Approximating Edit Distance in Truly Subquadratic Time: Quantum and MapReduce*†

Mahdi Boroujeni ‡ Soheil Ehsani §‖ Mohammad Ghodsi ‡¶
MohammadTaghi HajiAghayi §‖ Saeed Seddighin §‖

Abstract

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" [21]. Our main result is a quantum constant approximation algorithm for computing the edit distance in truly subquadratic time. More precisely, we give an $O(n^{1.858})$ quantum algorithm that approximates the edit distance within a factor of 7. We further extend this result to an $O(n^{1.781})$ 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 3, with sublinearly many machines and sublinear memory. Also, our algorithm runs in a logarithmic number of rounds.

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.

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 $O(n^2/\log^2 n)$ [29]. The shortcoming of these studies is partly addressed by the work of Backurs and Indyk [7] wherein the authors show 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

*Portions of this research were completed while the first, third, and fifth authors were visitors at the Simons Institute for the Theory of Computing.

†The omitted proofs can be found in the full version of this paper.

‡Sharif University of Technology. Email: safarnejad@ce.sharif.edu, ghodsi@sharif.edu

§University of Maryland. Email: {ehsani,hajiagha}@cs.umd.edu, sseddigh@umd.edu

¶Institute for Research in Fundamental Sciences (IPM).

‖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.

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

example, a human genome consists of almost three 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.* [26] where d is the edit distance between the two strings. Subsequent research improved this to $n^{3/7}$ [8], to $n^{1/3+o(1)}$ [9], to $2^{\widetilde{O}(\sqrt{\log n})}$ [5], and the latest of which provides a polylogarithmic approximation guarantee in subquadratic time [3]. 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 [21]).

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, etc. [10, 11, 16, 22, 25, 28, 33, 34]. However, quantum techniques can only be applied to limited structures. For instance, many classic problems such as sorting or even counting the number of 1's 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, neither to many classic DP-type problems such as finding the lcs (longest common subsequence), dtw (dynamic time wrapping) 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 [17] and Dürr *et al.* [14] 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 quantum setting and perhaps achieving 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 3. 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 paper, we focus on the well-known MapReduce theoretical framework initiated by Karloff, Suri, and Vassilvitskii [24] (and later further refined by Andoni, Nikolov, Onak, and Yaroslavtsev [4]) Designing MapReduce algorithms for simulating sequential dynamic programs for important problems was recently initiated by Im, Moseley, and Sun [20]. 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. 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 first to improve upon the trivial $O(n^2)$ classic 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.

The closest works to our results are [5] and [2]. In particular, they use a space embedding approach from [32] 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. In [6], the authors give a parallel

[2] within a constant factor

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 $O(\mathsf{polylog}(n))$ [24]. The major advantage of our MapReduce algorithm over the algorithm of [6] 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 $O(\log(n))$.

A similar approach is taken in the work of Nayebi *et al.* [31] 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.

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 5. 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* [18] and *amplitude amplification* [13] to improve the lookup time on an unordered set. On the algorithmic side, we benefit from classic 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 $\mathsf{edit}(s_1, s_2)$. By definition, edit distance meets all of the *identity of indiscernibles*[3], *symmetry*[4], and *triangle inequality*[5] properties, thus for any set of strings $\mathcal{M}$, $\langle \mathcal{M}, \mathsf{edit} \rangle$ 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 which 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 [15]. We give two approximation algorithms that solve the metric estimation problem with approximation factors $3 + \epsilon$ and $\mathsf{e_m}(\epsilon) = O(1/\epsilon)$ with $\widetilde{O}(n^{5/3}\mathsf{poly}(1/\epsilon))$ and $\widetilde{O}(n^{3/2+\epsilon}\mathsf{poly}(1/\epsilon))$ oracle queries, respectively. Notice that the running times of the algorithms are $O(n^2\mathsf{poly}(1/\epsilon))$, but the query complexities are subquadratic. This allows us to approximate metrics spaces with sublinear points for which answering an oracle query is time-consuming. We emphasize that our metric estimation results are general and can be used for any metric. In the second step, we show that any algorithm that solves the metric estimation problem within an approximation factor α can be used as a black box to obtain a $1+2\alpha+\epsilon$ 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 $7 + \epsilon$ approximation algorithm for edit distance. Finally, 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 details. 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 is 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 in Sections 3 and 4.

[3] $\mathsf{edit}(s_1, s_2) = 0 \Leftrightarrow s_1 = s_2$.
[4] $\mathsf{edit}(s_1, s_2) = \mathsf{edit}(s_2, s_1)$.
[5] $\mathsf{edit}(s_1, s_2) + \mathsf{edit}(s_2, s_3) \geq \mathsf{edit}(s_1, s_3)$.

[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.

The proofs can be found in the full version of this paper.

- 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 $\mathcal{O}$ 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 $\mathcal{O}$ to access the distances.

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

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 [18] 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 $0 \leq m \leq n$, 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 [18] 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 [12]) *The listing problem can be solved with $O(\sqrt{nm})$ oracle queries via quantum computation.*

The second quantum technique that we use in this paper 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 $\mathsf{par}(f) = \prod_{i \in [n]} f(i)$. In the parity problem, we are given oracle access to f and the goal is to determine $\mathsf{par}(f)$ with as few oracle calls as possible.

Parity

Input: an integer n, and access to an oracle $\mathcal{O}$ 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: $\mathsf{par}(f) = \prod_{i \in [n]} f(i)$.

Of course, if the numbers of -1's or 1's are substantially smaller than n ($o(n)$), 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.* [15], 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 [15]) *The parity problem cannot be solved with fewer than $\Omega(n)$ queries with quantum computation.*

Based on the result of Farhi *et al.* [15], 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 $\mathsf{Cor}(\mathsf{I})$ of the metric estimation and prove that $\mathsf{Cor}(\mathsf{I})$ 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.* [15].

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 $\widetilde{O}(n^{5/3}\mathsf{poly}(1/\epsilon))$ queries.

Theorem 3.2 [restated]. *For any $\epsilon > 0$, there exists a quantum algorithm that solves metric estimation with $\widetilde{O}(n^{5/3}\mathsf{poly}(1/\epsilon))$ queries within an approximation factor of $3+\epsilon$. Moreover, the running time of the algorithm is $\widetilde{O}(n^2\mathsf{poly}(1/\epsilon))$.*

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 lie in the interval $[l, u]$. Therefore, one can divide the distances into $\log_{1+\epsilon/3}(u/l) = \widetilde{O}(\mathsf{poly}(1/\epsilon))$ 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 $\widetilde{O}(n^{5/3}\mathsf{poly}(1/\epsilon))$ queries, yields a $3+\epsilon$ approximation factor algorithm for metric estimation that uses at most $\widetilde{O}(n^{5/3}\mathsf{poly}(1/\epsilon))$ oracle calls.

In what follows, we describe the ideas to solve the problem for a fixed threshold t. The algorithm is explained in details 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 $\mathcal{O}$, 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 of 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.

Table 1: Quality of the approximation algorithms for metric estimation

Approx. factor	$\alpha < 3$	$\alpha = 3+\epsilon$
Number of queries	$\Omega(n^2)$ (Theorem 3.1)	$\widetilde{O}(n^{5/3}\mathsf{poly}(1/\epsilon))$ (Theorem 3.2)
Approx. factor	$\alpha = \mathsf{e_m}(\epsilon)$	$\alpha =$ any constant
Number of queries	$\widetilde{O}(n^{3/2+\epsilon}\mathsf{poly}(1/\epsilon))$ (Theorem 3.4)	$\Omega(n^{3/2})$ (Theorem 3.7)

In addition to Theorem 3.2, we show in Section 3 that with a deeper analysis, one can use the same ideas to further improve the query complexity to $\widetilde{O}(n^{3/2+\epsilon}\mathsf{poly}(1/\epsilon))$ by allowing the approximation guarantee to grow up to $\mathsf{e_m}(\epsilon) = O(1/\epsilon)$.

Theorem 3.4 [restated]. *For any $\epsilon > 0$, there exists a quantum algorithm that solves metric estimation with $\widetilde{O}(n^{3/2+\epsilon}\mathsf{poly}(1/\epsilon))$ queries within an approximation factor of $\mathsf{e_m}(\epsilon) = O(1/\epsilon)$. Moreover, the running time of the algorithm is $\widetilde{O}(n^2\mathsf{poly}(1/\epsilon))$.*

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

2.2 Approximating Edit Distance within a Factor $7+\epsilon$ 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 αd operations[7] where α 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.* [26] 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 and s_2 and guaranteed that the edit distance between the strings is at most $\delta(|s_1|+|s_2|)$ where δ 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 α is the approximation factor of the algorithm. We refer to this subtask of edit distance as *the δ-bounded edit distance* problem.

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

We combine a divide and conquer technique with dynamic programming in order to approximate δ-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 two parameters $0<\beta<1$ and $\gamma>1$. γ is an integer number but β is a real number between 0 and 1. We use β and γ as two parameters of our algorithm, and after the analysis, we show which values for β and γ 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 *window size* and define a window of s_1, as a string of length l over the characters 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: for every $0\le i\le \lfloor\frac{|s_1|-l}{g}\rfloor$, put a window $[ig+1, ig+l]$ (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. 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 is either 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 2.1. *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 (i) all old characters of s_2 are in the windows of S' and (ii) every old character of s_2 which 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 either be 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.

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 $O(n+|W_1||W_2|)$.

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, one can compute an optimal window-compatible transformation of s_1 into s_2 in time*

[7] insertion, deletion, or substitution

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

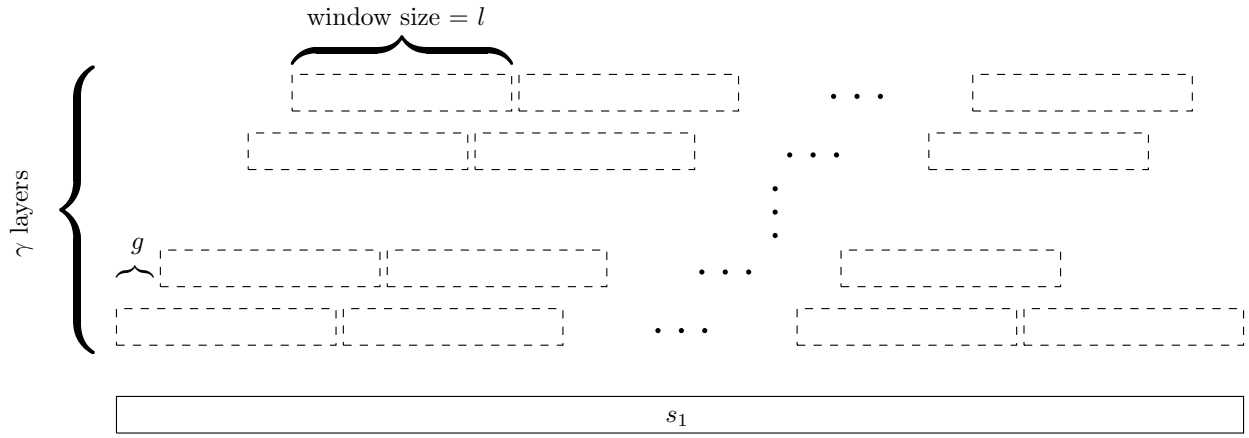
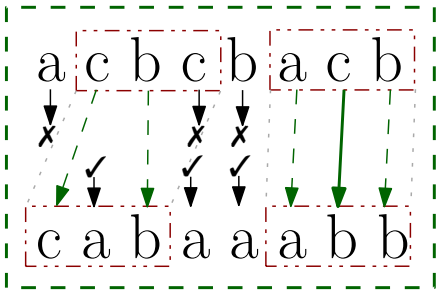
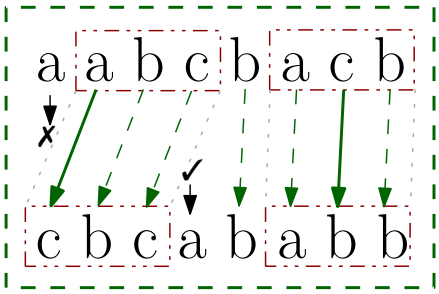


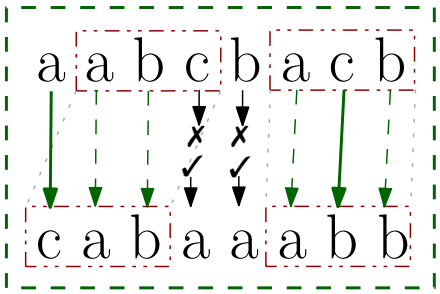
Figure 1: s_1 is shown with a solid rectangle and windows of W_1 are depicted via dashed rectangles.



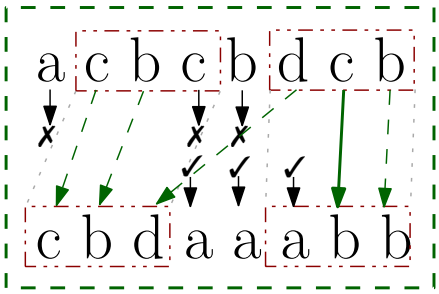
(a) An example of a window-compatible transformation.



(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.

Figure 2: Figures 2a, 2b, 2c, and 2d 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.

$O(n + |W_1||W_2|)$.

Lemma 4.1 shows that window-compatible transformations are easy to find. It also follows from Lemma 4.1 that any α approximation matrix for the edit distances of the windows suffices to find an approximately opti-

mal window-compatible transformation (with the same approximation factor) in time $O(n + |W_1||W_2|)$. 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 $\delta(|s_1| + |s_2|)$.

Lemma 4.2 [restated]. Given that $\mathsf{edit}(s_1, s_2) \leq \delta n$, there exists a window-compatible transformation of s_1 into s_2 with at most $(3\delta + 1/\gamma)n + 2l$ operations.

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 4.2 argues that such a transformation is to some extent optimal. Based on this, we outline our algorithm for δ-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 $\langle \mathcal{M}, \mathsf{edit} \rangle$, 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 metric estimation, find a $3+\epsilon$ approximation solution to the edit distances for every pair of windows (Theorem 3.2).
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 δ-bounded edit distance problem.

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

Lemma 4.3 [restated] There exists a quantum algorithm that solves the δ-bounded edit distance problem within an approximation factor of $7 + \epsilon$ in time $\widetilde{O}(n^{2-1/7}\mathsf{poly}(1/\epsilon))$.

By Lemma 4.3, we can approximate the δ-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 $\mathsf{edit}(s_1, s_2) \leq \delta(|s_1|+|s_2|)$ 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 $\mathsf{edit}(s_1, s_2) \geq 1$. Now, we start with $\rho = 1/n$ and every time run our solution for δ-bounded edit distance with parameter $\delta = \rho$, to find an approximation proof for $\mathsf{edit}(s_1, s_2) = \rho n$. If our algorithm finds a proper transformation with at most $(7\rho+\epsilon)n$ operations, then we report that solution. Otherwise, we know that $\mathsf{edit}(s_1, s_2) > \rho n$, and thus multiply ρ 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 $\widetilde{O}(n^{2-1/7}\mathsf{poly}(1/\epsilon))$. We later refer to this technique as *guess and multiply*.

Theorem 4.1 [restated] There exists a quantum algorithm that solves edit distance within an approximation factor of $7 + \epsilon$ in time $\widetilde{O}(n^{2-1/7}\mathsf{poly}(1/\epsilon))$.

2.3 Improving the Running Time via Bootstrapping So far, we 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. Firstly, instead of using the $3 + \epsilon$ approximation algorithm for metric estimation, we can lose a factor of $\mathsf{e_m}(\epsilon)$ in the approximation and estimate the distances in time $\widetilde{O}(n^{3/2+\epsilon}\mathsf{poly}(1/\epsilon))$ (Theorem 3.4). 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.

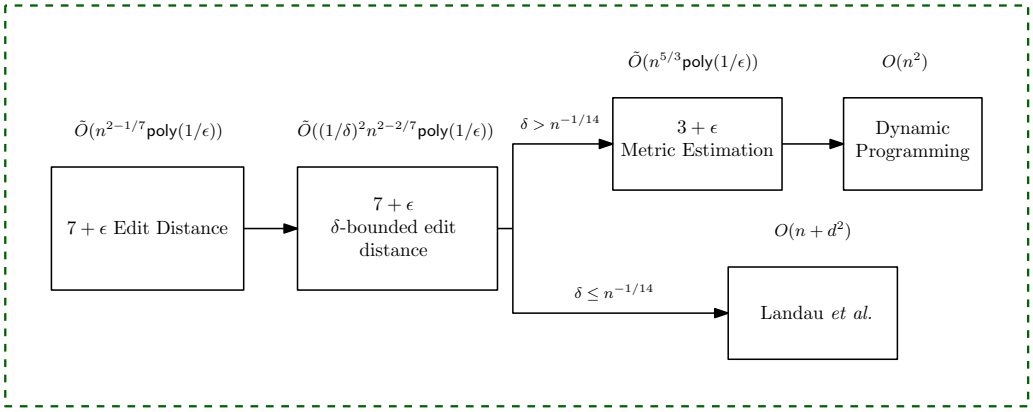


Figure 3: The diagram depicts the components of the $7+\epsilon$ algorithm for edit distance. $x \to y$ shows that component x uses component y as a black box.

In this section, we show how we combine these ideas to achieve an $\widetilde{O}(n^{2-(5-\sqrt{17})/4+\epsilon}\mathsf{poly}(1/\epsilon)) \simeq \widetilde{O}(n^{1.781})$ time algorithm. As to why the exponent converges to $2-(5-\sqrt{17})/4$, we refer the reader to a discussion in the full version of this paper.

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 $\widetilde{O}(n^{2-(5-\sqrt{17})/4+\epsilon}\mathsf{poly}(1/\epsilon))$. We call our algorithm for this problem $\mathcal{A}(\epsilon)$, and refer to its time complexity and approximation factor with $\mathsf{t_e}(\epsilon)$ and $\mathsf{e_e}(\epsilon)$, respectively. We inductively show that

$$\mathsf{t_e}(\epsilon) = \widetilde{O}(n^{2-(5-\sqrt{17})/4+\epsilon}\mathsf{poly}(1/\epsilon))$$

and $\mathsf{e_e}(\epsilon) = O(1/\epsilon)^{O(\log 1/\epsilon)}$. Notice that if $2-(5-\sqrt{17})/4+\epsilon \geq 2$, $\mathcal{A}(\epsilon)$ can be trivially implemented with the classic $O(n^2)$ algorithm and the approximation factor $\mathsf{e_e}(\epsilon)=1$. Now, assume that $2-(5-\sqrt{17})/4+\epsilon < 2$.

An $\widetilde{O}((1/\delta)^2 n^{2-(5-\sqrt{17})/2+2\epsilon}\mathsf{poly}(1/\epsilon))$ time algorithm for δ-bounded edit distance suffices to design $\mathcal{A}(\epsilon)$. If $\delta \leq n^{-(5-\sqrt{17})/8+\epsilon/2}$ we run the $O(n+\delta^2 n^2)$ of Landau *et al.* [26], otherwise the running time of our algorithm is $\widetilde{O}(n^{2-(5-\sqrt{17})/4+\epsilon}\mathsf{poly}(1/\epsilon))$. 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 δ-bounded edit distance problem in time $\widetilde{O}((1/\delta)^2 n^{2-(5-\sqrt{17})/2+2\epsilon}\mathsf{poly}(1/\epsilon))$. To this end, we again define two parameters β and γ and set the window size equal to $\lfloor n^{1-\beta} \rfloor$ and the gap size equal to $g = \lfloor l/\gamma \rfloor$. Similar to what 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 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 $\langle \mathcal{M}, \mathsf{edit} \rangle$, 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 $\mathsf{t_e}(2\epsilon) = \widetilde{O}(n^{2-(5-\sqrt{17})/4+2\epsilon}\mathsf{poly}(1/\epsilon))$ as on oracle function. Using the approximation algorithm of metric estimation, find an $\mathsf{e_m}(\epsilon)\mathsf{e_e}(2\epsilon)$ approximation solution to the edit distances for every pair of windows (Theorem 3.4).

3. Based on the estimated distances, find an $\mathsf{e_m}(\epsilon)\mathsf{e_e}(2\epsilon)$ approximately optimal window-compatible transformation (Lemma 4.1).

4. Report the transformation as an approximation proof for the δ-bounded edit distance problem.

Notice that there are two modifications to the previous algorithm. First, instead of using the $3+\epsilon$ factor algorithm for metric estimation, here, we use an $\mathsf{e_m}(\epsilon)$ approximation factor algorithm that runs in time $\widetilde{O}(n^{3/2+\epsilon}\mathsf{poly}(1/\epsilon))$. 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. By setting the right values for parameters β and γ, the running time and approximation factor of algorithm $\mathcal{A}(\epsilon)$ would be $\widetilde{O}(n^{2-(5-\sqrt{17})/4+\epsilon}\mathsf{poly}(1/\epsilon))$ and $\mathsf{e_e}(\epsilon) = O(1/\epsilon)^{O(\log 1/\epsilon)}$, respectively.

Theorem 4.3 [restated] There exists an $\widetilde{O}(n^{2-(5-\sqrt{17})/4+\epsilon})$ time quantum algorithm that approximates edit distance within a factor

$\mathsf{e}_{\mathsf{e}}(\epsilon) = O(1/\epsilon)^{O(\log 1/\epsilon)}$.

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

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 α 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 α. 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 $\widetilde{O}(n^{3/2+\epsilon}\mathsf{poly}(\epsilon))$ oracle call for a larger constant approximation $\mathsf{e}_{\mathsf{m}}(\epsilon) = O(1/\epsilon)$.

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 $\mathcal{O}$ 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: $\mathsf{par}(f) = \prod_{i\in[n]} f(i)$.

Note that, $\mathsf{par}(f)$ is either $+1$ or -1 for every function f. Farhi *et al.* [15] proved that at least $\Omega(n)$ oracle queries are necessary to find $\mathsf{par}(f)$. 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 [14, 30]. We are now ready to present our reduction.

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, 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 $\mathsf{par}(f)$ 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.*

3.2 A $3+\epsilon$ Approximation Algorithm with $\widetilde{O}(n^{5/3}\mathsf{poly}(1/\epsilon))$ 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 $\widetilde{O}(n^{5/3}\mathsf{poly}(1/\epsilon))$ 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 $[l, u]$. We separate the numbers into disjoint intervals. If $l = 0$, 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 $l \le x \le u$ which is not covered in the previous intervals and add a new interval $[x, (1+\epsilon)x]$ to the list. Since $u = \mathsf{poly}(n)$, the number of intervals is $\mathsf{poly}(\log n)\mathsf{poly}(1/\epsilon) = \widetilde{O}(\mathsf{poly}(1/\epsilon))$. 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.

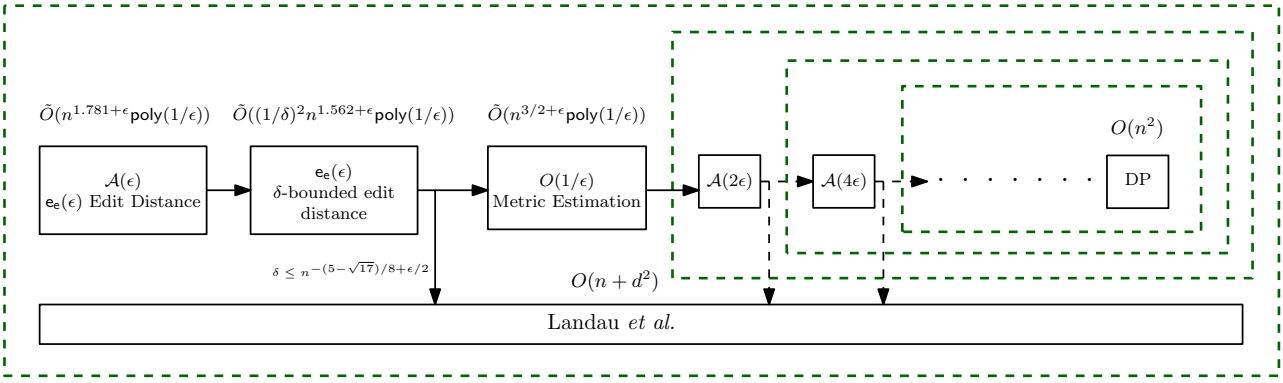


Figure 4: The diagram illustrates the bootstrapping technique to achieve an $\widetilde{O}(n^{1.781})$ time quantum algorithm for approximating edit distance. $x \to y$ shows that component x uses component y as a black box.

In order to approximate threshold estimation, we subsequently make use of Grover's search algorithm [12]. 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^τ and *high degree* otherwise. Our algorithm deals with low degree vertices and high degree vertices differently. We set the value of τ 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^τ 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^τ 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^τ. 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 $3t$ 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.

Algorithm 1: EstimateWithThreshold$(n, \mathcal{O}, t)$

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 $d(p_i, p_j) \le t$ we have $A_{i,j} = 1$, and for each $A_{i,j} = 1$ we have $d(p_i, p_j) \le 3t$.

```
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;
```

THEOREM 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 $O(n^2)$.

Now, we are ready to present our $3+\epsilon$ approximation algorithm with query complexity $\widetilde{O}(n^{5/3}\mathsf{poly}(1/\epsilon))$.

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$(n, \mathcal{O}, \epsilon, l, u)$

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

In this section, we achieved an algorithm with subquadratic query complexity and approximation factor $3+\epsilon$ for any $\epsilon > 0$ which 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 larger constants.

3.3 A Constant Approximation Algorithm with $\widetilde{O}(n^{3/2+\epsilon}\mathsf{poly}(1/\epsilon))$ 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 $\widetilde{O}(n^{3/2+\epsilon}\mathsf{poly}(1/\epsilon))$, 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 $\widetilde{O}(n^{3/2+\epsilon}\mathsf{poly}(1/\epsilon))$ where the approximation factor and the expected running time are $\mathsf{e_m}(\epsilon) = O(1/\epsilon)$ and $\widetilde{O}(n^2\mathsf{poly}(1/\epsilon))$, 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 η 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* which have at least one neighbor in *representatives*, or both. Next, we run the following procedure: for every vertex v_i which is not in *representatives*, we first check if it is a follower. For a follower vertex which 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 which 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 far as it doesn't

increase the query complexity too much. By increasing the degree threshold to its 3rd power, we have this property. The number of vertices in nested recursions depleted, as soon as the degree threshold become larger that the number of vertices, in which case we treat all vertices as low degree, thus the next time we have zero vertices and the process finishes.

The pseudocode of the algorithm is shown below.

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

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 ϵ, 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 $d(p_i, p_j)$.

1 **if** $n = 0$ **then**
2 Output an empty matrix;
3 **else**
4 Sample a hitting set $\mathcal{R}$ with $O((n/{n_0}^\tau)\log n)$ points;
5 Initialize an $n \times n$ matrix A;
6 **for** *all points in* $\mathcal{M}$ *as* v_i **do**
7 Find a neighbor of v_i or v_i itself in $\mathcal{R}$ and save it as $l(v_i)$ (the leader of v_i);
8 **if** *no such neighbor of* v_i *exists and* v_i *is not in* $\mathcal{R}$ **then**
9 List all neighbors of v_i;
10 $A' \leftarrow$ FastEstimateWithThreshold$(\mathcal{R}, \mathcal{O}, 3t, 3\epsilon, ({n_0}^\tau)^3)$;
11 **for** *all pairs of points in* $\mathcal{M}$ *as* (v_i, v_j) *where* $l(v_i) \neq \varnothing$ *and* $l(v_j) \neq \varnothing$ **do**
12 **if** $A'(l(v_i), l(v_j)) = 1$ **then**
13 $A(v_i, v_j) \leftarrow 1$;
14 $A \leftarrow A \vee A'$;
15 Output A;

THEOREM 3.4. *Algorithm 3 called with the threshold t, the parameter ϵ 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 $\widetilde{O}(n^2)$.*

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 $\mathsf{e_m}(\epsilon)$.

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

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 a $\mathsf{e_m}(\epsilon)$ approximation of $d(p_i, p_j)$ in $\langle \mathcal{M}, d \rangle$

1 Initialize the distance estimation matrix A, A° and $A^\bullet$;
2 $A^\circ \leftarrow$FastEstimateWithThreshold$(n, \mathcal{O}, 0, \epsilon, n^{2\epsilon})$;
3 Initialize the threshold: $t \leftarrow \max(1, l)$;
4 **while** $t \leq u$ **do**
5 $t \leftarrow t \cdot (1 + \epsilon)$;
6 $A^\bullet \leftarrow$FastEstimateWithThreshold$(n, \mathcal{O}, t, \epsilon, n^{2\epsilon})$;
7 $A \leftarrow A + (A^\bullet - A^\circ) \cdot \mathsf{e_m}(\epsilon)$;
8 $A^\circ \leftarrow A^\circ \vee A^\bullet$;
9 Output A;

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

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 Ambainiss 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 π 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 out lower bound.

LEMMA 3.1. *Any quantum algorithm which 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* $\Omega(n^{3/2})$.

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 permutations and vice versa. Therefore, we have the following theorem.

THEOREM 3.7. *Any quantum algorithm which estimates distances of a metric space of* n *points with a constant approximation factor has a query complexity of at least* $\Omega(n^{3/2})$.

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 $7+\epsilon$ for an arbitrarily small number $\epsilon > 0$ and time complexity $\widetilde{O}(n^{2-1/7}\mathsf{poly}(1/\epsilon))$. The outline of the algorithm is presented in Section 2. Here we provide detailed proofs of the lemmas and theorems that are previously used for edit distance.

LEMMA 4.1. *Given a matrix of edit distances between the substrings corresponding to every pair of windows of* W_1 *and* W_2, *one can compute the optimal window-compatible transformation of* s_1 *into* s_2 *in time* $O(n + |W_1||W_2|)$.

COROLLARY 4.1. *Given an* α*-approximation matrix of edit distances between the substrings corresponding to every pair of windows of* W_1 *and* W_2, *one can compute an* α*-approximation of the optimal window-compatible transformation of* s_1 *into* s_2 *in time* $O(n + |W_1||W_2|)$.

LEMMA 4.2. *Given that* $\mathsf{edit}(s_1, s_2) \leq \delta n$, *there exists a window-compatible transformation of* s_1 *into* s_2 *with respect to* W_1 *and* W_2 *that has at most* $(3\delta + 1/\gamma)n + 2l$ *operations.*

The next lemma proves the approximation factor and time complexity of our $7+\epsilon$ approximation algorithm for the δ-bounded edit distance problem.

LEMMA 4.3. *There exists a quantum algorithm that solves the* δ*-bounded edit distance problem within an approximation factor of* $7+\epsilon$ *in time* $\widetilde{O}(n^{2-1/7}\mathsf{poly}(1/\epsilon))$.

THEOREM 4.1. *There exists a quantum algorithm that solves edit distance within an approximation factor of* $7+\epsilon$ *in time* $\widetilde{O}(n^{2-1/7}\mathsf{poly}(1/\epsilon))$.

5 Approximating Edit Distance in MapReduce

Edit distance has been studied in parallel and distributed models since the 90s. 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 MapReduce model. More precisely, we give a $(3+\epsilon)$-approximation algorithm which 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 [6] 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 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 [23].

We begin by stating some of the MapReduce notions and definitions in Section 5.1 and next explain our algorithm is Section 5.2.

5.1 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 no exceed the total memory of all machines together. In addition to this, the total outputs of a reducer should not be more that its memory. Once *all* reducers finished 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, Karloff, Suri, and Vassilvitskii [24] 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.* [27] 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 5.2 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.

5.2 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 δ-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 δ-bounded edit distance problem.

We use two different approaches for large δ's and small δ's. For large δ'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 δ's though, we use a new method based on $(\mathsf{min}, +)$ matrix multiplication, also known as distance multiplication. We denote it by $\star$. We separate the large and the small δ's with a critical value based on the number of machines[9].

For $(\mathsf{min}, +)$ matrix multiplication in the MapReduce model, we use a parameterized version of the algorithm presented in [19].

THEOREM 5.1. (PROVED IN [19]) *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 5.1 to halve the number of matrices in two rounds; therefore we have the following corollary.

COROLLARY 5.1. (OF THEOREM 5.1) *The $(\mathsf{min}, +)$ multiplication of n^a matrices of size $n^b \times n^b$ can be*

[9] for $n^{8/9}$ machines $\delta^* = n^{-8/27}$.

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 5.1, 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 5.1 where $x = 2 - 2y/3$. Also note that for $0 \leq y \leq a + 3b/2$, we use Theorem 5.1 with $1 \leq x \leq 2$, hence all $1/x$ terms are ignored.

In Sections 5.2.1 and 5.2.2, we discuss our approach for large δ's and small δ's, respectively. In Section 5.2.3, we discuss the remaining details of the algorithm.

5.2.1 Our Approach for Large δ's The overall idea of our solution for large δ'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 some pairs of windows. Previously, we found an approximated edit distance between all pairs of windows using metric estimation. On the contrary, here we can do better than finding the edit distance between all pairs based on the following observation.

LEMMA 5.1. *Given that $\mathsf{edit}(s_1, s_2) \leq \delta n$, there exists a window-compatible transformation of s_1 into s_2 with respect to W_1 and W_2, that 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 $\lceil \delta n/g \rceil$, and the number of operations is at most $(3\delta + 1/\gamma)n + 2l$.*

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; the difference is that the memory and the running time is slightly reduced by Lemma 5.1. The second round uses only one machine.

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

LEMMA 5.2. *We can solve the δ-bounded edit distance problem for*

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

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

5.2.2 Our Approach for Small δ's 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.* [26] with time $O(n+d^2)$. This solution cannot (trivially) become parallel. Here, we instead use a novel approach based on $(\mathsf{min}, +)$ 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 $\mathsf{edit}(s_1, s_2)$ (Corollary 1 of [35]).

Let $d(i, j+1, i', j'+1)$ be the edit distance between two substrings of $s_1[i, \ldots, j]$ and $s_2[i', \ldots, j']$. We have the following lemma.

LEMMA 5.3. *For an arbitrary k, $i < k \leq j$, we have:*

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

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 [35]), 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 5.3 in matrices, we have the following corollary.

COROLLARY 5.2. (OF LEMMA 5.3) *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 $(\mathsf{min}, +)$ matrix multiplication operator.*

Notice that $\mathsf{edit}(s_1, s_2) = d(1, |s_1|+1, 1, |s_2|+1)$, 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 5.2 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 5.1.

LEMMA 5.4. *We can solve the δ-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 at most $O(\log n)$ *MapReduce rounds.*

5.2.3 Conclusion We compute edit distance by solving the δ-bounded edit distance problems for several δ's in parallel. For each $\delta = n^{-\alpha}$ we use the appropriate MapReduce algorithm based on the value of x and α. 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 $(3+\epsilon)$-approximation algorithm for edit distance is as follows.

THEOREM 5.2. *We can solve the edit distance problems in 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* $O((1/\epsilon^2)n^{(11-5x)/8+\epsilon'})$ *for one machine in time* $O(n^{(51-29x)/16})$ *(for one machine), and for*
- $13/20 \leq x \leq 7/6$ *with a memory of at most* $O((1/\epsilon^2)n^{2(4-x)/7+\epsilon'})$ *in time* $O(n^{(58-25x)/21})$ *(for one machine).*

By setting $x = 8/9$, we minimize the maximum of the number of machines and the memory of each machine. This is shown in Figure 5.

COROLLARY 5.3. *We can solve the edit distance problems in MapReduce model with an approximation factor of* $3+\epsilon$ *in* $O(\log n)$ *rounds with* $\widetilde{O}((1/\epsilon)n^{8/9})$ *machines, a memory of* $O((1/\epsilon^2)n^{8/9+\epsilon'})$ *for each machine, and in time* $O(n^{2-8/27})$ *(for one machine), where* $\epsilon' > 0$ *is an arbitrary constant.*

6 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 $O(n^2)$ solution for these problems are very analogous, unfortunately, 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, the usability of Lemma 4.2, which states that there is a window-compatible solution which 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 any 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 $\mathsf{dtw}(s1,s3) = 2k-1$. 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 the Lemma 4.2. 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 windows-compatible transformation is at most t.

Likewise, approximating lcs in classic computers is also harder that 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 $\mathsf{co\text{-}lcs}(s_1,s_2) = |s_1| + |s_2| - \mathsf{lcs}(s_1,s_2)$. 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 $7+\epsilon$ in quantum computers and an approximation factor of $3+\epsilon$ 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

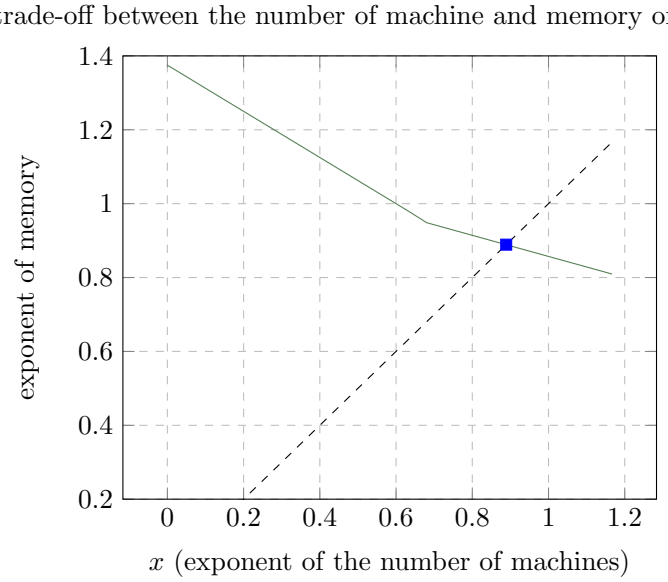


Figure 5: The trade-off between the number of machines and memory of each machine is shown. In $x = 8/9$ the maximum of the number of machines and the memory of each machine is minimized.

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 6.1. *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 which computes the exact Fréchet distance in time $O(n^{2-\epsilon})$.*

Theorem 6.1 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.

7 Open Problems

Indeed the most important open problem concerning edit distance is whether a subquadtaric time algorithm can approximate the edit distance of two strings within a constant factor? In addition to this, our work gives raise to a number of questions that we believe are important to study in future work.

- *Is there a quantum algorithm that approximates edit distance within a factor better than 7 in truly subquadratic time?*
- *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?*

8 Acknowledgment

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

References

[1] A. Ambainis. Quantum lower bounds by quantum arguments. In *STOC*, pages 636–643. ACM, 2000.

[2] A. Andoni and R. Krauthgamer. The smoothed complexity of edit distance. In *ICALP*, pages 357–369. Springer, 2008.

[3] A. Andoni, R. Krauthgamer, and K. Onak. Polylogarithmic approximation for edit distance and the asymmetric query complexity. In *FOCS*, pages 377–386. IEEE, 2010.

[4] A. Andoni, A. Nikolov, K. Onak, and G. Yaroslavtsev. Parallel algorithms for geometric graph problems. In *STOC*, pages 574–583. ACM, 2014.

[5] A. Andoni and K. Onak. Approximating edit distance in near-linear time. In *STOC*, pages 199–204. ACM, 2009.

[6] A. Apostolico, M. J. Atallah, L. L. Larmore, and S. McFaddin. Efficient parallel algorithms for string editing and related problems. *SIAM Journal on Computing*, 19(5):968–988, 1990.

[7] A. Backurs and P. Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). In *STOC*, pages 51–58. ACM, 2015.

[8] Z. Bar-Yossef, T. Jayram, R. Krauthgamer, and R. Kumar. Approximating edit distance efficiently. In *FOCS*, pages 550–559. IEEE, 2004.

[9] T. Batu, F. Ergun, and C. Sahinalp. Oblivious string embeddings and edit distance approximations. In *SODA*, pages 792–801. SIAM, 2006.

[10] R. Beals. Quantum computation of fourier transforms over symmetric groups. In *STOC*, pages 48–53. ACM, 1997.

[11] A. Belovs. Learning-graph-based quantum algorithm for k-distinctness. In *FOCS*, pages 207–216. IEEE, 2012.

[12] M. Boyer, G. Brassard, P. Høyer, and A. Tapp. Tight bounds on quantum searching. *Fortschritte der Physik*, 46(4-5):493–505, 1998.

[13] G. Brassard, P. Høyer, M. Mosca, and A. Tapp. Quantum amplitude amplification and estimation. *Contemporary Mathematics*, 305:53–74, 2002.

[14] C. Dürr, M. Heiligman, P. Høyer, and M. Mhalla. Quantum query complexity of some graph problems. *SIAM Journal on Computing*, 35(6):1310–1328, 2006.

[15] E. Farhi, J. Goldstone, S. Gutmann, and M. Sipser. A Limit on the speed of quantum computation in determining parity. *Physical Review Letters*, 81:5442–5444, 1998.

[16] E. Farhi, J. Goldstone, S. Gutmann, and M. Sipser. Invariant quantum algorithms for insertion into an ordered list. *arXiv preprint quant-ph/9901059*, 1999.

[17] F. L. Gall. Improved quantum algorithm for triangle finding via combinatorial arguments. In *FOCS*, pages 216–225, 2014.

[18] L. K. Grover. A fast quantum mechanical algorithm for database search. In *STOC*, pages 212–219. ACM, 1996.

[19] M. HajiAghayi, S. Lattanzi, S. Seddighin, C. Stein, and S. Vassilvitskii. MapReduce meets fine-grained complexity: MapReduce algorithms for APSP, matrix multiplication, 3-SUM, and beyond. Manuscript submitted for publication.

[20] S. Im, B. Moseley, and X. Sun. Efficient massively parallel methods for dynamic programming. In *STOC*, pages 798–811. ACM, 2017.

[21] P. Indyk. Algorithmic applications of low-distortion geometric embeddings. In *FOCS*, pages 10–33. IEEE, 2001.

[22] S. Jeffery, R. Kothari, and F. Magniez. Nested quantum walks with quantum data structures. In *SODA*, pages 1474–1485. SIAM, 2013.

[23] S. Jhaver, L. Khan, and B. Thuraisingham. Calculating edit distance for large sets of string pairs using MapReduce. Paper presented at ASE International Conference on Big Data, Beijing, China, August 2014.

[24] H. Karloff, S. Suri, and S. Vassilvitskii. A model of computation for MapReduce. In *SODA*, pages 938–948. SIAM, 2010.

[25] H. Krovi and A. Russell. Quantum fourier transforms and the complexity of link invariants for quantum doubles of finite groups. *Communications in Mathematical Physics*, 334(2):743–777, 2015.

[26] G. M. Landau, E. W. Myers, and J. P. Schmidt. Incremental string comparison. *SIAM Journal on Computing*, 27(2):557–582, 1998.

[27] S. Lattanzi, B. Moseley, S. Suri, and S. Vassilvitskii. Filtering: a method for solving graph problems in MapReduce. In *SPAA*, pages 85–94. ACM, 2011.

[28] F. Le Gall. Improved quantum algorithm for triangle finding via combinatorial arguments. In *FOCS*, pages 216–225. IEEE, 2014.

[29] W. J. Masek and M. S. Paterson. A faster algorithm computing string edit distances. *Journal of Computer and System Sciences*, 20(1):18–31, 1980.

[30] A. Montanaro, R. Jozsa, and G. Mitchison. On exact quantum query complexity. *Algorithmica*, 71(4):775–796, 2015.

[31] A. Nayebi and V. V. Williams. Quantum algorithms for shortest paths problems in structured instances. *arXiv preprint arXiv:1410.6220*, 2014.

[32] R. Ostrovsky and Y. Rabani. Low distortion embeddings for edit distance. In *STOC*, pages 218–224, New York, NY, USA, 2005. ACM.

[33] H. Ramesh and V. Vinay. String matching in $\widetilde{O}(\sqrt{n}+\sqrt{m})$ quantum time. *Journal of Discrete Algorithms*, 1(1):103–110, 2003.

[34] P. W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *FOCS*, pages 124–134. IEEE, 1994.

[35] E. Ukkonen. Algorithms for approximate string matching. *Information and Control*, 64(1-3):100–118, 1985.