Think of the metric as a graph $G$ where every point corresponds to a vertex of the graph and two vertices are adjacent if the distance of their corresponding points is at most $t$. Let $0 < \tau < 1$ be a fixed parameter. We call a vertex $v$ of the graph *low degree* if the number of edges incident to $v$ are bounded by $n^\tau$ and *high degree* otherwise. Our algorithm deals with low-degree vertices and high-degree vertices differently. We set the value of $\tau$ after the analysis and show it gives us the best bound.

In our algorithm, we iterate over the vertices of the graph and find their neighbors one by one. To this end, fix a vertex $v_i$ and suppose we wish to find all of its neighbors. Due to Grover's search (Theorem 2.1), we can list up to $n^\tau$ neighbors of $v_i$ with $\sqrt{n^\tau n} = n^{(1+\tau)/2}$ queries. Moreover, with an additional Grover's search, we can determine whether the degree of $v_i$ is more $n^\tau$ with $O(\sqrt{n})$ queries. If $v_i$ is low degree, we already have all its neighbors, and thus we can report those edges and remove $v_i$ from the graph. Otherwise, the degree of $v_i$ is more than $n^\tau$. In this case, we make $O(n)$ oracle calls and find the distances of all other points from the corresponding point of $v_i$, namely $p_i$. Based on these distances, we construct two sets of vertices $N(v_i,t)$ and $N(v_i,2t)$, where the former contains all vertices corresponding to points within a distance of at most $t$ of $p_i$ and the latter contains all of the vertices corresponding to points within a distance of at most $2t$ from $p_i$. We then proceed by reporting all the edges between $N(v_i,t)$ and $N(v_i,2t)$ and removing $N(v_i,t)$ from the graph. A pseudocode for this algorithm is shown in Algorithm 1.

Lemma 3.2. *For $\tau = 1/3$, Algorithm 1 approximates threshold estimation within a factor of 3 with $O(n^{5/3})$ oracle calls. Moreover, the running time of Algorithm 1 is $O(n^2)$.*

Proof. The correctness of our algorithm follows from the triangle inequality. We first show that for every pair of points $p_i$ and $p_j$ such that $d(p_i,p_j) \le t$, $A_{i,j} = 1$ at the end of the algorithm. To this end, consider the first time that we remove either $v_i$ or $v_j$ from the vertices. This could happen in two ways: either one of $v_i$ or $v_j$ is removed from the graph as a low-degree vertex or any of them is removed in an iteration of the algorithm for some high-degree vertex. In the former case, since we find all neighbors of the low-degree vertices, we detect the edge between them, and thus $A_{i,j} = 1$. Now, suppose that one of these vertices, say, $v_i$, is removed from the graph in an iteration for a vertex $v_x$ of the graph. Therefore, $d(v_i,v_x) \le t$. Moreover, due to the triangle inequality, $d(v_j,v_x) \le d(v_j,v_i) + d(v_i,v_x) \le 2t$ and thus $v_j \in N(v_x,2t)$. Thus, we set $A_{i,j} = 1$. Moreover, it

**ALGORITHM 1:** EstimateWithThreshold$(n, O, t)$

**Data**: The number of points in the metric space $\mathcal{M} = \{p_1, p_2, \ldots, p_n\}$, oracle access to the distances between points, and a threshold $t$.
**Result**: A 0-1 matrix $A$ of size $n \times n$, where for each $d(p_i, p_j) \leq t$ we have $A_{i,j} = 1$, and for each $A_{i,j} = 1$ we have $d(p_i, p_j) \leq 3t$.

```
Initialize a graph G with n vertices;
while V(G) is not empty do
    Select a vertex v_i from V(G);
    List up to n^τ neighbors of v_i and find out whether v_i is high degree or low degree;
    if v_i is low degree then
        Update the matrix A according to the edges of v_i;
        Remove v_i from V(G);
    else
        Find the distances of p_i from all other points;
        Construct N(v_i, t) and N(v_i, 2t) based on the distances;
        For every x ∈ N(v_i, t) and y ∈ N(v_i, 2t), set A_{x,y} = 1;
        V(G) ← V(G) \ N(v_i, t);
Output A;
```

follows from the triangle inequality that if we set $A_{i,j} = 1$ for some $i$ and $j$, then the distance of the points $p_i$ and $p_j$ is bounded by $3t$.

Trivially, the running time of the algorithm is $O(n^2)$. In what follows we show the query complexity of the algorithm is bounded by $O(n^{5/3})$. Let $Q(n)$ denote the query complexity of the algorithm for the case where $|V(G)| = n$. To compute $Q(n)$, we consider two cases separately: (1) when we select a vertex $v_i$ that is low degree and (2) when we select a vertex $v_i$ that is high degree. In any case, we make a search to list up to $n^\tau$ neighbors of $v_i$ and we make at least $O(n^{(1+\tau)/2})$ oracle calls. In addition to this, we make $O(\sqrt{n})$ more oracle calls to find out whether $v_i$ is low degree. In case $v_i$ is low degree, we remove $v_i$ from the graph and continue on with an instance with $n-1$ vertices. Otherwise, we make $O(n)$ more oracle calls and then remove $N(v_i, t)$ from the graph, which leaves us an instance with at most $n - n^\tau$ vertices. Therefore, we formulate $Q(n)$ as follows:

$$Q(n) = \begin{cases} O(n^{(1+\tau)/2}) + O(\sqrt{n}) + Q(n-1) & \text{if } v_i \text{ is low degree,} \\ O(n^{(1+\tau)/2}) + O(\sqrt{n}) + O(n) + Q(n - n^\tau) & \text{otherwise.} \end{cases}$$

Now we set $\tau = 1/3$ and thus we obtain

$$Q(n) = \begin{cases} O(n^{2/3}) + O(\sqrt{n}) + Q(n-1) = O(n^{2/3}) + Q(n-1) & \text{if } v_i \text{ is low degree,} \\ O(n^{2/3}) + O(\sqrt{n}) + O(n) + Q(n - n^{1/3}) = O(n) + Q(n - n^{1/3}) & \text{otherwise.} \end{cases}$$

A trivial analysis shows that for every vertex that we remove from $V(G)$, we make $O(n^{2/3})$ amortized query calls and thus the total number of queries is bounded by $n \cdot O(n^{2/3}) = O(n^{5/3})$. □

Now, we are ready to present our $3 + \epsilon$ approximation algorithm with query complexity $\widetilde{O}(n^{5/3}\text{poly}(1/\epsilon))$. For each $i$, using Algorithm 1, we can find all distances in range $[0, l(1+\epsilon/3)^{i+1}]$ with some false-positive distances in range $[l(1+\epsilon/3)^{i+1}, 3l(1+\epsilon/3)^{i+1}]$. By knowing the same information for $i-1$, we have all points in range $[0, l(1+\epsilon/3)^i]$ with some false-positive distances in range $[l(1+\epsilon/3)^i, 3l(1+\epsilon/3)^i]$. Thus, we can find all points in range $[l(1+\epsilon/3)^i], l(1+\epsilon/3)^{i+1}]$, some

false positives in range $[l(1+\epsilon/3)^{i+1}, 3l(1+\epsilon/3)^{i+1}]$, and some false negatives that estimated correctly before. All of these distances are in range $[l(1+\epsilon/3)^{i}, 3l(1+\epsilon/3)^{i+1}]$. Therefore, we can estimate these distances as $3l(1+\epsilon/3)^{i+1}$ and the approximation factor is $\frac{3l(1+\epsilon/3)^{i+1}}{l(1+\epsilon/3)^{i}} = 3(1+\epsilon/3) = 3+\epsilon$. The time and query complexity of this algorithm is the time and query complexity of Algorithm 1 times $\log_{1+\epsilon/3}(u/l) = \widetilde{O}(1/\epsilon)$. We handle zero distances separately. You can find the pseudocode of this algorithm in the following.

**ALGORITHM 2:** EstimateMetric$(n, O, \epsilon, l, u)$

**Data**: The number of points in the metric space $\mathcal{M} = \{p_1, p_2, \ldots, p_n\}$, oracle access to the distances between points, a small number $\epsilon > 0$, a lower bound, and an upper bound for the distances.
**Result**: An $n \times n$ matrix $A$, where $A_{i,j}$ is a $3+\epsilon$ approximation of $d(p_i, p_j)$

```
Initialize three matrices A, A°, and A•;
A° ← EstimateWithThreshold(n, O, 0);
Initialize the threshold: t ← max(1, l);
while t ≤ u do
    t ← t · (1 + ϵ/3);
    A• ←EstimateWithThreshold(n, O, t);
    A ← A + (A• − A°) · 3t;
    A° ← A° ∨ A•
output A
```

Theorem 3.3. *Algorithm 2 solves the metric estimation problem with approximation factor* $3+\epsilon$, *quantum query complexity* $\widetilde{O}(n^{5/3}\mathrm{poly}(1/\epsilon))$, *and time complexity of* $\widetilde{O}(n^2\mathrm{poly}(1/\epsilon))$ *for an arbitrary small constant* $\epsilon > 0$.

Proof. The correctness of Algorithm 2 follows from that of Algorithm 1. Moreover, Algorithm 2 runs Algorithm 1 $\widetilde{O}(\mathrm{poly}(1/\epsilon))$ times; therefore, the query complexity of Algorithm 2 is $\widetilde{O}(n^{5/3}\mathrm{poly}(1/\epsilon))$. Furthermore, the running time of Algorithm 2 is $\widetilde{O}(n^2\mathrm{poly}(1/\epsilon))$. □

In this section, we achieved an algorithm with subquadratic query complexity and approximation factor $3+\epsilon$ for any $\epsilon > 0$ that is nearly optimal due to Theorem 3.1. In Section 3.3, we reduce the quantum query complexity to $O(n^{3/2+\epsilon})$, but the approximation factor grows to a larger constant.

### 3.3 A Constant Approximation Algorithm with $\widetilde{O}(n^{3/2+\epsilon}\mathrm{poly}(1/\epsilon))$ Queries

In Sections 3.1 and 3.2, we showed that the best approximation factor that we can get with subquadratic oracle calls are bounded from below by 3 and that a $3+\epsilon$ approximation is possible. In this section, we complement this result by showing that the query complexity can be further reduced to $\widetilde{O}(n^{3/2+\epsilon}\mathrm{poly}(1/\epsilon))$, and moreover, we show that the required query complexity is at least $\Omega(n^{3/2})$ for any constant approximation factor. To this end, we present a quantum algorithm with expected query complexity $\widetilde{O}(n^{3/2+\epsilon}\mathrm{poly}(1/\epsilon))$, where the approximation factor and the expected running time are $\mathrm{e_m}(\epsilon) = O(1/\epsilon)$ and $\widetilde{O}(n^2\mathrm{poly}(1/\epsilon))$, respectively.

As stated before, the problem reduces to threshold estimation. Similar to what we did for Theorem 3.3, we divide the vertices into two categories, low degree and high degree. Low-degree vertices are easy to deal with; we simply list all of their neighbors using Grover's search and report all of them. If a vertex is high degree though, the algorithm needs to be more intelligent.

The overall idea is summarized in the following: we find a small group of vertices, namely *representatives*, that hits at least one vertex from the neighborhood of any large degree vertex. Using a

standard argument of hitting sets, we can show that a subset of $\widetilde{O}(n/\eta)$ vertices chosen uniformly at random, as *representatives*, hits every neighborhood of size at least $\eta$ with high probability. Notice that these neighborhoods are at most $n$ fixed but unknown subsets. Other vertices outside *representatives* are either low-degree vertices or *followers* that have at least one neighbor in *representatives*, or both. Next, we run the following procedure: for every vertex $v_i$ that is not in *representatives*, we first check if it is a follower. For a follower vertex that has at least one neighbor in *representatives*, we select one such vertex and call that the leader of $v_i$. Otherwise, if there is no such neighbor, we conclude that $v_i$ is indeed low degree; thus, we can find all its neighbors via Grover's search and update the solution. Next, we solve the problem recursively for all of the representatives. For any $v_i$ and $v_j$ that are connected, we want the leader of $v_i$ and the leader of $v_j$ to become connected in the recursive result. As a consequence of the triangle inequality, we can achieve this by tripling the threshold. Finally, we construct our solution based on the approximated solution of the representatives and the leader-follower relations, simply by connecting any two vertices where their leaders are connected. The approximation factor increases with each recursion, but since the number of recursions is a constant, we achieve a constant approximation factor. Furthermore, in each recursion call, we can increase the degree threshold as long as it does not increase the query complexity too much. By increasing the degree threshold to its third power, we have this property. The number of vertices in each nested recursion decreases. When the degree threshold becomes larger than the number of vertices, all vertices are considered low degree; thus, the next recursion call has zero vertices, and the process finishes.

The pseudocode of the algorithm is shown below.

**ALGORITHM 3:** FastEstimateWithThreshold($\mathcal{M}, O, t, \epsilon, {n_0}^\tau$)

**Data**: The number of points in the metric space $\mathcal{M} = \{p_1, p_2, \ldots, p_n\}$, oracle access to the distances between points, a threshold $t$, a small number $\epsilon$, and a degree threshold ${n_0}^\tau$

**Result**: An $n \times n$ matrix $A$, where $A_{i,j}$ is an $\mathsf{e_m}(\epsilon)$ approximation of $d(p_i, p_j)$

```
if n = 0 then
    Output an empty matrix;
else
    Sample a hitting set R with O((n/n_0^τ) log n) points;
    Initialize an n × n matrix A;
    for all points in M as v_i do
        Find a neighbor of v_i or v_i itself in R and save it as l(v_i) (the leader of v_i);
        if no such neighbor of v_i exists and v_i is not in R then
            List all neighbors of v_i;
    A' ← FastEstimateWithThreshold(R, O, 3t, 3ε, (n_0^τ)^3);
    for all pairs of points in M as (v_i, v_j), where l(v_i) ≠ ∅ and l(v_j) ≠ ∅ do
        if A'(l(v_i), l(v_j)) = 1 then
            A(v_i, v_j) ← 1;
    A ← A ∨ A';
    Output A;
```

Lemma 3.4. *Algorithm 3 called with the threshold $t$, the parameter $\epsilon$, and the degree threshold $n^{2\epsilon}$ finds all distances less than $t$ with some false-positive distances in range $[t, \mathsf{e_m}(\epsilon) \cdot t]$, where $\mathsf{e_m}(\epsilon) = O(1/\epsilon)$, in expected query complexity $\widetilde{O}(n^{3/2+\epsilon})$ and expected time complexity $\widetilde{O}(n^2)$.*

Proof. As aforementioned, we deal with three groups of vertices: *representatives*, *followers*, and low-degree vertices. Low-degree vertices may intersect with the other two, but each vertex is at least in one group. Here, for any low-degree vertex outside the other two groups, we find its neighborhood explicitly. Therefore, to show the correctness of the algorithm, we focus on two groups of followers and representatives. First, we show that we correctly find the group of representatives. A subset $\mathcal{R}$ of size $2(n/{n_0}^\tau)\ln n$ chosen uniformly at random misses one fixed neighborhood of size at least ${n_0}^\tau$ with a probability of at most $(1-\frac{{n_0}^\tau}{n})^{2(n/{n_0}^\tau)\ln n} \approx 1/\mathrm{e}^{2\ln n} = 1/n^2$. For all neighborhoods, which are at most $n$ fixed subsets of size at least ${n_0}^\tau$, the probability of missing at least one neighborhood is at most $n \cdot (1/n^2) = 1/n$ by the union bound. If $\mathcal{R}$ misses at least one large neighborhood, we can reset the algorithm. A standard argument of Las Vegas algorithms ensures that the expected query complexity and expected running time is no more than $\frac{n}{n-1}$ times the query complexity and running time of one execution, respectively (Exercise 1.3 of [51]). Now we can continue assuming we have leader-follower relations. Recall that for every follower $v_i$ we select one of its neighbors in $\mathcal{R}$ and call that vertex the leader of $v_i$. To simplify the last part of the algorithm, for any $v_i$ in $\mathcal{R}$, we call $v_i$ as the leader of itself. Thus, all followers and representatives have leaders.

Furthermore, we solve the problem for the group of representatives recursively, with different parameters. We triple the threshold in each recursion. Call the leader of two connected vertices $v_i$ and $v_j$ as $r_i$ and $r_j$, respectively. By the triangle inequality we have $d(r_i, r_j) \le d(r_i, v_i) + d(v_i, v_j) + d(v_j, r_j) \le 3t$. Thus, the leader of any two connected vertices is connected by the new threshold; hence, we find all distances less than $t$, perhaps with some false positives.

Before we compute the approximation factor $\mathrm{e_m}(\epsilon)$, we determine the number of nested recursion calls. We call the number of vertices in the $i$th recursion call $n_i$. Note that $n_i$ is the size of the representatives group of the $(i-1)$'th recursion. Thus, we have $n_i = O((n_{i-1}/{n_0}^{\tau_i})\log n_{i-1})$. Using induction, we can show that the degree threshold in the $i$th recursion call is ${n_0}^{\tau_i} = n^{(2\cdot 3^i)\epsilon}$, and therefore, $n_i = O(n^{1-(3^i-1)\epsilon}\cdot O(\log^i(n)))$. The number of vertices becomes zero in the $i$th recursion, where $1-(3^i-1)\epsilon < 0$ or $i > \log_3(1+1/\epsilon)$. Hence, we have at most $k(\epsilon) = \log_3(1/\epsilon)+1$ nested recursion calls, which is independent of $n$. Notice that $k(3\epsilon) = k(\epsilon) - 1$ and $k(3^{k(\epsilon)}\epsilon) = 0$.

What remains is to compute the approximation factor $\mathrm{e_m}(\epsilon)$. The maximum distance of a pair of vertices that we report an edge between them is at most $2(1+3\mathrm{e_m}(3\epsilon))$ times the threshold. We know that $\mathrm{e_m}(3^{k(\epsilon)}\epsilon) = 1$; therefore, $\mathrm{e_m}(\epsilon) \le 9/\epsilon = O(1/\epsilon)$.

The query complexity of Grover's search in the $i$th recursion is at most $n_i O(\sqrt{|\mathcal{R}|})$ to find the leader of each point, plus $n_i O(\sqrt{n_i \cdot {n_0}^{\tau_i}})$ to find all neighbors of some low-degree points. This is equal to $O(n^{3/2-(\frac{5\cdot 3^i-3}{2})\epsilon}\mathrm{polylog}(n)) + O(n^{3/2-(\frac{3^i-3}{2})\epsilon}\mathrm{polylog}(n))$. Notice that the latter term dominates the former, and the query complexity for $i=0$ dominates all of the recursions; therefore, the query complexity of Algorithm 3 is at most $O(n^{3/2+\epsilon})$.

The time complexity is at most $O(n^2)$ in each phase. Thus, the time complexity is $O(n^2\mathrm{polylog}(n))$. □

In what follows, we complete our algorithm using Algorithm 3 with several thresholds. This is the same as Algorithm 2 with minor differences such as line 8, where 3 has been replaced with $\mathrm{e_m}(\epsilon)$.

Theorem 3.5. *Algorithm 4 solves the metric estimation with approximation factor* $\mathrm{e_m}(\epsilon) = O(1/\epsilon)$, *with query complexity* $\widetilde{O}(n^{3/2+\epsilon}\mathrm{poly}(1/\epsilon))$ *in time* $\widetilde{O}(n^2\mathrm{poly}(1/\epsilon))$.

Proof. The correctness of Algorithm 4 follows from that of Algorithm 3, the same as we did in Theorem 3.3. Moreover, Algorithm 4 runs Algorithm 3 $\widetilde{O}(\mathrm{poly}(1/\epsilon))$ times; therefore, the query

**ALGORITHM 4:** FastEstimateMetric($\mathcal{M}, O, \epsilon, l, u$)

**Data**: The number of points in the metric space $\mathcal{M}$, oracle access to the distances between points, a small number $\epsilon > 0$, a lower bound, and an upper bound for the distances
**Result**: An $n \times n$ matrix $A$, where $A_{i,j}$ is an $\mathsf{e_m}(\epsilon)$ approximation of $d(p_i, p_j)$ in $\langle \mathcal{M}, d \rangle$

Initialize the distance estimation matrix $A$, $A^\circ$, and $A^\bullet$;
$A^\circ \leftarrow$FastEstimateWithThreshold$(n, O, 0, \epsilon, n^{2\epsilon})$;
Initialize the threshold: $t \leftarrow \max(1, l)$;
**while** $t \leq u$ **do**
  $t \leftarrow t \cdot (1 + \epsilon)$;
  $A^\bullet \leftarrow$FastEstimateWithThreshold$(n, O, t, \epsilon, n^{2\epsilon})$;
  $A \leftarrow A + (A^\bullet - A^\circ) \cdot \mathsf{e_m}(\epsilon)$;
  $A^\circ \leftarrow A^\circ \vee A^\bullet$;
Output $A$;

complexity of Algorithm 2 is $\widetilde{O}(n^{3/2+\epsilon}\mathsf{poly}(1/\epsilon))$. Furthermore, the running time of Algorithm 2 is $\widetilde{O}(n^2\mathsf{poly}(1/\epsilon))$. □

### 3.4 An $\Omega(n^{3/2})$ Time Lower Bound

Last but not least, we show that the query complexity of metric estimation cannot be reduced any further, so long as the approximation factor is constant; i.e., we need at least $\Omega(n^{3/2})$ queries to approximate metric estimation within a constant factor. We use Ambainis's lower-bound technique [1].

Theorem 3.6 (Proven in [1], Theorem 6). *Let $f(x_1, \ldots, x_n)$ be a function of $n$ variables with values from some finite set and $X, Y$ be two sets of inputs such that $f(x) \neq f(y)$ if $x \in X$ and $y \in Y$. Let $R \subset X \times Y$ be such that*

(1) *For every $x \in X$, there exist at least $m$ different $y \in Y$ such that $(x, y) \in R$.*
(2) *For every $y \in Y$, there exist at least $m'$ different $x \in X$ such that $(x, y) \in R$.*

*Let $l_{x,i}$ be the number of $y \in Y$ such that $(x, y) \in R$ and $x_i \neq y_i$ and $l_{y,i}$ be the number of $x \in X$ such that $(x, y) \in R$ and $x_i \neq y_i$. Let $l_{max}$ be the maximum of $l_{x,i} l_{y,i}$ over all $(x, y) \in R$ and $i \in \{1, \ldots, N\}$ such that $x_i \neq y_i$. Then, any quantum algorithm computing $f$ uses $\Omega(\sqrt{\frac{mm'}{l_{max}}})$ queries.*

Now we use an intermediate problem to prove the desired lower bound. A permutation matrix is a Boolean $n \times n$ matrix, which has exactly one entry 1 in each row and each column. It corresponds to a permutation $\pi$ where entries of 1 are in the form of $(i, \pi(i))$. The sign of a permutation matrix is defined as the sign of its corresponding permutation. The next lemma about the problem of determining the sign of a permutation matrix is the main part of our lower bound.

Lemma 3.7. *Any quantum algorithm that takes an $n \times n$ permutation matrix as the input and outputs the sign of the permutation matrix has a query complexity of at least $\Omega(n^{3/2})$.*

Proof. To apply Theorem 3.6, we use a single index to address an entity instead of two indices. Assume $f(x_1, x_2, \ldots, x_{n^2})$ is a function that takes a permutation matrix as input and outputs a value in $\{-1, 1\}$ as the sign of the matrix. Define $X$ as the set of permutation matrices with sign $-1$, $Y$ as the set of permutation matrices with sign 1, and $R \subset X \times Y$ such that $(x, y) \in R$ iff their corresponding matrices can be transformed to the other with a swap of just two rows. Therefore, we have $m = m' = \binom{n}{2}$. For an $i$ we have $l_{x,i} = n - 1$ and $l_{y,i} = 1$ if $x_i = 1$ and $l_{x,i} = 1$ and

$l_{y,i} = n - 1$ if $x_i = 0$; thus, $l_{max} = n - 1$. Therefore, by Theorem 3.6, every quantum algorithm to solve this problem has a query complexity of at least $\Omega\left(\sqrt{\frac{\binom{n}{2}^2}{n-1}}\right) = \Omega(n^{3/2})$. □

The problem of determining the sign of an $n \times n$ permutation matrix can be easily reduced to our problem, by constructing a bipartite graph with parts $X$ and $Y$, $n$ vertices in each part, and $n$ edges that form a complete matching between $X$ and $Y$. Every matching has a corresponding permutation and vice versa. Therefore, we have the following theorem.

Theorem 3.8. *Any quantum algorithm that estimates distances of a metric space of n points with a constant approximation factor has a query complexity of at least* $\Omega(n^{3/2})$.

Proof. We simply reduce the problem of determining the sign of a permutation matrix to this problem. Assume $n$ is an even number. For an instance of an $n/2 \times n/2$ permutation matrix $\mathcal{A}$, we construct a metric space $\mathcal{M}$ with $n$ points, $r_i$ for row $i$, and $c_j$ for column $j$ of the matrix. Make the distance between $r_i$ and $c_j$ equal to 1, where $\mathcal{A}_{i,j} = 1$ and a distance of $n^2$ otherwise. The distances meet the necessary conditions. Notice that we do not construct the distances; we construct an oracle that invokes the oracle of $\mathcal{A}$ at most one time. Using Lemma 3.7, the query complexity is at least $\Omega((n/2)^{3/2}) = \Omega(n^{3/2})$. □

### 3.5 Δ-Neighboring Metric Estimation

In Δ-neighboring metric estimation, for a metric $\mathcal{M} = \{p_1, p_2, \ldots, p_n\}$, we output $d(p_i, p_j)$ only if $|i-j| \leq \Delta$. For $\Delta = n$ this problem is equivalent to metric estimation. However, when $\Delta = o(n)$, we can improve the running times of the algorithms. To this end, we prove similar hardness results and approximation algorithm for Δ-neighboring metric estimation. In the following, we prove four main corollaries, similar to the previous four theorems.

To prove a similar lower bound on the query complexity of Δ-neighboring metric estimation, we again use a reduction from the parity problem. For a function $f : [O(\Delta n)] \to \{-1, 1\}$, we construct a metric $\mathcal{M} = \{a_1, b_1, a_2, b_2, \ldots, a_n, b_n\}$ with $2n$ points. Here, we map the function into $d(a_i, b_j)$s, where $|i - j| \leq \Delta$. Using a similar construction as Section 3.1, given an $\alpha < 3$ approximation estimation for $2\Delta$-neighboring metric estimation, one can determine $\text{par}(f)$ uniquely.

Corollary 3.9. *Any quantum algorithm that approximates the* Δ*-neighboring metric estimation problem with an approximation factor smaller than* 3 *needs to make at least* $\Omega(n\Delta)$ *oracle calls.*

Proof. Suppose for the sake of contradiction that there exists a quantum algorithm that estimates the distances within a factor smaller than 3 with $o(n\Delta)$ query calls of $Q$. Using the same construction as Theorem 3.1, for a given instance I of the parity problem consisting of $f : [m] \to \{0, 1\}$ and an oracle $O$ to access $f$, we construct an instance Cor(I) of Δ-neighboring metric estimation. Without loss of generality, we can assume $m = \Delta^2 + (n - \Delta)(2\Delta - 1) = O(n\Delta)$. Let $\langle \mathcal{M}, d \rangle$ where $\mathcal{M} = \{a_1, b_1, a_2, b_2, \ldots, a_n, b_n\}$ be a set of $2n$ points where the distance of the points $p_i$ and $p_j$ is denoted by $d(p_i, p_j)$. As mentioned before, all $d(a_i, a_j)$s and $d(b_i, b_j)$s are equal to 1. We then map $f(k)$s to $d(a_i, b_j)$s, where $|i - j| \leq \Delta$, similar to Theorem 3.1. The identity, symmetry, and triangle inequality conditions are met as before. Thus, $\langle \mathcal{M}, d \rangle$ is a valid metric space. One can trivially construct an oracle $Q$ for Cor(I) that reports the distance of a pair of points with a single query to $O$.

We first run the algorithm to approximate all of the distances via $o(n\Delta)$ query calls to $Q$. This costs us a total of $o(n\Delta)$ queries to $O$ since every query of $Q$ makes a call to $O$. Next, for every pair of points $(a_i, b_i)$ we determine the corresponding $f(k)$ the same as before. Finally, we take

the multiplication of all determined values for $f$ and compute $\mathrm{par}(f)$ with $o(n\Delta) = o(m)$ queries. This contradicts the observation of Farhi et al. [27]. □

Next, we solve threshold estimation within an approximation factor of 3 using $O(n\Delta^{2/3})$ quantum queries. Note that in threshold estimation of $\Delta$-neighboring metric estimation, every vertex has at most $2\Delta + 1$ potential neighbors. Therefore, here we call a vertex $v$ *low degree* if the number of edges incident to $v$ are bounded by $\Delta^\tau$ and *high degree* otherwise. Moreover, we solve the problem similar to Algorithm 1, taking into account the potential neighbors.

Corollary 3.10. *We can approximate threshold estimation within a factor of* 3 *with* $O(n\Delta^{2/3})$ *oracle calls. Moreover, the running time of our solution is* $O(n\Delta)$.

Proof. The correctness of our solution is analogous to Lemma 3.2. Moreover, the running time of the algorithm is $O(n\Delta)$. In what follows we show that the query complexity of the algorithm is bounded by $O(n\Delta^{2/3})$. Let $Q(n)$ denote the query complexity of the algorithm for the case where $|V(G)| = n$. To compute $Q(n)$, we consider two cases separately: (1) when we select a vertex $v_i$ that is low degree and (2) when we select a vertex $v_i$ that is high degree. In any case, we make a search to list up to $\Delta^\tau$ neighbors of $v_i$ and we make at least $O(\Delta^{(1+\tau)/2})$ oracle calls. In addition to this, we make $O(\sqrt{\Delta})$ more oracle calls to find out whether $v_i$ is low degree or not. In the case where $v_i$ is low degree, we remove $v_i$ from the graph and continue on with an instance with $n-1$ vertices. Otherwise, we make $O(\Delta)$ more oracle calls and then remove $N(v_i, t)$ from the graph, which leaves us an instance with at most $n - \Delta^\tau$ vertices. Therefore, we formulate $Q(n)$ as follows:

$$Q(n) = \begin{cases} O(\Delta^{(1+\tau)/2}) + O(\sqrt{\Delta}) + Q(n-1) & \text{if } v_i \text{ is low degree,} \\ O(\Delta^{(1+\tau)/2}) + O(\sqrt{\Delta}) + O(\Delta) + Q(n-\Delta^\tau) & \text{otherwise.} \end{cases}$$

Now we set $\tau = 1/3$ and thus we obtain

$$Q(n) = \begin{cases} O(\Delta^{2/3}) + O(\sqrt{\Delta}) + Q(n-1) = O(\Delta^{2/3}) + Q(n-1) & \text{if } v_i \text{ is low degree,} \\ O(\Delta^{2/3}) + O(\sqrt{\Delta}) + O(\Delta) + Q(n-\Delta^{1/3}) = O(\Delta) + Q(n-\Delta^{1/3}) & \text{otherwise.} \end{cases}$$

A trivial analysis shows that for every vertex that we remove from $V(G)$, we make $O(\Delta^{2/3})$ amortized query calls and thus the total number of queries is bounded by $n \cdot O(\Delta^{2/3}) = O(n\Delta^{2/3})$. □

Using the same technique, we can solve $\Delta$-neighboring metric estimation using the solution of Corollary 3.10.

Corollary 3.11. *We can solve the* $\Delta$*-neighboring metric estimation problem with approximation factor* $3+\epsilon$*, quantum query complexity* $\widetilde{O}(n\Delta^{2/3}\mathrm{poly}(1/\epsilon))$*, and time complexity* $\widetilde{O}(n\Delta\mathrm{poly}(1/\epsilon))$ *for an arbitrary small constant* $\epsilon > 0$.

Next, we improve the query complexity by increasing the approximation factor, similar to Section 3.3.

Corollary 3.12. *For a threshold* $t$*, a parameter* $\epsilon$*, and a degree threshold* $n^{2\epsilon}$ *we can find all distances less than* $t$ *with some false-positive distances in the range* $[t, \mathrm{e_m}(\epsilon)\cdot t]$*, where* $\mathrm{e_m}(\epsilon) = O(1/\epsilon)$*, in expected query complexity* $\widetilde{O}(n\Delta^{1/2+\epsilon})$ *and expected time complexity* $\widetilde{O}(n\Delta)$.

Proof. We use a solution similar to Algorithm 3, where we start with degree threshold $\Delta^{2\epsilon}$. The correctness and approximation factor of our solution are the same as Lemma 3.4.

To prove the query complexity, note that $n_i = O((n_{i-1}/\Delta^{\tau_i})/\log n_{i-1})$ and $\Delta^{\tau_i} = \Delta^{(2\cdot 3^i)\epsilon}$. Therefore, $n_i = O(n/\Delta^{(3^i-1)\epsilon} \cdot O(\log^i(n)))$. The expected query complexity of Grover's search in the $i$th recursion is at most $n_i O(\sqrt{(\Delta/n)|\mathcal{R}|})$ to find the leader of each point, plus $n_i O(\sqrt{n_i \cdot \Delta^{\tau_i}})$

to find all neighbors of some low-degree points. This is equal to $O(n\Delta^{1/2-(\frac{5\cdot 3^i-3}{2})\epsilon}\text{polylog}(n)) + O(n\Delta^{1/2-(\frac{3^i-3}{2})\epsilon}\text{polylog}(n))$. Notice that the latter term dominates the former, and the query complexity for $i = 0$ dominates all of the recursions; therefore, the query complexity of Algorithm 3 is at most $\widetilde{O}(n\Delta^{3/2+\epsilon}\text{poly}(1/\epsilon))$.

The time complexity is at most $O(n\Delta)$ in each phase. Thus, the time complexity is $O(n\Delta\text{poly}(1/\epsilon))$. □

Using Corollary 3.12, we can solve $\Delta$-neighboring metric estimation using the same technique as before.

Corollary 3.13. *We can solve $\Delta$-neighboring metric estimation with approximation factor $\mathsf{e}_\mathsf{m}(\epsilon) = O(1/\epsilon)$ and query complexity $\widetilde{O}(n\Delta^{1/2+\epsilon}\text{poly}(1/\epsilon))$ in time $\widetilde{O}(n\Delta\text{poly}(1/\epsilon))$.*

Finally, we can show a lower bound on the query complexity of any algorithm that approximates $\Delta$-neighboring metric estimation within a constant factor using similar techniques.

Corollary 3.14. *Any quantum algorithm that solves $\Delta$-neighboring metric estimation within a constant approximation factor has a query complexity of at least $\Omega(n\Delta^{1/2})$.*

Proof. In Lemma 3.7, instead of making a general permutation matrix, we make a block diagonal matrix, having $n/\Delta$ blocks, where each block is a $\Delta \times \Delta$ permutation matrix. Therefore, $m = m' = (n/\Delta)\binom{\Delta}{2}$. Moreover, for an $i$ we have $l_{x,i} = k-1$ and $l_{y,i} = 1$ if $x_i = 1$, and $l_{x,i} = 1$ and $l_{y,i} = k-1$ if $x_i = 0$; thus, $l_{max} = k-1$. Therefore, by Theorem 3.6, every quantum algorithm to solve this problem has a query complexity of at least $\Omega\left(\sqrt{\frac{(n/\Delta)^2\binom{\Delta}{2}^2}{\Delta-1}}\right) = \Omega(n\Delta^{1/2})$.

Next, we reduce it to an instance of $\Delta$-neighboring metric estimation with $2n$ points similar to Theorem 3.8. Note that if $\mathcal{A}_{i,j} = 1$, then $|i-j| \le \Delta$. Therefore, we have a metric similar to Theorem 3.1, and we conclude that solving $\Delta$-neighboring metric estimation within a constant approximation factor needs at least $\Omega(n\Delta^{1/2})$ queries. □

## 4 EDIT DISTANCE

In this section, we use the results of Section 3 to design a quantum approximation algorithm for the edit distance problem. Our algorithm has an approximation factor of $3+\epsilon$ for an arbitrarily small number $\epsilon > 0$ and time complexity $\widetilde{O}(n^{2-4/21}\text{poly}(1/\epsilon))$. The outline of the algorithm is presented in Section 2. Here, we provide detailed proofs of the lemmas and theorems we used previously for edit distance. The first lemma states that given the distances between windows, we can find an optimal window-compatible transformation using a DP-based algorithm.

Lemma 4.1. *Given a positive integer $\Delta$ and edit distances between pairs of windows of $W_1$ and $W_2$ in which their indices do not differ by more than $\Delta$, one can compute an optimal window-compatible transformation of $s_1$ into $s_2$ in time $O(n + \Delta \cdot \min(|W_1|, |W_2|))$, where two windows can match only if their indices do not differ by more than $\Delta$.*

Proof. We take a dynamic programming approach to find an optimal window-compatible transformation of the two strings. Recall that $W_1 = \langle w_1, w_2, \ldots, w_k\rangle$ and $W_2 = \langle w'_1, w'_2, \ldots, w'_{k'}\rangle$ are collections of windows for $s_1$ and $s_2$, respectively, with maximum window size $l$, positive parameter $\mu$, and gap size $g$. We note that every window $w$ corresponds to a subinterval of $[1, n]$, and thus we can use a linear time sorting algorithm, such as bucket-sort, and sort all the windows in time $O(n)$. Therefore, we can assume that the windows in $W_1$ and $W_2$ are sorted according to their right side. For a window $w$ of a string $s$, we denote the index of its left character in $s$ by $\text{left}(w)$

and the index of its right character in $s$ by $\text{right}(w)$. Additionally, for a window $w_i$ ($w'_j$) we define $\text{prev}(w_i)$ ($\text{prev}(w'_j)$) to be the last window in $W_1$ ($W_2$) before $w_i$ ($w'_j$) that does not overlap with $w_i$ ($w'_j$). Assuming the windows are sorted according to their right side, we can compute all $\text{prev}(w_i)$s and $\text{prev}(w'_j)$'s in linear time (i.e., $O(|W_1| + |W_2|)$).

For every $1 \le i \le k$ and $1 \le j \le k'$ we define auxiliary variables $c_{i,j}$s as the cost of an optimal window-compatible transformation of $s_1[1, \text{right}(w_i)]$ into $s_2[1, \text{right}(w'_j)]$, corresponding to $\langle w_1, w_2, \ldots, w_i\rangle$ and $\langle w'_1, w'_2, \ldots, w'_j\rangle$. For the sake of simplicity, we define $c_{i,0} = \text{right}(w_i)$, which is the cost of deleting all characters of $s_1[1, \text{right}(w_i)]$, and $c_{0,j} = \text{right}(w'_j)$, which is the cost of inserting all characters of $s_2[1, \text{right}(w'_j)]$. For every $1 \le i \le k$ and $1 \le j \le k'$ the following recursive formula holds:

$$c_{i,j} = \min\Big\{c_{i-1,j} + (\text{right}(w_i) - \text{right}(w_{i-1})),\ \ c_{i,j-1} + (\text{right}(w'_j) - \text{right}(w'_{j-1})),\ \ d(w_i, w'_j) + c_{\text{prev}(w_i),\text{prev}(w'_j)} + (\text{left}(w_i) - \text{right}(\text{prev}(w_i)) - 1) + (\text{left}(w'_j) - \text{right}(\text{prev}(w'_j)) - 1)\Big\}. \tag{1}$$

To compute $c_{i,j}$ recursively, we have three possibilities in an optimal window-compatible transformation. In particular, either $w_i$ is matched with $w'_j$ or at least one of $w_i$ and $w'_j$ is unmatched. If we match $w_i$ to $w'_j$, then there is a cost of $d(w_i, w'_j)$ for transforming $w_i$ to $w'_j$. Also, the other windows that overlap with $w_i$ or $w'_j$ cannot be used. Consequently, the problem reduces to finding an optimal window-compatible transformation of $s_1[1, \text{right}(\text{prev}(w_i))]$ into $s_2[1, \text{right}(\text{prev}(w'_j))]$ with respect to $\langle w_1, w_2, \ldots, w_{\text{prev}(w_i)}\rangle$ and $\langle w'_1, w'_2, \ldots, w'_{\text{prev}(w'_j)}\rangle$. This subproblem is captured by $c_{\text{prev}(w_i),\text{prev}(w'_j)}$. Moreover, some characters in between $\text{prev}(w_i)$ and $w_i$ (and between $\text{prev}(w'_j)$ and $w'_j$) may exist that should be deleted (inserted). For the case that $w_i$ is unmatched, we need $(\text{right}(w_i) - \text{right}(w_{i-1}))$ operations to remove every character after $w_{i-1}$ in $s_1[1, \text{right}(w_i)]$. This is because no other window can cover these characters. For the remaining characters the problem reduces to finding an optimal window-compatible transformation for $s_1[1, \text{right}(w_{i-1})]$ into $s_2[1, \text{right}(w'_j)]$ with respect to $\langle w_1, w_2, \ldots, w_{i-1}\rangle$ and $\langle w'_1, w'_2, \ldots, w'_j\rangle$, which is captured by $c_{i-1,j}$. Likewise, we can formulate the case where $w'_j$ is unmatched.

Note that $c_{k,k'} + (|s_1| - \text{right}(w_k)) + (|s_2| - \text{right}(w'_{k'}))$ is the cost of an optimal window-compatible transformation from $s_1$ to $s_2$. W.l.g. assume $k \le k'$. By iterating through $i$ from 1 to $k$ and $j$ from $\max(1, i - \Delta)$ to $\min(k', i + \Delta)$, one can simply calculate each $c_{i,j}$ where $|i - j| \le \Delta$ in time $O(1)$ using Equation (1). We can assume $c_{i,j} = \infty$, where $|i - j| > \Delta$. Therefore, we can compute $c_{k,k'}$ in time $O(\Delta \cdot \min(k, k'))$, which completes the proof. □

If we are not given the exact distances between the windows but rather an $\alpha$ approximation of them, the algorithm of Lemma 4.1 also finds an $\alpha$ approximation of the optimal window-compatible transformation due to the linearity of Equation (1).

Corollary 4.2. *Given a positive integer $\Delta$ and an $\alpha$-approximation of edit distances between pairs of windows of $W_1$ and $W_2$ in which their indices do not differ by more than $\Delta$, one can compute an $\alpha$-approximation of $s_1$ into $s_2$ in time $O(n + \Delta \cdot \min(|W_1|, |W_2|))$, where two windows can match only if their indices do not differ by more than $\Delta$.*

Lemma 4.3 presents our main argument to prove the approximation factor of our algorithm. Roughly speaking, it states that the cost of an optimal window-compatible transformation is close to the cost of the optimal transformation, i.e., edit distance between $s_1$ and $s_2$.

Lemma 4.3. *Given that $\text{edit}(s_1, s_2) \le \delta n$, there exists a window-compatible transformation of $s_1$ into $s_2$ with respect to $W_1$ and $W_2$, in which for each window $w_i \in W_1$ that matches to a window*

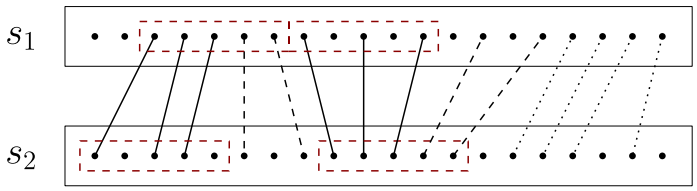



Fig. 5. An edge from $x \in s_1$ to $y \in s_2$ shows that opt transforms $x$ into $y$ with either no change or a substitution. Dashed edges represent those with one covered endpoint. Dotted edges represent those that remained at the end of the iteration.

*$w_2 \in W_2$, their indices do not differ by more than $\Delta = O(\log n(1/\mu)\delta n/g)$, and have at most $((1 + 2\mu)\delta + 1/\gamma)n + 2l$ operations.*

Proof. Recall that $l$ is the maximum length of the windows and $\gamma$ is the number of layers. Let opt be a minimum size transformation of $s_1$ into $s_2$. The overall idea of the proof is as follows. We first show that there exists a set of non-overlapping windows of maximum length $l$, such that a window-compatible transformation with respect to them approximates opt. Next, by shifting those windows and losing a small fraction on the approximation factor, we fit them to those in $W_1$ and $W_2$.

Consider a pair of characters $x \in s_1$ and $y \in s_2$, such that opt transforms $x$ into $y$ either with no change or through a substitution. We call such a pair an edge. Note that there is no collision in the set of all edges in opt (or generally in any transformation); i.e., for edges $(x_1, y_1)$ and $(x_2, y_2)$, if $x_1 < x_2$, then $y_1 < y_2$. Let $M = \langle (x_1, y_1), \ldots, (x_m, y_m) \rangle$ be the sequence of all edges in opt in order from left to right.

First, we find a set of windows as follows. Roughly speaking, we iterate through $M$ and at each step put as many edges as possible in two windows of length $l$, in $s_1$ and $s_2$, and shrink one of the windows if necessary. This step is shown in Figure 6. In particular, let $\rho(i)$ be the smallest index in $M$ such that $x_{\rho(i)}$ and $y_{\rho(i)}$ are not covered by any window up to step $i$. We first create a window $v_i$ of length $l$ starting from $x_{\rho(i)}$ and window $v'_i$ of length $l$ starting from $y_{\rho(i)}$, and we stop when any such window goes beyond the length of the strings. In this way, when we create $v_i$ and $v'_i$ of size $l$, there might be some edges that have one endpoint in $v_i$ and one endpoint beyond $v'_i$ or vice versa. Consider the case in which these edges have one endpoint in $v_i$. We then shrink $v_i$ by removing $(1 + \mu)^j$ characters from its right side, where $j$ is the minimum value such that all half-covered edges are not covered anymore. Now, we may have some edges that have one endpoint in $v'_i$ and one endpoint beyond the newly shrunk $v_i$. Let $h(i)$ be the number of such edges, and let $p(i)$ be the number of characters in $v'_i$ that opt transforms them through insertion. We claim that $h(i) \leq \mu p(i)$. This is because $v'_i$ has at most $|v_i| + h(i)$ edges in opt; hence, $p(i) \geq l - |v_i| - h(i)$. Also note that a shrinkage of size $(1 + \mu)^{j-1}$ would not delete any of these $h(i)$ endpoints. Therefore, we have $(1 + \mu)^{j-1} < l - |v_i| - h(i)$. Also, note that $l - |v_i| = (1 + \mu)^j$. This implies that

$$h(i) < (1 + \mu)^j - (1 + \mu)^{j-1} = (1 + \mu)^{j-1}(1 + \mu - 1) \leq (l - |v_i| - h(i))\mu \leq p(i)\mu.$$

In comparison to opt, a transformation with respect to $v_i$ and $v'_i$ can keep all the edges between $v_i$ and $v'_i$ and apply deletion and insertion for those edges that have one endpoint in $v_i$ and one endpoint out of $v'_i$. This costs at most $2h(i)$ more.

Besides, the number of remaining edges at the end of the iteration on $M$ is at most $l$. Such edges can be transformed by at most $2l$ insertions and deletions. Hence, requiring the transformation to be with respect to all $v_i$s and $v'_i$s adds at most $2l+2\sum h(i) \leq 2l+2\mu\sum p(i) \leq 2l+2\mu|\text{opt}| \leq 2l+2\mu\delta n$

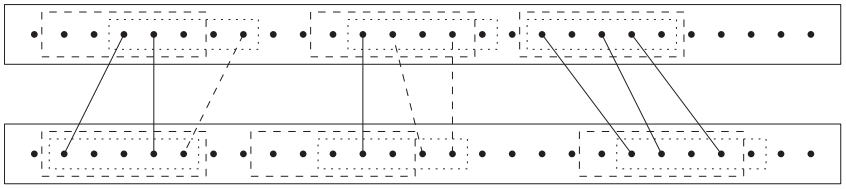


Fig. 6. Dotted rectangles represent $v_i$s and $v'_i$s. Dashed rectangles represent shifted windows that are in $W_1$ or $W_2$. Dashed lines represent edges that are left outside of windows after shifting.

more operations to the optimum solution. Equivalently, the optimum transformation with respect to these windows has at most $(1 + 2\mu)\delta n + 2l$ operations.

Finally, we note that the gap size between the windows in $W_1$ is $g = l/\gamma$; therefore, one can shift $v_i$s by at most $g/2$ to the right or left in order to map them to non-overlapping windows in $W_1$. Likewise, one can find non-overlapping windows for $v'_i$s in $W_2$. This step is shown in Figure 6. Every shift of a window leaves at most $g/2$ of the edges outside, which costs an extra $g$ operations. Since there are at most $n/l$ windows since $|v_i| + |v'_i| \geq l$ in each step, the overall cost of shifting the windows is $n/\gamma$. Therefore, there exists a subset of $W_1$ and $W_2$ such that the optimum window-compatible transformation of $s_1$ into $s_2$ with respect to them has at most $2l + (1 + 2\mu)\delta n + n/\gamma = ((1 + 2\mu)\delta + 1/\gamma)n + 2l$ operations.

The remaining part is to prove that the indices of any two matching windows in our solution do not differ by more than $O(\log n(1/\mu)\delta n/g)$. We already know that (e.g., see Corollary 1 of [57]) for any edge $(x_{\rho_i}, y_{\rho_i}) \in M$, their indices differ by at most $\text{edit}(s_1, s_2) \leq \delta n$. Therefore, for any $v_i$ and $v'_i$, positions of their first characters differ by at most $\delta n + g$ after shifting the windows. From this, we can directly conclude that the indices of the corresponding shifted windows in $W_1$ and $W_2$ differ by at most $(\log_{1+\mu} n) \cdot (\delta n + g)/g = O(\log n(1/\mu)(\delta n/g))$. □

The next lemma proves the approximation factor and time complexity of our $3+\epsilon$ approximation algorithm for the $\delta$-bounded edit distance problem.

Lemma 4.4. *There exists a quantum algorithm that solves the $\delta$-bounded edit distance problem within an approximation factor of $3 + \epsilon$ in time $\widetilde{O}(n^{2-4/21}\text{poly}(1/\epsilon))$.*

Proof. We can assume that $\delta > n^{-2/21}$; otherwise, one can use the $O(n+d^2)$ algorithm of Landau et al. [45] for strings of distance at most $d$ and find the exact edit distance in time $O(n + \delta^2 n^2) = O(n^{2-4/21})$.

We prove that the algorithm discussed in Section 2 leads to an approximation factor of $3 + \epsilon$ in quantum running time $\widetilde{O}(n^{2-4/21}\text{poly}(1/\epsilon))$. To this end, let us go through the algorithm step by step.

Note that the total number of windows is equal to $\widetilde{O}((1/\mu)n/g)$, where $g$ is the gap size. Therefore, Step (i) of the algorithm takes time $\widetilde{O}((1/\mu)n/g)$. In Step (ii), we use the $3 + \epsilon$ approximation algorithm for $\Delta$-neighboring metric estimation to approximate the distances between the windows. The running time of each oracle invocation is at most $O(l^2)$ since the length of windows is at most $l$. Also, there are $\widetilde{O}((1/\mu)n/g)$ points in this metric estimation instance. Moreover, according to Lemma 4.3, we can set $\Delta = O(\log n(1/\mu)\delta n/g)$. Therefore, due to Corollary 3.11, the total running time of Step (ii), assuming $\mu = O(\epsilon)$, is equal to

$$\widetilde{O}((l^2(n/g)(\delta n/g)^{2/3} + (n/g)(\delta n/g))\text{poly}(1/\epsilon)).$$

Note that $g = l/\gamma$. By assigning $l = n^{1-\beta}$, the overall running time of Step (ii) is

$$\widetilde{O}((n^{2-\beta/3}\delta^{2/3}\gamma^{5/3} + \delta n^{2\beta}\gamma^2)\text{poly}(1/\epsilon)).$$

Step (iii) takes time $O(n + \Delta \cdot \min(|W_1|, |W_2|))$ due to Lemma 4.1, and thus the running time of this step is $\widetilde{O}(n + ((1/\mu)\delta n/g)((1/\mu)n/g) = \widetilde{O}(n + (1/\epsilon)^2(\gamma n^{\beta})(\delta\gamma n^{\beta}))$. Thus, the overall running time of the algorithm up to Step (iii) is

$$\widetilde{O}((n^{2-\beta/3}\delta^{2/3}\gamma^{5/3} + n + \delta\gamma^2 n^{2\beta})\text{poly}(1/\epsilon)).$$

By assigning $\beta = 6/7$, the running time of the algorithm becomes

$$\widetilde{O}(n^{2-2/7}(\delta^{2/3}\gamma^{5/3} + \delta\gamma^2)\text{poly}(1/\epsilon)).$$

Recall that $\delta > n^{-2/21}$. By choosing $\epsilon' = \epsilon/5$ and $\gamma = (\epsilon'\delta)^{-1}$, the overall running time of the algorithm becomes $O(n^{2-4/21}\text{poly}(1/\epsilon))$. What remains is to show that assigning such values for $l$ and $\gamma$ gives the $3 + \epsilon$ approximation factor. Due to Lemma 4.3, there exists a window-compatible transformation of $s_1$ into $s_2$ with respect to $W_1$ and $W_2$ that has at most $((1 + 2\mu)\delta + 1/\gamma)n + 2l$ operations. In other words, restricting the transformation to be window compatible with respect to $W_1$ and $W_2$ leads to an additive error of at most $((1 + 2\mu)\delta + 1/\gamma)n + 2l$. Therefore, a $(3 + \epsilon')$-approximation of the distances between the windows in Step (iii) of the algorithm gives us a transformation with at most $(\delta n)(3 + \epsilon') + (2\mu\delta + 1/\gamma)n + 2l$ operations. This can be simplified as follows:

$$\begin{aligned}
(\delta n)(3+\epsilon') + (2\mu\delta + 1/\gamma)n + 2l &\le 3\delta n + \epsilon'\delta n + 2\mu\delta n + \frac{n}{\gamma} + 2l \\
&\le \left(3\delta + \epsilon'\delta + 2\mu\delta + \frac{1}{\gamma} + 2n^{-6/7}\right) n \\
&\le \left(3 + \epsilon' + 2\mu + \frac{1}{\delta\gamma} + \frac{2n^{-6/7}}{\delta}\right)\delta n \\
&\le \left(3 + \epsilon' + 2\mu + \epsilon' + 2n^{-16/21}\right)\delta n && \delta > n^{-2/21} \\
&\le (3 + 3\epsilon' + 2\mu)\delta n && \text{for every } n > \left(\frac{2}{\epsilon'}\right)^{21/16} \\
&\le (3 + \epsilon)\delta n.
\end{aligned}$$

In the last line, we fix $\epsilon' = \mu = \epsilon/5$. Therefore, the algorithm finds a window-compatible transformation of $s_1$ into $s_2$ with respect to $W_1$ and $W_2$ that is $(3+\epsilon)$-approximation and runs in quantum time $O(n^{2-4/21}\text{poly}(1/\epsilon))$. □

Recall that, using the technique of guess and multiply, we design an algorithm for edit distance by our algorithm for $\delta$-bounded edit distance, with similar running time and approximation factor.

Theorem 4.5. *There exists a quantum algorithm that solves edit distance within an approximation factor of* $3 + \epsilon$ *in time* $\widetilde{O}(n^{2-4/21}\text{poly}(1/\epsilon))$.

Proof. Let opt be the edit distance between the two strings. We can check if opt $= 0$ in time $O(n)$. Assume that opt $\ge 1$. We guess a value $\rho$ for opt by iterating through different multiplicative ranges from 1 to $n$. Let $\epsilon' = \epsilon/9$. In particular, in every step $i \ge 0$ we guess a range $[\delta n, (1 + \epsilon')\delta n)$ for opt, where $\delta = (1+\epsilon')^i/n$, and run the algorithm of Lemma 4.4 with parameters $\epsilon'$ and $(1+\epsilon')\delta$. Note that at each step we can verify whether the output of the algorithm is a valid transformation or not. We get the first valid transformation as soon as opt lies within the range of our guess. This valid transformation is of size at most $(3+\epsilon')(1+\epsilon')\delta$, which is no more than $(3+\epsilon)\delta n$. Also, there

are at most $\log_{1+\epsilon'}(n) \in \widetilde{O}(1/\epsilon)$ ranges for which we run the algorithm of Lemma 4.4. Hence, the overall time for the search is $\widetilde{O}(n^{2-4/21}\text{poly}(1/\epsilon))$. □

## 5 BOOTSTRAPPING

Recall that in Section 2.3, we described our bootstrap algorithm that uses itself as the oracle of the metric distance algorithm. Here, we compute the time complexity and the approximation factor of the algorithm.

Theorem 5.1. *There exists an $\widetilde{O}(n^{2-(5-\sqrt{17})/3+\epsilon}\text{poly}(1/\epsilon))$-time quantum algorithm that approximates edit distance within a factor $\mathrm{e}_{\mathrm{e}}(\epsilon) = O(1/\epsilon)^{O(\log 1/\epsilon)}$.*

Proof. The algorithm is presented in Section 2.3. Here we prove the claimed time complexity and approximation factor. We first solve $\delta$-bounded edit distance. We can assume $\delta > n^{-(5-\sqrt{17})/6+\epsilon/2}$; otherwise, we use the $O(n+\delta^2 n^2)$ of Landau et al. [45]. Suppose the time complexity of our algorithm for the edit distance problem is $\mathrm{t}_{\mathrm{e}}(\epsilon) = \widetilde{O}(n^{2-\phi_\epsilon}\text{poly}(1/\epsilon))$ and the time complexity of our algorithm for the bounded edit distance problem is $\widetilde{O}((1/\delta)n^{2-3\phi_\epsilon/2}\text{poly}(1/\epsilon))$. Notice that the total number of windows is equal to $\widetilde{O}((1/\mu)n/g)$ and thus Step (i) of the algorithm can be easily done in time $\widetilde{O}((1/\mu)n/g)$. In Step (ii), we use the $O(1/\epsilon)$ approximation algorithm of $\Delta$-neighboring metric estimation to approximate the distances of the windows. Using Lemma 4.3, we set $\Delta = \widetilde{O}((1/\mu)\delta n/g)$. Moreover, we use our algorithm of edit distance $\mathcal{A}(2\epsilon)$ recursively for the oracle of metric estimation. Notice that the length of every window is at most $l$. Furthermore, the number of points in the metric is equal to the number of windows, namely $\widetilde{O}((1/\mu)n/g)$. Note that the running time and query complexity of the $O(1/\epsilon)$ algorithm of $\Delta$-neighboring metric estimation are $\widetilde{O}(n\Delta\text{poly}(1/\epsilon))$ and $\widetilde{O}(n\Delta^{3/2+\epsilon}\text{poly}(1/\epsilon))$, respectively. Therefore, the total running time of this step is

$$\widetilde{O}((\mathrm{t}_{\mathrm{e}}(2\epsilon)(l)\cdot(n/g)(\delta n/(g\mu))^{1/2+\epsilon} + (n/g)(\delta n/(g\mu)))\text{poly}(1/\epsilon)).$$

Since $l = O(n^{1-\beta_\epsilon})$ and $n/g = O(\gamma n/l) = O(\gamma n^{\beta_\epsilon})$, and assuming $\mu = \epsilon$, the total running time is equal to

$$\widetilde{O}((n^{(1-\beta_\epsilon)(2-\phi_{2\epsilon})+(3/2)\beta_\epsilon+\epsilon\beta_\epsilon}\delta\gamma^{(3/2)+\epsilon} + n^{2\beta_\epsilon}\delta\gamma^2)\text{poly}(1/\epsilon)).$$

Step (iii) takes time $O(n+\Delta\cdot\min(|W_1|,|W_2|))$ due to Lemma 4.1 and thus the running time of this step is $O(n+n^{2\beta}\delta\gamma^2)$. Thus, if we set $\beta_\epsilon = (\sqrt{17}-1)/4+\epsilon$, $\phi_\epsilon = 1-\beta_\epsilon$, and $\gamma = (\epsilon\delta)^{-1}$, the running time of the algorithm for $\delta$-bounded edit distance would be $\widetilde{O}((1/\delta)n^{2-(5-\sqrt{17})/2+3\epsilon/2}\text{poly}(1/\epsilon))$. Assuming $\delta > n^{-(5-\sqrt{17})/6+\epsilon/2}$, the running time of our algorithm for $\delta$-bounded edit distance is equal to $\widetilde{O}(n^{2-(5-\sqrt{17})/3+\epsilon}\text{poly}(1/\epsilon))$. Using the technique of guess and multiply, the running time of the algorithm for the edit distance problem is also

$$\widetilde{O}(n^{2-(5-\sqrt{17})/3+\epsilon}\text{poly}(1/\epsilon)) = O(n^{1.708}).$$

To compute the approximation ratio, we should first compute the number of nested levels we use in the algorithm itself. In the $i$th recursion, we use $\mathcal{A}(2^{i-1}\epsilon)$, as it works better than an $O(n^2)$ algorithm. Thus, we have

$$2-(5-\sqrt{17})/3+2^{i-1}\epsilon<2 \implies 2^{i-1}\epsilon<(5-\sqrt{17})/3 \implies 2^i<C/\epsilon \implies i<\log_2(1/\epsilon)+O(1),$$

where $C$ is a constant independent of $n$. Hence, we have at most $\log(1/\epsilon)+O(1)$ levels of recursion. We know that $\mathrm{e}_{\mathrm{e}}(\epsilon) = \mathrm{e}_{\mathrm{m}}(\epsilon)\mathrm{e}_{\mathrm{e}}(2\epsilon)+\epsilon$, because if we get an $\alpha$ approximate of the optimal window-compatible transformation, it is an $\alpha+\epsilon$ approximate of the optimal solution, as discussed

in Lemma 4.4. We also know that $\mathsf{e}_\mathsf{m}(\epsilon) = O(1/\epsilon)$. Hence, we can compute the approximation factor as follows:

$$\mathsf{e}_\mathsf{e}(\epsilon) = \mathsf{e}_\mathsf{m}(\epsilon)\mathsf{e}_\mathsf{e}(2\epsilon) + \epsilon \leq (C/\epsilon)\mathsf{e}_\mathsf{e}(2\epsilon) = \frac{C^i}{1 \cdot 2 \cdot \ldots \cdot 2^{i-1} \cdot \epsilon^i}$$
$$= \frac{(1/\epsilon)^{C'}}{(1/\epsilon)^{(i-1)/2} \cdot \epsilon^i} = O(1/\epsilon)^{O(\log 1/\epsilon)},$$

where $C'$ is also a constant independent of $n$. This completes the proof. □

At last, we discuss why the exponent of our algorithm converges to $2 - (5 - \sqrt{17})/3$. Recall that the recursive formula for computing the running time of the algorithm is

$$\widetilde{O}(n^{2-\phi_\epsilon}\mathsf{poly}(1/\epsilon)) = \widetilde{O}((n^{(1-\beta_\epsilon)(2-\phi_{2\epsilon})+(3/2)\beta_\epsilon+\epsilon\beta_\epsilon}\delta\gamma^{(3/2)+\epsilon} + n^{2\beta_\epsilon}\delta\gamma^2)\mathsf{poly}(1/\epsilon)).$$

Intuitively, the running time is the maximum of two terms, and the best is when these terms are equal. Thus, when $\epsilon \to 0$, we can roughly tell: $2 - (3/2)\phi_0 = 2\beta_0 = (1 - \beta_0)(2 - \phi_0) + 1.5\beta_0$. This equation has only one positive answer, which is $\beta_0 = (\sqrt{17} - 1)/4$ and $\phi_0 = 4(1 - \beta_0)/3$; therefore, the exponent is equal to $2 - \phi_0 = 2 - (5 - \sqrt{17})/3$.

## 6 APPROXIMATING EDIT DISTANCE IN MAPREDUCE

Edit distance has been studied in parallel and distributed models since the 1990s. However, the sequential nature of the dynamic programming solution makes it difficult to parallelize; therefore, most of these solutions are slow or require lots of memory/communication. Using our framework, we give a somewhat balanced parallel algorithm for the edit distance problem in the MapReduce model. More precisely, we give a $(1+\epsilon)$-approximation algorithm that uses $O(n^{8/9})$ machines, each with a memory of size $O(n^{8/9})$. Moreover, our algorithm runs in a logarithmic number of rounds and has time complexity $O(n^{1.704})$ on one machine, which is truly subquadratic. The overall communication and total memory of our algorithm are also truly subquadratic, due to the sublinearity of the number of machines and the memory of each machine.

Our algorithm is significantly more efficient than previous PRAM algorithms, for instance [9], in terms of the number of machines, the overall memory, and the overall communication. In addition, this is the first result of its kind for edit distance in the MapReduce model. Although this subject has been studied before, previous studies targeted a different aspect of the problem, such as giving a heuristic algorithm, an algorithm for inputs from a particular distribution model, or an algorithm for edit distance between all pairs of several strings [39].

We begin by stating some of the MapReduce notions and definitions in Section 6.1 and next explain our algorithm is Section 6.2.

### 6.1 MapReduce Basics

In this section, we give a brief overview of the MapReduce setting and later show how our framework can be used to design a MapReduce algorithm for edit distance.

In the MapReduce model, an algorithm consists of several rounds. Each round has a mapping phase and a reducing phase. Every unit of information is represented in the form of a $\langle key; value\rangle$ pair in which both key and value are strings. The input, therefore, is a sequence of $\langle key; value\rangle$ pairs specifying the input data and their corresponding positions. For instance, in the case of edit distance, we assume the input pairs are either in the form of $\langle (s_1, i); s_1[i]\rangle$ or $\langle (s_2, i); s_2[i]\rangle$, where the value represents a character, and the key shows the position of this character in either $s_1$ or $s_2$.

Each round of a MapReduce algorithm is performed as follows: every single input pair is given to a mapper separately, and depending on the mapping algorithm, a sequence of $\langle key; value\rangle$s is generated with respect to the input key. Note that the mappers have to be *stateless* in the sense that the output of every mapper is only dependent on the single $\langle key; value\rangle$ pair given to it. Since the mappers are stateless, parallelism in the mapping phase is straightforward; all the inputs are evenly distributed between the machines. Moreover, there is no limit on the types of the $\langle key; value\rangle$ outputs that the mappers generate. Once *all* the mapper jobs are finished, the reducers start to run. Let $\mathcal{K}$ be the set of all keys generated by the mappers in the mapping stage. In the reducing stage, every $key \in \mathcal{K}$ along with all its associated values is given to a single machine. Note that there is no limit on the number of keys generated in the mapping phase as long as all the outputs together fit in the total memory of all machines. However, the values associated with every key should fit in the memory of a single machine since all such values are processed at once by a single reducer. Every reducer, upon receiving a key and a sequence of values associated to it $\langle key; v_1, v_2, v_3, \ldots, v_l\rangle$, runs a reducer-specific algorithm and generates a sequence of output pairs. Unlike the mapping phase, the output keys of a reducer should be identical to the input key given to them. Moreover, the reducers are not stateless since they have access to all values of a key at once, but they can only access their given key and the values associated with it and should be regardless of the other $\langle key; value\rangle$ pairs generated in the mapping phase. Similar to the mapping phase, the total size of the outputs generated by all reducers should not exceed the total memory of all machines together. In addition to this, the total outputs of a reducer should not be more than its memory. Once *all* reducers finish their jobs, the outputs are fed to the mappers for the next round of the algorithm.

For a problem with input length $n$, the goal is to design a MapRuduce algorithm running on $N_p$ machines each having a memory of $N_m$. $N_p$ and $N_m$ have to be sublinear in $n$ since the input is assumed to be huge in this setting. Moreover, since the overhead of a MapReduce round is time-consuming, the number of MapReduce rounds of the algorithms should be small (either constant or polylogarithmic). Many classic computational problems have been studied in the MapReduce setting. For instance, Karlo et al. [40] provide a MapReduce algorithm to compute an MST of a graph with a sublinear number of machines and a sublinear memory for every machine. Lattanzi et al. [46] design a filtering method and, based on that, provide MapReduce algorithms for fundamental graph problems such as maximal matchings, weighted matchings, vertex cover, edge cover, and minimum cuts.

We show in Section 6.2 that using $O(n^{8/9})$ machines and $O(n^{8/9})$ memory on each machine, one can design a MapReduce algorithm for edit distance that runs in $O(\log n)$ MapReduce rounds. Moreover, the running time of the algorithm is subquadratic.

### 6.2 Edit Distance in MapReduce

Our solution for approximating edit distance in MapReduce uses the same framework explained in Section 4. Therefore, we solve the problem by solving the $\delta$-bounded edit distance problem several times. The difference is that here we solve all of these subproblems simultaneously. This only imposes a multiplicative factor of $O((1/\epsilon)\log n)$ to the number of machines and a multiplicative factor of $1+\epsilon$ to the approximation factor; hence, in the following, we focus on solving the $\delta$-bounded edit distance problem.

We use two different approaches for large $\delta$s and small $\delta$s. For large $\delta$s, we use our framework and compute the edit distance between some pairs of windows of $s_1$ and $s_2$ all at once. For small $\delta$s, though, we use a new method based on (min, +) matrix multiplication, also known as distance multiplication. We denote (min, +) matrix multiplication by $\star$. We separate the large and the small $\delta$s with a critical value-based $N_p$[9] (i.e., the number of machines).

[9]For $N_p = n^{8/9}$ machines we have $\delta^* = n^{-8/27}$.

For (min, +) matrix multiplication in the MapReduce model, we use a parameterized version of an algorithm of [33].

Theorem 6.1 (Proved in [33]). *For any two $n \times n$ matrices $A$ and $B$ and $0 < x \leq 2$, $A \star B$ can be computed with $n^{3(1-x/2)}$ machines and memory $O(n^x)$ in $1 + \lceil (1 - x/2)/x \rceil$ MapReduce rounds. Moreover, the total running time of the algorithm is $O((1/x)n^3)$.*

Given that we have a chain of matrices to be multiplied instead of just two matrices, we can use Theorem 6.1 to halve the number of matrices in two rounds; therefore, we have the following corollary.

Corollary 6.2 (of Theorem 6.1). *The* (min, +) *multiplication of $n^a$ matrices of size $n^b \times n^b$ can be computed in $2\lceil a \log_2 n \rceil$ rounds of MapReduce with $n^y$ machines for any $0 \leq y \leq a + 3b/2$, with a memory of $O(n^{2(a+3b-y)/3})$ for each machine. Moreover, the running time of the algorithm (for one machine) is $\widetilde{O}(n^{a+3b-y})$.*

Notice that for two $n \times n$ matrices in Corollary 6.2, we have $a = 0$ and $b = 1$; hence, the number of machines is $n^y$ and the memory of each machine is $O(n^{2-2y/3})$, which is the same as Theorem 6.1, where $x = 2 - 2y/3$. Also note that for $0 \leq y \leq a + 3b/2$, we use Theorem 6.1 with $1 \leq x \leq 2$; hence, we can safely ignore all $1/x$ terms.

In Sections 6.2.1 and 6.2.2, we discuss our approach for large $\delta$s and small $\delta$s, respectively. In Section 6.2.3, we discuss the remaining details of our algorithm.

*6.2.1 Our Approach for Large $\delta$s.* The overall idea of our solution for large $\delta$s is to use our framework as follows: we first construct some windows for each string, then we find the edit distance between some pairs of windows, and afterward we find a window-compatible transformation, which is a good approximation to the desired edit distance between two input strings.

The first step of our approach is to find the edit distance between *useful pairs* of windows, which are in fact the pairs with an edit distance of at most $\Delta = \widetilde{O}((1/\mu)\delta n/g)$. Previously, we found an approximated edit distance between all pairs of windows using $\Delta$-neighboring metric estimation. Here, we find the distances between useful pairs of windows by running several instances of the naïve DP-based algorithm simultaneously on different machines.

We find the edit distance between useful pairs of windows in the first round. To do this, we give some pairs of windows to a machine and use the naïve DP-based algorithm to find the edit distance between them. In the next round, we combine the results of the first round to find the best window-compatible transformation. The second round is similar to Lemma 4.1, which runs on only one machine.

We have the following lemma for large $\delta$s (or small $\alpha$s). To simplify the notation, let $\delta = n^{-\alpha}$.

Lemma 6.3. *For a $x$, $0 \leq x \leq 7/6$, we can solve the $\delta$-bounded edit distance problem for*

- *$0 \leq x \leq 13/20$ and $\alpha \leq 3(x+1)/16$ with $n^x$ machines, and $O((1/\epsilon)^3 n^{(11-5x)/8+\epsilon'})$ memory for each machine in time $O((1/\epsilon)^3 n^{(35-13x)/16})$ (for one machine), and for*
- *$13/20 \leq x \leq 7/6$ and $\alpha \leq 2(4-x)/21$ with $n^x$ machines, and $O((1/\epsilon)^3 n^{2(4-x)/7+\epsilon'})$ memory for each machine in time $O((1/\epsilon)^3 n^{(50-23x)/21})$ (for one machine)*

*in two MapReduce rounds, where $\epsilon' > 0$ is an arbitrary constant.*

Proof. We already stated the sketch of our algorithm. To analyze the algorithm, we define and set some parameters carefully.

Recall that we used three parameters of $\beta$, $\mu$, and $\gamma$ to construct the windows of length at most $l = \lfloor n^{1-\beta} \rfloor$ with a gap size $g = \lfloor l/\gamma \rfloor$ for each of the two input strings. Lemma 4.3 states that given $\text{edit}(s_1, s_2) \leq \delta n = n^{1-\alpha}$, there exists a window-compatible transformation with at most

$((1+\mu)\delta + 1/\gamma)n + 2l$ operations. To keep the approximation factor as small as $1+\epsilon$, we should have $n/\gamma \ll \delta n$, $\mu = O(\epsilon)$, and $2l \ll \delta n$. Setting $\gamma = 1/\delta\epsilon$, $\mu = \epsilon/5$, and $\beta > \alpha$ suffice. By doing this, the number of windows for each string is at most

$$|W_1|, |W_2| = \widetilde{O}((1/\mu)n\gamma/l) = \widetilde{O}((1/\epsilon)^2 n^{\alpha+\beta}).$$

In the first round, we find the edit distance between all useful pairs of windows. By Lemma 4.3, the number of such useful pairs is at most

$$O(\log n(1/\mu)\delta n/g) \cdot min(|W_1|, |W_2|) = \widetilde{O}((1/\epsilon)^3 n^{\alpha+2\beta}).$$

Therefore, if we have $n^x$ machines, every machine gets $O((1/\epsilon)^3 n^{\alpha+2\beta-x})$ pairs. The edit distance between a pair of windows can be computed in time at most $O(l^2)$ and memory at most $O(l)$ where $l = \lfloor n^{1-\beta} \rfloor$. Hence, the memory of each machine in round 1 is $O((1/\epsilon)^3 n^{1+\alpha+\beta-x})$. Moreover, the time complexity of each machine in this round is $O((1/\epsilon)^3 n^{2+\alpha-x})$.

In the second round, we only use one machine to combine the results of the first round. This machine has to get edit distance between all pairs from all machines; hence, it needs $O((1/\epsilon)^3 n^{\alpha+2\beta})$ memory. The time complexity of this round is also $O((1/\epsilon)^3 n^{\alpha+2\beta})$.

By setting $\beta = \alpha + \epsilon'/2$ for an arbitrary constant $\epsilon' > 0$ and setting $\alpha$ as stated in the lemma, we get the desired result. □

*6.2.2 Our Approach for Small δs.* The other side of the edit distance problem is the case when the two given strings are similar. In this case, if we try to use our framework, we would encounter too many windows, and this exceeds the time and memory given to the algorithm. Previously, in this case, we used the algorithm of Landau et al. [45] with running time $O(n + d^2)$. This solution cannot (trivially) become parallel. Here, we instead use a novel approach based on (min, +) matrix multiplication. We again use the fact that a character $c_1$ from $s_1$ can only be transformed (with no change or a substitution) to a character $c_2$ in $s_2$ only if their positions differ by at most $\text{edit}(s_1, s_2)$ (Corollary 1 of [57]).

Let $d(i, j+1, i', j'+1)$ be equal to $\text{edit}(s_1[i, j], s_2[i', j'])$. We have the following lemma.

Lemma 6.4. *For an arbitrary $k$, $i < k \leq j$, we have*

$$d(i, j+1, i', j'+1) = \min_{i'-1 \leq k' \leq j'} \Big\{ d(i, k+1, i', k'+1) + d(k+1, j+1, k'+1, j+1) \Big\}.$$

Proof. For a fixed $k'$, $i'-1 \leq k' \leq j'$, we can construct a transformation from $s_1[i, j]$ to $s_2[i', j']$ using two transformations: one from $s_1[i, k]$ to $s_2[i', k']$ and the other from $s_1[k+1, j]$ to $s_2[k'+1, j']$; therefore,

$$\text{edit}(s_1[i, j], s_2[i', j']) \leq \text{edit}(s_1[i, k], s_2[i', k']) + edit(s_1[k+1, j], s_2[k'+1, j']),$$

which means

$$d(i, j+1, i', j'+1) \leq d(i, k+1, i', k'+1) + d(k+1, j+1, k'+1, j+1).$$

Therefore, for all $k'$s we have

$$d(i, j+1, i', j'+1) \leq \min_{i'-1 \leq k' \leq j'} \Big\{ d(i, k+1, i', k'+1) + d(k+1, j+1, k'+1, j+1) \Big\}.$$

Moreover, if we define $k^*$ as the largest index that a character from $s_1[i, \ldots, k]$ transfers into $s_2[k^*]$ in an optimal transformation, or $k^* = i'-1$ if no such index exists, we have

$$\text{edit}(s_1[i, j], s_2[i', j']) \leq \text{edit}(s_1[i, k], s_2[i', k^*]) + edit(s_1[k+1, j], s_2[k^*+1, j']);$$

therefore,

$$d(i, j+1, i', j'+1) = d(i, k+1, i', k^*+1) + d(k+1, j+1, k^*+1, j+1),$$

which completes the proof. □

Moreover, computing $d(i, j+1, i', j'+1)$ is useful only when $|i-i'| \leq d$ and $|j-j'| \leq d$ (Corollary 1 of [57]); therefore, for a fixed $i$ and $j$, all of these *useful values* form a $(2\delta n+1) \times (2\delta n+1)$ matrix, namely $D^{i,j}$. Rewriting Lemma 6.4 using matrices, we have the following corollary.

Corollary 6.5 (of Lemma 6.4). *For an arbitrary $k$, $i \leq k \leq j$, we have $D^{i,j} = D^{i,k} \star D^{k,j}$, where $\star$ is the* $(\min, +)$ *matrix multiplication operator.*

Notice that $\text{edit}(s_1, s_2) = d(1, |s_1|+1, 1, |s_2|+1)$, which is an element of $D^{1,|s_1|}$. To compute this matrix, we do as follows: for a parameter $y$, $0 \leq y \leq 1$, which we'll fix later, we partition $s_1$ into $n^y$ substrings of length at most $n^{1-y}$. Each of these substrings has a matching substring in $s_2$ with a length at most $n^{1-y}+2\delta n$. Using the naïve DP-based algorithm, we construct a $(2\delta n+1) \times (2\delta n+1)$ matrix for each of these $n^y$ substrings in the first round. The matrices are $D^{1,t}, D^{t+1,2t}, \ldots, D^{(\lceil |s_1|/t \rceil -1)t+1, |s1|}$, where $t = n^{1-y}$. By Corollary 6.5 we have

$$D^{1,|s_1|} = D^{1,t} \star D^{t+1,2t} \star \cdots \star D^{(\lceil |s_1|/t \rceil -1)t+1, |s1|}.$$

Therefore, we obtain the result in remaining rounds by the matrix multiplication algorithm of Corollary 6.2.

Lemma 6.6. *We can solve the $\delta$-bounded edit distance problem for*

- *$0 \leq x \leq 13/20$ and $\alpha \geq 3(x+1)/16$ with $n^x$ machines, and $O(n^{(11-5x)/8})$ memory of each machine in time $O(n^{(51-29x)/16})$ (for one machine), and for*
- *$13/20 \leq x \leq 7/6$ and $\alpha \geq 2(4-x)/21$ with $n^x$ machines, and $O(n^{2(4-x)/7})$ memory of each machine in time $O(n^{(58-25x)/21})$ (for one machine)*

*in $O(\log n)$ MapReduce rounds.*

Proof. We already described our algorithm. Here, we analyze the described algorithm in more detail. In the first round, constructing the full matrix is a time-consuming process for one machine; therefore, we break this job into $n^t$ parts. More precisely, we partition rows of the solution matrix into $n^t$ parts and give the task of computing each part to one machine. Therefore, in the first round, the number of machines is equal to $n^{y+t} = n^x$. The memory of each machine is the maximum of its input size, its running memory, and its output size, which are equal to $2n^{1-y}+2d$, $O(n^{1-y})$, and $n^{2-2\alpha-t}$, respectively. The time complexity of one machine using the DP-based algorithm is $O(n^{1-y} \cdot n^{1-y} \cdot n^{1-\alpha-t}) = O(n^{3-2y-\alpha-t})$.

The second part of the algorithm is analogous to Corollary 6.2, where $a = y$ and $b = 1-\alpha$; therefore, if the number of machines is $n^x$, the memory of each machine is $n^{2(y+3(1-\alpha)-x)/3}$. Moreover, the time complexity of one machine is $\widetilde{O}(n^{3-3\alpha+y-x})$.

Setting $y = (6\alpha+2x-3)/5$ and $t = x-y$ gives us the desired result. Also, note that the range of $x$ is consistent with Corollary 6.2. □

*6.2.3 Conclusion.* We compute edit distance by solving the $\delta$-bounded edit distance problems for several $\delta$s in parallel. For each $\delta = n^{-\alpha}$ we use the appropriate MapReduce algorithm based on the value of $x$ and $\alpha$. When all subproblems are finished, we also have a final round for combining the results of these subproblems to obtain the final (approximated) edit distance. Therefore, the desired MapReduce $1+\epsilon$ approximation algorithm for edit distance is as follows.

Theorem 6.7. *We can solve the edit distance problems in the MapReduce model in at most $O(\log n)$ MapReduce rounds with $\widetilde{O}((1/\epsilon)n^x)$ machines and for*

- *$0 \leq x \leq 13/20$ with a memory of at most $O((1/\epsilon)^3 n^{(11-5x)/8+\epsilon'})$ for one machine in time $O(n^{(51-29x)/16})$ (for one machine), and for*

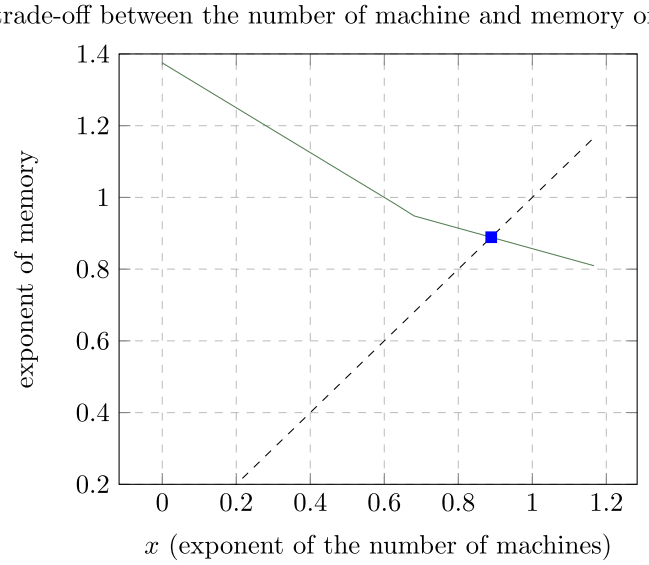



Fig. 7. The trade-off between the number of machines and memory of each machine is shown. In $x = 8/9$ the maximum of the number of machines and the memory of each machine is minimized.

- $13/20 \le x \le 7/6$ *with a memory of at most* $O((1/\epsilon)^3 n^{2(4-x)/7+\epsilon'})$ *in time* $O(n^{(58-25x)/21})$ *(for one machine).*

Proof. We solve the problem for $\delta = 0$ and $\delta = (1+\epsilon/3)^k/n$ for $0 \le k \le O((1/\epsilon)\log n)$ in parallel machines. For each subproblem, we use $\lceil n^x \rceil$ machines.

In the zero case, we only check whether $s_1 = s_2$ or not. This can be done with at most $n^{1-x}$ memory and $O(n^{1-x})$ for each machine. We handle other subproblems by Lemmas 6.3 or 6.4.[10] Therefore, the memory of each machine is at most $O((1/\epsilon)^3 n^{(11-5x)/8+\epsilon'})$ for $0 \le x \le 13/20$ and $O((1/\epsilon)^3 n^{2(4-x)/7+\epsilon'})$ for $13/20 \le x \le 7/6$.

The time complexity of each machine is the maximum time of Lemmas 6.3 and 6.4, which is at most $O(n^{(51-29x)/16})$ for $0 \le x \le 13/20$ and $O(n^{(58-25x)/21})$ for $13/20 \le x \le 7/6$. The number of rounds is $O(\log n)$ in total. □

By setting $x = 8/9$, we can minimize the maximum of the number of machines and the memory of each machine. This is shown in Figure 7.

Corollary 6.8. *We can solve the edit distance problems in the MapReduce model with an approximation factor of* $1+\epsilon$ *in* $O(\log n)$ *rounds with* $\widetilde{O}((1/\epsilon)n^{8/9})$ *machines, a memory of* $O((1/\epsilon)^3 n^{8/9+\epsilon'})$ *for each machine, and in time* $O(n^{2-8/27})$ *(for one machine), where* $\epsilon' > 0$ *is an arbitrary constant.*

## 7 OTHER SIMILARITY MEASURES

Edit distance is one of many similarity measures for comparing two strings. Furthermore, it is one of many problems with a simple two-dimensional DP solution. Other measures and similar problems include longest common subsequence (lcs), Fréchet distance (fre), and dynamic time warping (dtw). While the $O(n^2)$ solutions for these problems are very analogous, unfortunately,

[10] In fact, for small $\delta$s we can run our algorithm just once for the largest $\delta$.

our approach does not directly apply to them. In the following, we discuss some reasons behind this difficulty. The update rule of these measures are defined as follows:

$$\begin{aligned}
\mathsf{edit}(i,j) &= min\Big\{\mathsf{edit}(i-1,j)+1, \mathsf{edit}(i,j-1)+1, \mathsf{edit}(i-1,j-1)+(s_1[i]\neq s_2[j])\Big\}\\
\mathsf{lcs}(i,j) &= max\Big\{\mathsf{lcs}(i-1,j), \mathsf{lcs}(i,j-1), \mathsf{lcs}(i-1,j-1)+(s_1[i]\neq s_2[j])\Big\}\\
\mathsf{dtw}(i,j) &= min\Big\{\mathsf{dtw}(i-1,j), \mathsf{dtw}(i,j-1), \mathsf{dtw}(i-1,j-1)\Big\}+dis(i,j)\\
\mathsf{fre}(i,j) &= max\Big\{min\{\mathsf{fre}(i-1,j), \mathsf{fre}(i,j-1), \mathsf{fre}(i-1,j-1)\}, dis(i,j)\Big\}.
\end{aligned}$$

Our framework for approximating edit distance is based on two assumptions. First is the usability of Lemma 4.3, which states that there is a window-compatible solution that is a good approximation to the optimal solution. Second, to use the metric estimation, the desired measure should be a distance function, namely a metric.

Two similarity measures dtw and lcs are not metric; moreover, they cannot be approximated by a metric. For example, for dtw consider $s_1 = a^{2k+1}$, $s_2 = a^k b a^k$, and $s_3 = ab^{2k-1}a$. We have $\mathsf{dtw}(s_1, s_2) = 1$ and $\mathsf{dtw}(s_2, s_3) = 0$, but $\mathsf{dtw}(s_1, s_3) = 2k-1$. Therefore, the triangle inequality does not hold here.

The similarity measure lcs is, in fact, the opposite of a metric function; i.e., for two similar strings, their lcs is large, and for two different strings, their lcs is small. The first property of a distance function does not hold here, for a non-empty string s, $\mathsf{lcs}(s,s) \neq 0$. The other part of our approach where lcs has a drawback is Lemma 4.3. For a window size $l$, one can consider $s_1 = (ab^{l-1}a^{l-1})^t$ and $s_2 = (a^l c^{l-1})t$. We have $\mathsf{lcs}(s_1, s_2) = lt$, but lcs of a window-compatible transformation is at most $t$.

Likewise, approximating lcs in classical computers is also harder than edit. None of the results for approximating edit is shown for lcs, unless when $lcs(s_1, s_2) = \Omega(n)$. Another way around this is to approximate co-lcs instead of lcs, where $\text{co-}\mathsf{lcs}(s_1, s_2) = |s_1| + |s_2| - \mathsf{lcs}(s_1, s_2)$. This measure is very similar to edit distance but without the substitution operation. Using our framework, we can approximate co-lcs with the same approximation factor of $3+\epsilon$ in quantum computers and an approximation factor of $1+\epsilon$ in MapReduce.

Fréchet distance is rather a similarity measure for curves instead of strings. For strings, the problem becomes trivial, i.e., zero for same strings and one for different strings. However, fre on curves has a similar dynamic programming solution to edit. This similarity in solution leads us to consider this problem too. If we study the problem regardless of its geometric properties, i.e., all distances are given as a matrix, we can prove that approximating fre is as hard as computing its exact value.

Theorem 7.1. *If there exists a quantum (or MapReduce) approximation algorithm for Fréchet distance with a constant approximation factor in time $O(n^{2-\epsilon})$, which takes distances as a matrix in the input, there also exists a quantum or MapReduce algorithm that computes the exact Fréchet distance in time $O(n^{2-\epsilon})$.*

Proof. The idea is a gap-producing reduction from the problem to itself. We can do a binary search on the actual value of Fréchet distance; thus, in each step for a threshold $t$, we want to know whether $\mathsf{fre}(a,b) \leq t$ or not. We define a new distance function as

$$dis'(a,b) = \begin{cases} 1 & dis(a,b) \leq t,\\ n^2 & \text{otherwise.}\end{cases}$$

If $\mathsf{fre}(a,b) \leq t$, then we have $\mathsf{fre}'(a,b) = 1$ and $\mathsf{fre}'(a,b) = n^2$ otherwise. Therefore, if we solve the new instance with an approximation algorithm with a constant factor, we can decide whether

$\text{fre}'(a, b) = 1$ or not. Thus, we can decide whether $\text{fre}(a, b) \leq t$ or not. Hence, we can find the exact value of Fréchet distance by executing the approximation algorithm in $O(\log n)$ round. This argument works for both quantum algorithms and MapReduce algorithms. □

Theorem 7.1 does not rule out the possibility of a subquadratic quantum algorithm or MapReduce algorithm for Fréchet distance, but it states that relaxing the problem in this way does not make the problem easier.

## 8 CONCLUSION AND OPEN PROBLEMS

Indeed, the most important open problem concerning edit distance is whether a subquadratic time algorithm in classical computers can approximate the edit distance of two strings within a constant factor. In this regard, our article proposes the following approach. Suppose we want to approximate the pairwise edit distance between $m$ given strings, each of size $n$, within a constant factor. We call this problem *pairwise edit distance*. A naïve solution for pairwise edit distance has running time $O(m^2n^2)$. Obviously, any subquadratic time algorithm for approximating edit distance within a constant factor improves upon this running time. In this article, we show that an improvement in the running time of pairwise edit distance also leads to a subquadratic time algorithm for approximating edit distance within a constant factor. We believe this actually opens a new direction for approximating edit distance for classical computers.

In addition to this, our work gives rise to a number of questions that we believe are important to study in future work:

- *How efficiently can we approximate metric estimation in classical computers with a subquadratic number of queries, when the distance function is edit distance?*
- *Is there subquadratic quantum algorithms that approximate other similarity measures within a constant factor, lcs in particular?*
- *Can a quantum algorithm approximate the edit distance of two strings within a constant factor in near-linear time?*
- *Is it possible to show a non-trivial lower bound on the quantum computational complexity of computing edit distance?*

## ACKNOWLEDGMENTS

We would like to thank Andrew Childs, Omid Etesami, Salman Beigi, and Mohammad Ali Abam for their comments on an earlier version of the article.

## REFERENCES

[1] Andris Ambainis. 2002. Quantum lower bounds by quantum arguments. *J. Comput. System Sci.* 64, 4 (2002), 750–767. https://doi.org/10.1006/jcss.2002.1826

[2] Alexandr Andoni, Piotr Indyk, and Robert Krauthgamer. 2009. Overcoming the $\ell_1$ non-embeddability barrier: Algorithms for product metrics. In *Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'09)*. SIAM, 865–874. https://doi.org/10.1137/1.9781611973068.94

[3] Alexandr Andoni and Robert Krauthgamer. 2010. The computational hardness of estimating edit distance. *SIAM J. Comput.* 39, 6 (April 2010), 2398–2429. https://doi.org/10.1137/080716530

[4] Alexandr Andoni and Robert Krauthgamer. 2012. The smoothed complexity of edit distance. *ACM Trans. Algorithms* 8, 4 (Oct. 2012), Article 44, 25 pages. https://doi.org/10.1145/2344422.2344434

[5] Alexandr Andoni, Robert Krauthgamer, and Krzysztof Onak. 2010. Polylogarithmic approximation for edit distance and the asymmetric query complexity. In *Proceedings of the 51st Annual IEEE Symposium on Foundations of Computer Science (FOCS'10)*. IEEE, 377–386. https://doi.org/10.1109/FOCS.2010.43

[6] Alexandr Andoni, Aleksandar Nikolov, Krzysztof Onak, and Grigory Yaroslavtsev. 2014. Parallel algorithms for geometric graph problems. In *Proceedings of the 46th Annual ACM SIGACT Symposium on Theory of Computing (STOC'14)*. ACM, 574–583. https://doi.org/10.1145/2591796.2591805

[7] Alexandr Andoni and Negev Shekel Nosatzki. 2020. Edit distance in near-linear time: It's a constant factor. arXiv:cs.DS/2005.07678

[8] Alexandr Andoni and Krzysztof Onak. 2012. Approximating edit distance in near-linear time. *SIAM J. Comput.* 41, 6 (2012), 1635–1648. https://doi.org/10.1137/090767182

[9] Alberto Apostolico, Mikhail J. Atallah, Lawrence L. Larmore, and Scott McFaddin. 1990. Efficient parallel algorithms for string editing and related problems. *SIAM J. Comput.* 19, 5 (1990), 968–988. https://doi.org/10.1137/0219066

[10] Arturs Backurs and Piotr Indyk. 2015. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). In *Proceedings of the 47th Annual ACM SIGACT Symposium on Theory of Computing (STOC'15)*. ACM, 51–58. https://doi.org/10.1145/2746539.2746612

[11] Ziv Bar-Yossef, T. S. Jayram, Robert Krauthgamer, and Ravi Kumar. 2004. Approximating edit distance efficiently. In *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS'04)*. IEEE, 550–559. https://doi.org/10.1109/FOCS.2004.14

[12] Tuğkan Batu, Funda Ergun, and Cenk Sahinalp. 2006. Oblivious string embeddings and edit distance approximations. In *Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithm (SODA'06)*. SIAM, 792–801.

[13] Robert Beals. 1997. Quantum computation of Fourier transforms over symmetric groups. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing (STOC'97)*. ACM, 48–53. https://doi.org/10.1145/258533.258548

[14] Aleksandrs Belovs. 2012. Learning-graph-based quantum algorithm for k-distinctness. In *Proceedings of the 53rd Annual IEEE Symposium on Foundations of Computer Science (FOCS'12)*. IEEE, 207–216. https://doi.org/10.1109/FOCS.2012.18

[15] Mahdi Boroujeni, Soheil Ehsani, Mohammad Ghodsi, MohammadTaghi HajiAghayi, and Saeed Seddighin. 2018. Approximating edit distance in truly subquadratic time: Quantum and MapReduce. In *Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'18)*. SIAM, 1170–1189. https://doi.org/10.1137/1.9781611975031.76

[16] Mahdi Boroujeni, Masoud Seddighin, and Saeed Seddighin. 2020. Improved algorithms for edit distance and LCS: Beyond worst case. In *Proceedings of the 31st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'20)*. SIAM, 1601–1620. https://doi.org/10.1137/1.9781611975994.99

[17] Mahdi Boroujeni and Saeed Seddighin. 2019. Improved MPC algorithms for edit distance and Ulam distance. In *The 31st ACM Symposium on Parallelism in Algorithms and Architectures (SPAA'19)*. ACM, 31–40. https://doi.org/10.1145/3323165.3323205

[18] Michel Boyer, Gilles Brassard, Peter Høyer, and Alain Tapp. 1998. Tight bounds on quantum searching. *Fortschritte der Physik* 46, 4–5 (1998), 493–505. https://doi.org/10.1002/(SICI)1521-3978(199806)46:4/5<493::AID-PROP493>3.0.CO;2-P

[19] Joshua Brakensiek, Moses Charikar, and Aviad Rubinstein. 2020. A simple sublinear algorithm for gap edit distance. arXiv:cs.DS/2007.14368

[20] Joshua Brakensiek and Aviad Rubinstein. 2020. Constant-factor approximation of near-linear edit distance in near-linear time. In *Proceedings of the 52nd Annual ACM Symposium on Theory of Computing (STOC'20)*. ACM, 685–698. https://doi.org/10.1145/3357713.3384282

[21] Gilles Brassard, Peter Høyer, Michele Mosca, and Alain Tapp. 2002. Quantum amplitude amplification and estimation. *Contemp. Math.* 305 (2002), 53–74. https://doi.org/10.1090/conm/305/05215

[22] Diptarka Chakraborty, Debarati Das, Elazar Goldenberg, Michal Koucky, and Michael Saks. 2018. Approximating edit distance within constant factor in truly sub-quadratic time. In *Proceedings of the 59th Annual IEEE Symposium on Foundations of Computer Science (FOCS'18)*. IEEE, 979–990. https://doi.org/10.1109/FOCS.2018.00096

[23] Diptarka Chakraborty, Debarati Das, Elazar Goldenberg, Michal Koucký, and Michael Saks. 2020. Approximating edit distance within constant factor in truly sub-quadratic time. *J. ACM* 67, 6, Article 36 (Oct. 2020), 22 pages. https://doi.org/10.1145/3422823

[24] Moses Charikar and Robert Krauthgamer. 2006. Embedding the Ulam metric into $\ell_1$. *Theory Comput.* 2, 11 (2006), 207–224. https://doi.org/10.4086/toc.2006.v002a011

[25] Christoph Dürr, Mark Heiligman, Peter Høyer, and Mehdi Mhalla. 2006. Quantum query complexity of some graph problems. *SIAM J. Comput.* 35, 6 (2006), 1310–1328. https://doi.org/10.1137/050644719

[26] Alireza Farhadi, MohammadTaghi HajiAghayi, Aviad Rubinstein, and Saeed Seddighin. 2020. Asymmetric streaming algorithms for edit distance and LCS. arXiv:cs.DS/2002.11342

[27] Edward Farhi, Jeffrey Goldstone, Sam Gutmann, and Michael Sipser. 1998. A limit on the speed of quantum computation in determining parity. *Phys. Rev. Lett.* 81 (1998), 5442–5444. https://doi.org/10.1103/PhysRevLett.81.5442

[28] Edward Farhi, Jeffrey Goldstone, Sam Gutmann, and Michael Sipser. 1999. Invariant quantum algorithms for insertion into an ordered list. arXiv:quant-ph/9901059

[29] Elazar Goldenberg, Robert Krauthgamer, and Barna Saha. 2019. Sublinear algorithms for gap edit distance. In *Proceedings of the 60th Annual IEEE Symposium on Foundations of Computer Science (FOCS'19)*. IEEE, 1101–1120. https://doi.org/10.1109/FOCS.2019.00070

[30] Elazar Goldenberg, Aviad Rubinstein, and Barna Saha. 2020. Does preprocessing help in fast sequence comparisons? In *Proceedings of the 52nd Annual ACM Symposium on Theory of Computing (STOC'20)*. ACM, 657–670. https://doi.org/10.1145/3357713.3384300

[31] Lov K. Grover. 1996. A fast quantum mechanical algorithm for database search. In *Proceedings of the 28th Annual ACM Symposium on Theory of Computing (STOC'96)*. ACM, 212–219. https://doi.org/10.1145/237814.237866

[32] Bernhard Haeupler, Aviad Rubinstein, and Amirbehshad Shahrasbi. 2019. Near-linear time insertion-deletion codes and $(1+\varepsilon)$-approximating edit distance via indexing. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing (STOC'19)*. ACM, 697–708. https://doi.org/10.1145/3313276.3316371

[33] MohammadTaghi HajiAghayi, Silvio Lattanzi, Saeed Seddighin, and Cliff Stein. 2019. MapReduce meets fine-grained complexity: MapReduce algorithms for APSP, matrix multiplication, 3-SUM, and beyond. arXiv:cs.DS/1905.01748

[34] MohammadTaghi HajiAghayi, Saeed Seddighin, and Xiaorui Sun. 2019. Massively parallel approximation algorithms for edit distance and longest common subsequence. In *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'19)*. SIAM, 1654–1672. https://doi.org/10.1137/1.9781611975482.100

[35] Ramesh Hariharan and V. Vinay. 2003. String matching in $\widetilde{O}(\sqrt{n}+\sqrt{m})$ quantum time. *J. Discrete Algorithms* 1, 1 (2003), 103–110. https://doi.org/10.1016/S1570-8667(03)00010-8

[36] Sungjin Im, Benjamin Moseley, and Xiaorui Sun. 2017. Efficient massively parallel methods for dynamic programming. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing (STOC'17)*. ACM, 798–811. https://doi.org/10.1145/3055399.3055460

[37] Piotr Indyk. 2001. Algorithmic applications of low-distortion geometric embeddings. In *Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science (FOCS'01)*. IEEE, 10–33. https://doi.org/10.1109/SFCS.2001.959878

[38] Stacey Jeffery, Robin Kothari, and Frédéric Magniez. 2013. Nested quantum walks with quantum data structures. In *Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'13)*. SIAM, 1474–1485. https://doi.org/10.1137/1.9781611973105.106

[39] Shagun Jhaver, Latifur Khan, and Bhavani Thuraisingham. 2014. Calculating edit distance for large sets of string pairs using MapReduce. Paper presented at ASE International Conference on Big Data.

[40] Howard Karloff, Siddharth Suri, and Sergei Vassilvitskii. 2010. A model of computation for MapReduce. In *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'10)*. SIAM, 938–948. https://doi.org/10.1137/1.9781611973075.76

[41] Tomasz Kociumaka and Barna Saha. 2020. Sublinear-time algorithms for computing & embedding gap edit distance. In *Proceedings of the 61st Annual IEEE Symposium on Foundations of Computer Science (FOCS'20)*. IEEE, Virtual Conference.

[42] Michal Koucký and Michael Saks. 2020. Constant factor approximations to edit distance on far input pairs in nearly linear time. In *Proceedings of the 52nd Annual ACM Symposium on Theory of Computing (STOC'20)*. ACM, 699–712. https://doi.org/10.1145/3357713.3384307

[43] Robert Krauthgamer and Yuval Rabani. 2009. Improved lower bounds for embeddings into $L_1$. *SIAM J. Comput.* 38, 6 (2009), 2487–2498. https://doi.org/10.1137/060660126

[44] Hari Krovi and Alexander Russell. 2015. Quantum Fourier transforms and the complexity of link invariants for quantum doubles of finite groups. *Comm. Math. Phys.* 334, 2 (2015), 743–777. https://doi.org/10.1007/s00220-014-2285-5

[45] Gad M. Landau, Eugene W. Myers, and Jeanette P. Schmidt. 1998. Incremental string comparison. *SIAM J. Comput.* 27, 2 (1998), 557–582. https://doi.org/10.1137/S0097539794264810

[46] Silvio Lattanzi, Benjamin Moseley, Siddharth Suri, and Sergei Vassilvitskii. 2011. Filtering: A method for solving graph problems in MapReduce. In *Proceedings of the 23rd Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA'11)*. ACM, 85–94. https://doi.org/10.1145/1989493.1989505

[47] François Le Gall. 2014. Improved quantum algorithm for triangle finding via combinatorial arguments. In *Proceedings of the 55th Annual IEEE Symposium on Foundations of Computer Science (FOCS'14)*. IEEE, 216–225. https://doi.org/10.1109/FOCS.2014.31

[48] William J. Masek and Michael S. Paterson. 1980. A faster algorithm computing string edit distances. *J. Comput. Syst. Sci.* 20, 1 (1980), 18–31. https://doi.org/10.1016/0022-0000(80)90002-1

[49] Michael Mitzenmacher and Saeed Seddighin. 2020. Dynamic algorithms for LIS and distance to monotonicity. In *Proceedings of the 52nd Annual ACM Symposium on Theory of Computing (STOC'20)*. ACM, 671–684. https://doi.org/10.1145/3357713.3384240

[50] Ashley Montanaro, Richard Jozsa, and Graeme Mitchison. 2015. On exact quantum query complexity. *Algorithmica* 71, 4 (2015), 775–796. https://doi.org/10.1007/s00453-013-9826-8

[51] Rajeev Motwani and Prabhakar Raghavan. 1995. *Randomized Algorithms*. Cambridge University Press, UK.
[52] Aran Nayebi and Virginia Vassilevska Williams. 2014. Quantum algorithms for shortest paths problems in structured instances. arXiv:quant-ph/1410.6220
[53] Rafail Ostrovsky and Yuval Rabani. 2007. Low distortion embeddings for edit distance. *J. ACM* 54, 5 (Oct. 2007), Article 23, 16 pages. https://doi.org/10.1145/1284320.1284322
[54] Aviad Rubinstein, Saeed Seddighin, Zhao Song, and Xiaorui Sun. 2019. Approximation algorithms for LCS and LIS with truly improved running times. In *Proceedings of the 60th Annual IEEE Symposium on Foundations of Computer Science (FOCS'19)*. IEEE, 1121–1145. https://doi.org/10.1109/FOCS.2019.00071
[55] Aviad Rubinstein and Virginia Vassilevska Williams. 2019. SETH vs approximation. *ACM SIGACT News* 50, 4 (2019), 57–76. https://doi.org/10.1145/3374857.3374870
[56] Peter W. Shor. 1997. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.* 26, 5 (1997), 1484–1509. https://doi.org/10.1137/S0097539795293172
[57] Esko Ukkonen. 1985. Algorithms for approximate string matching. *Inf. Control* 64, 1–3 (1985), 100–118. https://doi.org/10.1016/S0019-9958(85)80046-2

Received April 2018; revised September 2020; accepted January 2021