On the Quantum Complexity of Closest Pair and Related Problems

Scott Aaronson
The University of Texas at Austin, TX, USA
aaronson@cs.utexas.edu

Nai-Hui Chia
The University of Texas at Austin, TX, USA
nai@cs.utexas.edu

Han-Hsuan Lin
The University of Texas at Austin, TX, USA
linhh@cs.utexas.edu

Chunhao Wang
The University of Texas at Austin, TX, USA
chunhao@cs.utexas.edu

Ruizhe Zhang
The University of Texas at Austin, TX, USA
rzzhang@cs.utexas.edu

Abstract

The closest pair problem is a fundamental problem of computational geometry: given a set of n points in a d-dimensional space, find a pair with the smallest distance. A classical algorithm taught in introductory courses solves this problem in $O(n \log n)$ time in constant dimensions (i.e., when $d = O(1)$). This paper asks and answers the question of the problem's quantum time complexity. Specifically, we give an $\widetilde{O}(n^{2/3})$ algorithm in constant dimensions, which is optimal up to a polylogarithmic factor by the lower bound on the quantum query complexity of element distinctness. The key to our algorithm is an efficient history-independent data structure that supports quantum interference.

In polylog(n) dimensions, no known quantum algorithms perform better than brute force search, with a quadratic speedup provided by Grover's algorithm. To give evidence that the quadratic speedup is nearly optimal, we initiate the study of quantum fine-grained complexity and introduce the *Quantum Strong Exponential Time Hypothesis (QSETH)*, which is based on the assumption that Grover's algorithm is optimal for CNF-SAT when the clause width is large. We show that the naïve Grover approach to closest pair in higher dimensions is optimal up to an $n^{o(1)}$ factor unless QSETH is false. We also study the bichromatic closest pair problem and the orthogonal vectors problem, with broadly similar results.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness; Theory of computation → Design and analysis of algorithms; Theory of computation → Quantum complexity theory

Keywords and phrases Closest pair, Quantum computing, Quantum fine grained reduction, Quantum strong exponential time hypothesis, Fine grained complexity

Digital Object Identifier 10.4230/LIPIcs.CCC.2020.16

Funding SA was supported by a Vannevar Bush Fellowship from the US Department of Defense, a Simons Investigator Award, and the Simons "It from Qubit" collaboration. NHC, HHL, and CW were supported by SA's Vannevar Bush Faculty Fellowship. RZ received support from the National Science Foundation (grant CCF-1648712).

Acknowledgements We would like to thank Lijie Chen and Pasin Manurangsi for helpful discussion. We would like to thank anonymous reviewers for their valuable suggestions on this paper.

35th Computational Complexity Conference (CCC 2020).
Editor: Shubhangi Saraf; Article No. 16; pp. 16:1–16:43

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

In the closest pair problem (CP), we are given a list of points in $\mathbb{R}^d$, and asked to find two that are closest. (See Figure 1 for an illustration of this problem.) This is a fundamental problem in computational geometry and has been extensively studied. Indeed, CP is one of the standard examples in textbooks (such as [20] and [32]) to introduce the divide-and-conquer technique. Moreover, CP relates to problems that have critical applications in spatial data analysis and machine learning, such as empirical risk minimization [7], point location [44, 13], time series motif mining [35], spatial matching problems [51], and clustering [36]. Therefore, any improvement on CP may imply new efficient algorithms for related applications.

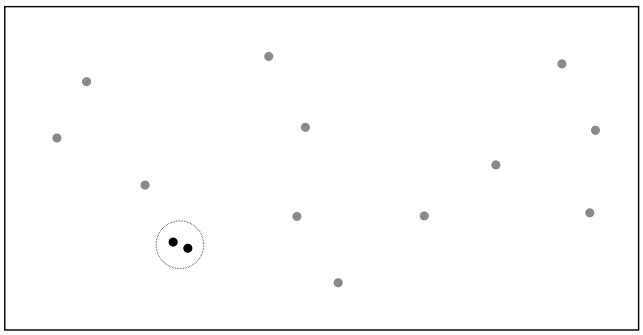

Figure 1 An instance of the CP, where the the closest pair is labeled in the circle.

Like with many other geometric problems, the hardness of CP rises as the dimension d increases. Shamos and Hoey gave the first $O(n \log n)$ deterministic algorithm in $\mathbb{R}^2$ by using Voronoi diagrams [44], improving on the trivial $O(n^2 d)$ upper bound. Then, Bentley and Shamos gave an algorithm with $2^{O(d)} n \log n$ running time via a divide-and-conquer approach [10]. A randomized algorithm by Khuller and Matias [30, 40] takes $2^{O(d)} n$ expected running time. A trivial lower bound for CP is $\Omega(n)$, since one must read all points to find the closest pair in the worst case. Yao showed an $\Omega(n \log n)$ lower bound for CP on the algebraic decision tree model [52].

When we consider CP in polylog(n) dimensions, the running time of all existing algorithms blows up to $\Omega(n^2)$, and thus it is unknown if there exists an algorithm matching the unconditional lower bounds. Nevertheless, under the *Strong Exponential Time Hypothesis (SETH)*, Karthik and Manurangsi [29], and David et al. [22], recently proved a conditional lower bound of $n^{2-o(1)}$ for CP in polylog(n) dimensions. This implies that the brute force approach is nearly optimal in polylog(n) dimensions unless SETH is false. SETH was introduced by Impagliazzo and Paturi [26], and is the assumption that for all $\epsilon > 0$, there exists an integer $k > 2$ such that no algorithm can solve k-SAT in time $O(2^{(1-\epsilon)n})$.

The main idea behind the results of [29, 22] is to prove a "fine-grained" reduction from CNF-SAT to CP in polylog(n) dimensions. Fine-grained reductions are reductions between computational problems that keep track of the exact polynomial exponents. For instance, [29] showed that CNF-SAT with $2^{n(1-o(1))}$ time is reducible to CP in polylog n dimensions with $n^{2-o(1)}$ time, and thus the lower bound for CP in polylog n dimensions is $n^{2-o(1)}$ unless SETH is false.

Surprisingly, to our knowledge, the quantum time complexity of CP was hardly investigated before. The trivial quantum algorithm for CP is to use Grover's search algorithm on all n^2 pairs, which takes $O(nd)$ time. Sadakane et al. [41] sketched a quantum algorithm that runs in $O(n^{1-1/(4\lceil d/2 \rceil)})$ time. Volpato and Moura [47] claimed a quantum algorithm that

uses $O(n^{2/3})$ *queries*, but no analysis was given of the *running time*, and as we will see, the conversion from the query-efficient algorithm to a time-efficient algorithm is nontrivial. As for the lower bound, any quantum algorithm for CP needs $\Omega(n^{2/3})$ time, since Aaronson and Shi [1] proved such a lower bound for element distinctness, and CP contains element distinctness as a special case, where a closest pair has distance 0.

In this work, we resolve the quantum time complexity of CP. In constant dimensions, we observe that by using a quantum walk for element distinctness [5, 33], we can achieve $O(n^{2/3})$ queries for CP. However, to obtain the same time complexity, the algorithm needs some geometric data structure that supports fast updates and checking, and that – crucially – is "history-independent", i.e., the data structure is uniquely represented, disregarding the order of insertion and deletion. History-independence is essential since different representations of the same data would destroy quantum interference between basis states.

We propose a geometric data structure that is history-independent and that supports fast checking and updates. Our data structure works by discretizing $\mathbb{R}^d$ into hypercubes with length $\epsilon/\sqrt{d}$. Then, we use a hash table, skip lists, and a radix tree to maintain the locations of the points and hypercubes. This data structure is history-independent, and we can easily find pairs with distance at most ϵ with it. We then find the closest pair by a binary search. By using our data structure and a quantum walk [5, 33], we achieve quantum time complexity $\widetilde{O}(n^{2/3})$.

For CP in polylog(n) dimensions, one may expect a conditional lower bound under SETH. However, SETH fails when quantum algorithms are considered since a simple application of Grover's search algorithm on all assignments solves CNF-SAT in time $\widetilde{O}(2^{n/2})$. Furthermore, existing fine-grained reductions may require time greater than $O(2^{n/2})$.

In this paper, we introduce the *Quantum Strong Exponential Time Hypothesis (QSETH)* and *quantum fine-grained reductions*. We define QSETH as follows.

▶ **Definition 1** (QSETH). *For all $\epsilon > 0$, there exists some $k \in \mathbb{N}$ such that there is no quantum algorithm solving k-SAT in time $O(2^{(1-\epsilon)\frac{n}{2}})$.*

We then observe that the classical definition of fine-grained reductions cannot capture the features of quantum reductions such as superposed queries and speedups from quantum algorithms. For instance, a fine-grained reduction may reduce problem A to solving many instances of problem B and then output the best solution; in this case, one can use Grover's search algorithm to achieve a quadratic speedup. Therefore, instead of summing the running time over all instances as in Definition 16, we use a quantum algorithm which solves all instances in superposition and outputs the answer. We give a formal definition of quantum fine-grained reductions in Definition 25 and show that under QSETH, any quantum algorithm for CP in polylog(n) dimensions requires $n^{1-o(1)}$ time. This implies that Grover's algorithm is optimal for the problem up to an $n^{o(1)}$ factor.

Intuitively, QSETH is the conjecture that applying Grover's search algorithm over all assignments in superposition is the optimal quantum algorithm for CNF-SAT. This is similar to SETH, which says that a brute force search is optimal for CNF-SAT. A series of works on CNF-SAT [43, 39, 38, 25, 42] shows that for some constant $c \in [1, 2]$, there exist (randomized) algorithms for n-variable k-SAT that run in time $2^{n(1-c/k)}$. As k grows, the running time of these algorithms approach 2^n. When k is small, however, there are algorithms with better running times. For instance, when $k = 3$, Schöning [43] obtained an algorithm with $O(1.334^n)$ running time, which was later improved to $O(1.308^n)$ by Paturi et al. [39]. However, none of the above mentioned algorithms have good running time on larger k's, so SETH remains a plausible conjecture.

When k is small enough, there are also quantum algorithms for k-SAT [4, 21] running in time much less than $O(2^{n/2})$. However, these quantum algorithms mainly use Grover search to speed up the classical algorithms of [43, 39], and thus do not perform well for large k, either. Therefore, we conjecture that for large enough k, no quantum algorithm can do much better than Grover search.

Finally, we study the bichromatic closest pair problem (BCP) and the orthogonal vector problem (OV). Briefly, OV is to find a pair of vectors that are orthogonal given a set of vectors in $\{0,1\}^d \in \mathbb{R}^d$, and BCP is, given two sets A, B (representing two colors) of n points in $\mathbb{R}^d$, to find the pair (a,b) of minimum distance with $a \in A$ and $b \in B$.

We can summarize all of our results as follows.

▶ **Theorem 2** (Informal). *Assuming QSETH, there is no quantum algorithm running in time* $n^{1-o(1)}$ *for* OV, CP, *and* BCP *when* $d = \mathrm{polylog}(n)$.

▶ **Theorem 3** (Informal). *The quantum time complexity of* CP *in* $O(1)$ *dimensions*[I] *is* $\widetilde{\Theta}(n^{2/3})$[II].

▶ **Theorem 4** (Informal). *For any* $\delta > 0$, *there exists a quantum algorithm for* BCP *with* $\widetilde{O}(n^{1-\frac{1}{2d}+\delta})$ *running time. There exists a quantum algorithm which solves* $(1+\xi)$*-approximate* BCP *in time* $\widetilde{O}(\xi^{-d}n^{2/3})$.

▶ **Theorem 5** (Informal). *The quantum time complexity of* OV *in* $O(1)$ *dimensions*[III] *is* $\Theta(n^{1/2})$.

Table 1 also summarizes what is known about upper and lower bounds on the classical and quantum time complexities of all of these problems.

Table 1 A summary of our quantum complexity results and comparison to classical results. The bold entries highlight our contributions in this paper.

	Dimension		Lower Bound	Upper Bound
CP	$\Theta(1)$	Classical	$\widetilde{\Omega}(n)$ [52]	$\widetilde{O}(n)$ [44, 10, 30]
		Quantum	$\mathbf{\Omega(n^{2/3})}$ Theorem 56	$\mathbf{\widetilde{O}(n^{2/3})}$ Corollary 55
	$\mathrm{polylog}\, n$	Classical	$n^{2-o(1)}$ (Under SETH) [29]	$O(n^2)$
		Quantum	$\mathbf{n^{1-o(1)}}$ **(Under QSETH)** Theorem 26	$\mathbf{\widetilde{O}(n)}$ Theorem 15
OV	$\Theta(1)$	Classical	$\Omega(n)$	$O(n)$ [48]
		Quantum	$\mathbf{\Omega(n^{1/2})}$ Theorem 68	$\mathbf{O(n^{1/2})}$ Theorem 68
	$\mathrm{polylog}\, n$	Classical	$n^{2-o(1)}$ (Under SETH) [49]	$n^{2-o(1)}$ [2, 16]
		Quantum	$\mathbf{n^{1-o(1)}}$ **(Under QSETH)** Theorem 26	$\mathbf{\widetilde{O}(n)}$ Theorem 15
BCP	$\Theta(1)$	Classical	$\Omega(n)$	$O\left(n^{2-\frac{2}{\lceil d/2\rceil+1}+\delta}\right)$ [3]
		Quantum	$\mathbf{\Omega(n^{2/3})}$ Theorem 67	$\mathbf{\widetilde{O}(n^{1-\frac{1}{2d}+\delta})}$ for BCP Theorem 66 $\mathbf{\widetilde{O}(\xi^{-d}n^{2/3})}$ for $(1+\xi)$-BCP Theorem 64
	$2^{O(\log^*(n))}$[IV]	Classical	$n^{2-o(1)}$ (Under SETH) [17]	$n^{2-o(1)}$ [2, 16]
		Quantum	$\mathbf{n^{1-o(1)}}$ **(Under QSETH)** Theorem 35	$\mathbf{\widetilde{O}(n)}$ Theorem 15

[I] We actually give a slightly stronger result: the same time complexities still hold when $d = O\left(\frac{\log\log n}{\log\log\log n}\right)$.
[II] The $\widetilde{\Theta}$ notation is Θ with logarithmic factors hidden in both upper and lower bounds.
[III] The same time complexities still hold when $d = O(\log\log n)$.
[IV] $\log^*(n) := \log^*(\log n) + 1$ for $n > 1$ and $\log^*(1) := 0$. Hence, $2^{O(\log^* n)}$ is an extremely slow-growing function.

Related work

A recent independent work by Buhrman, Patro and Speelman [15] also studied quantum strong exponential time hypothesis. They defined (a variant of) QSETH based on the hardness of testing properties on the set of satisfying assignments of a SAT formula, e.g., the parity of the satisfying assignments. Based on these hardness assumptions extended from the original QSETH, they gave conditional quantum lower bounds for OV, the Proofs of Useful Work [8] and the edit distance problem. In comparison, we formally define the *quantum fine-grained reductions* and prove lower bounds for CP, OV, and BCP under the original form of QSETH by showing the existence of quantum fine-grained reductions from CNF-SAT to the these problems.

1.1 Proof overview

For ease of presentation, some notations and descriptions will be informal here. Formal definitions and proofs will be given in subsequent sections.

We give an optimal (up to a polylogarithmic factor) quantum algorithm that solves CP for constant dimensions in time $\widetilde{O}(n^{2/3})$. First note that there exists a Johnson graph corresponding to an instance of CP, where each vertex corresponds to a subset of $n^{2/3}$ points of the input of CP, and two vertices are connected when the intersection of the two subsets (they are corresponding to) has size $n^{2/3}-1$. A vertex is marked if the subset it corresponds to contains a pair with distance at most ϵ. Then, the goal is to find a marked vertex on this Johnson graph and use binary search over ϵ to find the closest pair. Our algorithm for finding a marked vertex is based on the quantum walk search framework by Magniez et al. [33], which can be viewed as the quantum version of the Markov chain search on a graph (in our case, a Johnson graph). The complexity of this quantum walk algorithm is $O(\mathsf{S}+\frac{1}{\sqrt{\lambda}}(\frac{1}{\sqrt{\delta}}\mathsf{U}+\mathsf{C}))$, where λ is the fraction of marked states in the Johnson graph, δ is its spectral gap, S is the cost for preparing the algorithm's initial state, U is the cost for implementing one step of the quantum walk, and C is the cost for checking the solution. For our Johnson graph, $\lambda = n^{-2/3}$ and $\delta = n^{-2/3}$. If we consider only the query complexity, $\mathsf{S} = n^{2/3}$, $\mathsf{U} = O(1)$, and $\mathsf{C} = 0$. However, the time complexity for C is huge in the straightforward implementation, e.g., storing all points in an array according to the index order, as we need to check all the pairs from the $n^{2/3}$ points, which will kill the quantum speedup. To tackle this, we discretize the space into small hypercubes. With this discretization, it suffices to check $O((\sqrt{d})^d)$ neighbor hypercubes to find a pair with distance at most ϵ. To support the efficient neighborhood search, we need an efficient data structure.

Existing data structures do not meet our need. They either have prohibitive dependence on the dimension, such as $\Omega(n^{\lceil d/2 \rceil})$ time for constructing and storing Voronoi diagrams [31], or do not have unique representation (i.e., they are history-dependent), such as fair-split trees and dynamic trees [13]. Note that the requirement of unique representation is due to the fact that different representations of the same data would destroy the interference that quantum computation relies on. To solve this problem, we propose a uniquely represented data structure that can answer queries about ϵ-close pairs and insert/delete points efficiently. This data structure is based on a hash table, skip lists, and a radix tree. With this data structure, $\mathsf{U} = O(\log n)$ and $\mathsf{C} = O(1)$. Hence, we have the desired time complexity (see Section 4.2). We give another method for solving CP that only uses a radix tree as the data structure. With only a radix tree, the algorithm cannot handle cases with multiple solutions, and we need to subsequently reduce the size of the problem until there is at most one solution (see Section 4.3). These two quantum algorithms have the same time complexity.

Our quantum algorithm for solving approximate BCP follows the same spirit as that for CP, except that we use a finer discritization of the space (see Section 5.1). To solve BCP exactly, we need a history-independent data structure for nearest-neighbor search, but no such data structure is known. Instead, we adapt the nearest-neighbor search data structure by Clarkson [19] to the quantum algorithm proposed by Buhrman et al. [14] for element distinctness, which does not require history-independence of the data structure because in the algorithm of [14], no insertions and deletions are performed once the data structure for a set of points is constructed (see Section 5.2). Sadakane et al. [41] sketched an algorithm for BCP with similar ideas and running time, but we give the first rigorous analysis.

To derive our quantum fine-grained complexity results for OV and CP when $d = \operatorname{polylog} n$ under QSETH, we first define quantum fine-grained reductions. In our definition, we consider problems whose input is given in the quantum query model, and allow the reduction to perform superposed queries and run quantum algorithms, e.g., amplitude amplification. The classical reductions from CNF-SAT to CP [29, 22] and OV [50] are not "quantum fine-grained" under QSETH. These reductions fail because their running time exceeds $2^{n/2(1-\epsilon)}$, which is the conjectured time complexity for CNF-SAT under QSETH. Therefore, we cannot derive from them any non-trivial lower bounds for CP or OV based on QSETH. In the following, we use the advantages of quantum algorithms to make these reductions work.

There are two main obstacles in "quantizing" the fine-grained reductions under QSETH. The first obstacle is that the time cost for preparing the input of the problem we reduce to is already beyond the required running time. For instance, consider the reduction from CNF-SAT to OV. Let φ be a CNF-SAT instance on n variables and m clauses. The classical fine-grained reduction divides all n variables into two sets A and B of size $n/2$, and then maps all assignments for variables in A and B to two sets V_A and V_B of $2^{n/2}$ vectors each. It is obvious that the time for writing down V_A and V_B is already $\Theta(2^{n/2})$. Nevertheless, many quantum algorithms achieve sublinear query complexities by querying the input oracle in superposition. Hence, instead of first constructing the input of OV at once and then running the algorithm, we can simulate it "on-the-fly": whenever the OV's algorithm queries the input oracle with some superposition of indices, we use a quantum subroutine to realize the input oracle by mapping the query indices to the corresponding assignments in CNF-SAT, and then to the corresponding vectors in V_A and V_B. This subroutine takes only $O(n)$ time, and therefore the quantum reduction, which has running time $O(n)$ times the running time of the OV algorithm, is quantum fine-grained.

Another difficulty in quantizing the fine-grained reductions is that some reduction needs to call the oracle multiple times, and the number of calls exceeds the required running time. However, it is possible to achieve quadratic speedup if these oracle calls are non-adaptive. For the reduction from BCP to CP, we can reduce a BCP instance to $n^{1.8+o(1)} \log n$ instances of CP, which is already larger than the conjectured $\Omega(n)$ quantum lower bound of BCP. By further studying the reduction, we find that the solution to BCP is the minimum of the solutions to the the constructed CP instances. Therefore, we can use the quantum minimum-finding algorithm to reduce the total time complexity to $\widetilde{O}(\sqrt{n^{1.8+\epsilon}} \cdot t_{\mathsf{CP}})$, which is enough to show that BCP is quantum fine-grained reducible to CP.

With the above-mentioned techniques, we quantize the classical fine-grained reductions, and show that CNF-SAT, with conjectured lower bound $\Omega(2^{n/2})$, is quantum fine-grained reducible to OV and CP with lower bound $\Omega(n')$[V], when the dimension d is $\operatorname{polylog}(n')$.

[V] n is the input size of CNF-SAT, and n' is the input size of OV and CP.

2 Preliminaries

▶ **Definition 6** (Distance measure). *For any two vectors $a, b \in \mathbb{R}^d$, the distance between them in the ℓ_2-metric is denoted by $\|a-b\| = \left(\sum_{i=1}^{d} |a_i - b_i|^2\right)^{1/2}$. Their distance in the ℓ_0-metric (Hamming distance) is denoted by $\|a-b\|_0 = |\{i \in [d] : a_i \neq b_i\}|$, i.e., the number of coordinates on which a and b differ.*

2.1 Quantum query model

We consider the quantum query model in this work. Let $X := \{x_1, \ldots, x_n\}$ be a set of n input points and $\mathcal{O}_X$ be the corresponding oracle. We can access the i-th data point x_i by making the query

$$|i\rangle |0\rangle \xrightarrow{\mathcal{O}_X} |i\rangle |x_i\rangle , \tag{1}$$

and we can make queries to elements in X in superposition. Note that $\mathcal{O}_X$ is an unitary transformation in the formula above. Hence, a quantum algorithm with access to $\mathcal{O}_X$ can be represented as a sequence of unitary transformations.

Consider a quantum algorithm $\mathcal{A}$ with access to an oracle $\mathcal{O}$ and a initial state $|0\rangle := |0\rangle_Q |0\rangle_A |0\rangle_W$, where the registers Q and A are for the queries and the answers from the oracle, and the register W is the working space which is always hold by $\mathcal{A}$. Then, we can represent the algorithm as

$$U_T \mathcal{O} U_{T-1} \cdots \mathcal{O} U_1 |0\rangle . \tag{2}$$

Let $|\psi\rangle_i = U_i \mathcal{O} \cdots \mathcal{O} U_1 |0\rangle := \sum_{i,z} |i\rangle_Q |0\rangle_A |z\rangle_W$ be the state right before applying the i-th $\mathcal{O}$, then

$$\mathcal{O} |\psi\rangle_i := \sum_{i,z} |i\rangle_Q |x_i\rangle_A |z\rangle_W . \tag{3}$$

2.2 Quantum subroutine for unstructured searching and minimum finding

▶ **Definition 7** (Unstructured search). *Given a set P of n elements in $\{0,1\}$, decide whether there exists a 1 in P.*

▶ **Theorem 8** (Grover's search algorithm [24, 37]). *There is a quantum algorithm for unstructured search with running time $O(\sqrt{n})$.*

By Theorem 8 and BBBV's argument [9], the quantum time complexity of unstructured search is $\Theta(\sqrt{n})$. We can also get a $\widetilde{O}(\sqrt{n})$ quantum algorithm for minimum finding by combining Grover's search algorithm and binary search.

▶ **Theorem 9** (Quantum minimum finding [23]). *There is a quantum algorithm that finds from a set of n elements with values in $\mathbb{R}$, the index of the minimum element of the set, with success probability $\frac{1}{2}$ and run time $\widetilde{O}(\sqrt{n})$*

2.3 Problem definitions

In this subsection, we first formally define OV, CP, and BCP. Then we show the folklore algorithms for CP, BCP, and OV by Grover's algorithm, which run in time $\widetilde{O}(n)$.

▶ **Definition 10** (Orthogonal Vectors, OV). *Given two sets A, B of n vectors in $\{0,1\}^d$ as input, find a pair of vectors $a \in A$, $b \in B$ such that $\langle a, b\rangle = 0$, where the inner product is taken in $\mathbb{Z}$.*[VI]

We denote OV with input length n and dimension d as $\mathsf{OV}_{n,d}$. We will use this notation when we need to specify the parameters in the following sections.

▶ **Definition 11** (Closest Pair Problem, CP). *Given a set P of n points in $\mathbb{R}^d$ and a distance measure Δ, find a pair of distinct points $a, b \in P$ such that $\Delta(a, b)$ is the smallest among all distinct pairs in P.*

Similar to OV, we denote CP with input length n and dimension d as $\mathsf{CP}_{n,d}$. We will use this notation when the parameters in the following sections are required to be specified. Note that in this work, we consider $\Delta(a, b) = \|a - b\|$ as the distance measure for CP and BCP.

▶ **Definition 12** (Bichromatic Closest Pair Problem, BCP). *Given two sets A, B of n points in $\mathbb{R}^d$ and a distance measure Δ, find a pair of points $a \in A$, $b \in B$ such that*

$$\Delta(a, b) = \min_{a \in A, b \in B} \Delta(a, b). \tag{4}$$

We also define an approximate version of BCP as follows.

▶ **Definition 13** ($(1+\xi)$-approximate Bichromatic Closest Pair Problem, $(1+\xi)$-BCP). *Given two sets A, B of n points $\in \mathbb{R}^d$ and a distance measure Δ, find a pair of points $a \in A$, $b \in B$ such that*

$$\Delta(a, b) \leq (1 + \xi) \min_{a \in A, b \in B} \Delta(a, b). \tag{5}$$

Same as CP, we use $\mathsf{BCP}_{n,d}$ and $(1+\xi)$-$\mathsf{BCP}_{n,d}$ to specify the parameters.

▶ **Definition 14** (Element Distinctness Problem, ED). *Let $f : [n] \to [m]$ be a given function. Decide whether there exist distinct $i, j \in [n]$ such that $f(i) = f(j)$.*

For this problem, Ambainis [5] gave a quantum algorithm with time complexity $\widetilde{O}(n^{2/3})$, which matches the lower bound proved by Aaronson and Shi [1] up to a polylogarithmic factor.

▶ **Theorem 15.** *There are $\widetilde{O}(n)$-time quantum algorithms for* CP *and* BCP *when $d = O(\mathrm{poly}\log n)$.*

Proof. We can solve CP and BCP by searching the minimum distance through all pairs by the algorithm of Theorem 9. There are $O(n^2)$ pairs and checking each pair took $O(d)$ time, so the total running time is $O(nd)$. For $d = O(\mathrm{poly}\log n)$, the time complexity equals to $\widetilde{O}(n)$. ◀

2.4 Fine-grained complexity

As we have mentioned earlier in the introduction, a fine-grained reduction from problem P to Q with conjectured lower bounds $p(n)$ and $q(n)$, respectively, has the property that if we can improve the $q(n)$ time for Q, then we can also improve the $p(n)$ time for P. We give the formal definition by Williams [46] in below.

[VI] Our definition is slightly different than some of the literature, for example, [18], which is searching among pairs inside one set. Those two definitions are equivalent up to constant in complexities.

▶ **Definition 16** (Fine-grained reduction, [46]). *Let $p(n)$ and $q(n)$ be non-decreasing functions of n. Problem P is (p,q)-reducible to problem Q, denoted as $(\mathsf{P},p) \leq_{\mathrm{FG}} (\mathsf{Q},q)$, if for every ϵ, there exist $\delta > 0$, an algorithm R for solving P with access to an oracle for Q, a constant d, and an integer $k(n)$, such that for every $n \geq 1$, the algorithm R takes any instance of P of size n and*

- *R runs in at most $d \cdot (p(n))^{1-\delta}$-time,*
- *R produces at most $k(n)$ instances of Q adaptively, that is, the jth instance X_j is a function of $\{(X_i, y_i)\}_{1 \leq i < j}$ where X_i is the ith instance produced and y_i is the answer of the oracle for Q on instance X_i, and*
- *the sizes n_i of the instances X_i for any choice of oracle answers y_i obeys the inequality*

$$\sum_{i=1}^{k(n)} (q(n_i))^{1-\epsilon} \leq d \cdot (p(n))^{1-\delta}. \tag{6}$$

Let $(\mathsf{P},p) \leq_{\mathrm{FG}} (\mathsf{Q},q)$ for some non-decreasing function $p(n)$ and $q(n)$. If for every $\epsilon > 0$, we can solve problem Q in time $q(n)^{1-\epsilon}$ with probability 1 for all input length n, then there exists a $\delta > 0$ such that we can solve the problem P in time $p(n)^{1-\delta}$ by Equation (6).

Here are some known results about fine-grained reductions.

▶ **Theorem 17** ([29, 49]).

$$(\mathsf{CNF\text{-}SAT}_n, 2^n) \leq_{\mathrm{FG}} (\mathsf{OV}_{n_1,d_1}, n_1^2) \leq_{\mathrm{FG}} (\mathsf{BCP}_{n_2,d_2}, n_2^2) \leq_{\mathrm{FG}} (\mathsf{CP}_{n_3,d_3}, n_3^2), \tag{7}$$

where $d_1 = \Theta(\log n_1)$, $d_2 = \Theta(\log n_2)$ and $d_3 = (\log n_3)^{\Omega(1)}$.

▶ Remark 18. The second reduction from OV to BCP has been improved to $d_2 = 2^{O(\log^* n)}$ by Chen [17].

There are several plausible hypotheses in fine-grained complexity, which can imply conditional hardness results for many interesting problems. We first give the definition of the strong exponential time hypothesis (SETH).

▶ **Hypothesis 19** (Strong Exponential Time Hypothesis, SETH). *For every $\epsilon > 0$, there exists a $k = k(\epsilon) \in \mathbb{N}$ such that no algorithm can solve k-SAT (i.e., satisfiability on a CNF of width k) in $O(2^{(1-\epsilon)m})$ time where m is the number of variables. Moreover, this holds even when the number of clauses is at most $c(\epsilon) \cdot m$ where $c(\epsilon)$ denotes a constant that depends only on ϵ.*

Another popular conjecture is the orthogonal vector hypothesis (OVH):

▶ **Definition 20** (Orthogonal Vector Hypothesis, OVH). *For every $\epsilon > 0$, there exists a $c \geq 1$ such that $\mathsf{OV}_{n,d}$ requires $n^{2-\epsilon}$ time when $d = c \log n$.*

▶ Remark 21. Under SETH, we can have the following conclusions from Theorem 17:

- OVH is true.
- For all $\epsilon > 0$, there exists a $c > 0$ such that $\mathsf{BCP}_{n,c\log n}$ cannot be solved by any randomized algorithm in time $O(n^{2-\epsilon})$.
- For all $\epsilon > 0$, there exists a $c > 0$ such that $\mathsf{CP}_{n,(\log n)^c}$ cannot be solved by any randomized algorithm in time $O(n^{2-\epsilon})$.

2.5 The framework for quantum walk search

In this subsection, we review the quantum walk framework for the Markov chain search problem and demonstrate how to use it to solve the element distinctness problem. For simplicity, we use the transition matrix P to refer to a Markov chain, where $P = (p_{xy})_{x,y\in X}$ for X being the state space of P and p_{xy} being the transition probability from x to y. An irreducible and ergodic Markov chain has a unique stationary distribution π, which is also the unique eigenvector of P with eigenvalue 1. Let $M \subseteq X$ be a set of marked elements. In the Markov chain search problem, the objective is to find an $x \in M$. We can perform the following actions: setup, sampling from the π with cost S; update, making a transition with cost U, and checking whether the current state is marked or not with cost C. To solve the search problem classically, we perform a random walk as follows. Start from a point sampled from π and check if it is marked. If not, make a number of transitions on P until it mixes, and then check again. We then repeat this process until a marked state is found. The cost of this random walk algorithm is $O(\mathsf{S} + \frac{1}{\lambda}(\frac{1}{\delta}\mathsf{U} + \mathsf{C}))$, where $\lambda := |M|/|X|$ and δ is the spectral gap of P.

Quantum analogues of random walks, namely, quantum walks, have been developed for solving different problems. In 2003, Ambainis [5] proposed a quantum walk algorithm for solving the element distinctness problem. His algorithm also solves the Markov chain search problem on the Johnson graph with cost $O(\mathsf{S} + \frac{1}{\sqrt{\lambda}}(\frac{1}{\sqrt{\delta}}\mathsf{U} + \mathsf{C}))$. In 2004, Szegedy [45] gave a quantum walk algorithm for more generalized Markov chains with cost $O(\mathsf{S} + \frac{1}{\sqrt{\lambda\delta}}(\mathsf{U} + \mathsf{C}))$. We can view Szegedy's quantum walk as a quantum counterpart of a random walk, where one checks the state after each transition. Szegedy's quantum walk only detects the presence of a marked state, but cannot find one without extra costs. In 2006, Magniez et al. [33] proposed a quantum walk search framework that unified the advantages of the quantum walks in [5] and [45]. In this quantum walk framework, we can perform the following operations:

- **Setup:** with cost S. preparing the initial state $|\pi\rangle = \frac{1}{\sqrt{|X|}}\sum_x \sqrt{\pi_x}\,|x\rangle$.
- **Update:** with cost U. applying the transformation $|x\rangle\,|0\rangle \mapsto |x\rangle \sum_{y\in X}\sqrt{p_{xy}}\,|y\rangle$.
- **Checking:** with cost C, applying the transformation: $|x\rangle \mapsto \begin{cases} -|x\rangle & \text{if } x \in M \\ |x\rangle & \text{otherwise.} \end{cases}$

The main result of [33] is summarized as follows.

▶ **Lemma 22** ([33]). *Let P be an irreducible and ergodic Markov chain P on X. Let $M \subseteq X$ be a subset of marked elements. Let $\lambda := |M|/|X|$ and δ be the spectral gap of P. Then, there exists a quantum algorithm that with high probability, determines M is empty or finds an $x \in M$ with cost $O(\mathsf{S} + \frac{1}{\sqrt{\lambda}}(\frac{1}{\sqrt{\delta}}\mathsf{U} + \mathsf{C}))$.*

To solve the element distinctness problem, we define a Markov chain, following the work [5, 11, 28]. The state space X is all subsets of $[n]$ with size r. The Markov chain is based on the Johnson graph on X, where an edge is connecting S and S' if and only if $|S \cap S'| = r - 1$. The transition probability on each edge is hence $\frac{1}{r(n-r)}$. A state S is marked when there exist distinct $i, j \in S$ such the i^{th} and the j^{th} items are the same. The Markov chain has spectral gap $\delta \geq 1/r$ (see [28]) and it is easy to verify that $\lambda \geq \binom{n-2}{r-2}/\binom{n}{r} = O(r^2/n^2)$. If we only consider the query complexity, the setup procedure costs r queries, the update procedure costs one query, and the checking procedure does not cost any query. Choosing $r = n^{2/3}$ yields the optimal query complexity $O(n^{2/3})$.

3 Quantum fine-grained complexity

In this section, we give the formal definitions of the quantum fine-grained reduction and quantum strong exponential time hypothesis (QSETH). Moreover, we show that under QSETH, for $d = \text{polylog}(n)$, the lower bounds for $\mathsf{CP}_{n,d}$ and $\mathsf{OV}_{n,d}$ are $n^{1-o(1)}$, which nearly matches the upper bounds given in Theorem 15.

3.1 Quantum fine-grained reduction and QSETH

QSETH is defined based on the assumption that the best quantum algorithm for CNF-SAT is Grover search when the clause width k is large enough.

▶ **Hypothesis 23** (QSETH). *For every $\epsilon > 0$, there exists a $k = k(\epsilon) \in \mathbb{N}$ such that no quantum algorithm can solve k-SAT (i.e., satisfiability on a CNF of width k) in $O(2^{(1/2-\epsilon)n})$ time where n is the number of variables. Moreover, this holds even when the number of clauses is at most $c(\epsilon)n$ where $c(\epsilon)$ denotes a constant that depends only on ϵ.*

Obviously, the Grover search can solve CNF-SAT in $\widetilde{O}(2^{n/2})$. To the best of the our knowledge, there is no quantum algorithm that can do better than $O(2^{n/2})$ for any k.

We recall that in the quantum query model, the input of a problem is given by a quantum oracle. Specifically, let P be a problem, and X be an instance of P in the classical setting. Then, in the quantum query model, X will be given by an oracle $\mathcal{O}_X$. We will denote an algorithm or an oracle $\mathcal{A}$ with access to $\mathcal{O}_X$ by $\mathcal{A}(\mathcal{O}_X)$.

We say $\mathcal{A}_\epsilon$ is an ϵ-oracle for problem P, if for every instance $\mathcal{O}_X$, it holds that

$$\Pr[\mathcal{A}_\epsilon(\mathcal{O}_X) = \mathsf{P}(X)] \geq 1 - \epsilon, \tag{8}$$

and the running time is $O(1)$, where $\mathsf{P}(X)$ is the answer of X for problem P.

▶ **Definition 24** (Quantum oracles). *Let $X := \{x_1, \ldots, x_n\}$ be an instance of some problem and $\mathcal{O}_X$ be the corresponding quantum oracle. To realize $\mathcal{O}_X$, we do not need to write down the whole X; instead, we can just design a quantum circuit to realize the mapping*

$$|i\rangle |0\rangle \xrightarrow{\mathcal{O}_X} |i\rangle |x_i\rangle . \tag{9}$$

▶ **Definition 25** (Quantum fine-grained reduction). *Let $p(n)$ and $q(n)$ be nondecreasing functions of n. Let P and Q be two problems in the quantum query model and $\mathcal{A}_\epsilon$ be an ϵ-oracle for Q with error probability $\epsilon \leq 1/3$. P is quantum (p, q)-reducible to Q, denoted as $(\mathsf{P}, p) \leq_{\text{QFG}} (\mathsf{Q}, q)$, if for every ϵ, there exits a $\delta > 0$, and algorithm R with access to $\mathcal{A}_\epsilon$, a constant d, and an integer $k(n)$, such that for every $n \geq 1$, the algorithm R takes any instance of P of size n and satisfies the following:*

- *R can solve P with success probability at least $2/3$ in time at most $d \cdot p(n)^{1-\delta}$.*
- *R performs at most $k(n)$ quantum queries to A_ϵ. Specifically, in the j^{th} query, let $\mathbf{X}_j := \{X_{1,j}, X_{2,j}, \ldots\}$ be a set instances of Q. Then, R realizes the oracles $\{\mathcal{O}_{X_{1,j}}, \mathcal{O}_{X_{2,j}}, \ldots\}$ in superposition and applies A_ϵ to solve the instances.*
- *The following inequality holds.*

$$\sum_{j=1}^{k(n)} c(\mathbf{X}_j) \cdot q(n_j)^{1-\epsilon} \leq d \cdot p(n)^{1-\delta},$$

where $c(\mathbf{X}_j)$ is the time required for R to realize the oracles $\{\mathcal{O}_{X_{1,j}}, \mathcal{O}_{X_{2,j}}, \ldots\}$ in superposition and $n_j := \max_i |X_{i,j}|$.

In Definition 25, the input of A_ϵ is given as a quantum oracle such that A_ϵ can be a quantum query algorithm with running time strictly less than the input size. Moreover, the quantum reduction R can realize quantum oracles $\{\mathcal{O}_{X_{1,j}}, \mathcal{O}_{X_{2,j}}, \dots\}$ in superposition, and thus the time required is $\max_i c(X_{i,j})$ (where $c(X_{i,j})$ is the time required to realize $\mathcal{O}_{X_{i,j}}$) instead of $\sum_i c(X_{i,j})$. This also allows R to use fast quantum algorithms to process the information of $A'_\epsilon s$ output (e.g., amplitude amplification).

3.2 Lower bounds for CP, OV, and BCP in higher dimensions under QSETH

Here, we give nearly linear lower bounds for OV and CP under QSETH by showing that there exist quantum fine-grained reductions from SAT to these problems.

▶ **Theorem 26.** *Assuming QSETH, for all $\epsilon > 0$, there exists a c such that $\mathsf{OV}_{n,c\log n}$ and $\mathsf{CP}_{n,(\log n)^c}$ cannot be solved by any quantum algorithm in time $O(n^{1-\epsilon})$.*

We prove Theorem 26 by showing that there exist quantum fine-grained reductions from CNF-SAT to OV, OV to BCP, and BCP to CP with desired parameters. We first give the reduction from CNF-SAT to OV as a warm-up.

▶ **Lemma 27.**

$$(\mathsf{CNF\text{-}SAT}_n, 2^{n/2}) \leq_{\mathrm{QFG}} (\mathsf{OV}_{n_1,d_1}, n_1), \tag{10}$$

where $n_1 = 2^{n/2}$ and $d_1 = \Theta(n)$.

Proof. Let ϕ be a CNF formula with n variables and $m = \Theta(n)$ clauses. Let $\mathcal{A}$ be an algorithm for OV. We first recall the classical reduction. Let $\phi := \phi_1 \wedge \cdots \wedge \phi_m$. We divide the n variables into two sets A and B with $|A| = |B| = \frac{n}{2}$. Let $A := \{x_1, \dots, x_{n/2}\}$ and $B := \{x_{n/2+1}, \dots, x_n\}$. We let $S_A := \{a_1, \dots, a_{2^{n/2}}\}$ be all assignments to A and $S_B := \{b_1, \dots, b_{2^{n/2}}\}$ be all assignments to B. We describe two mappings $f_A : S_A \to \{0,1\}^m$ and $f_B : S_B \to \{0,1\}^m$ as follows:

$$f_A(a_i) = [\phi_1(a_i), \dots, \phi_m(a_i)]^T, \text{ and} \tag{11}$$

$$f_B(b_i) = [\phi_1(b_i), \dots, \phi_m(b_i)]^T, \tag{12}$$

where $\phi_j(a_i) = 0$ if a_i is a satisfied assignment for ϕ_j, and $\phi_j(a_i) = 1$ otherwise; we define $\phi_i(b_i)$ in the same way. Let $F_A := \{f_A(a_i) : \ i \in [2^{n/2}]\}$ and $F_B := \{f_B(b_i) : \ i \in [2^{n/2}]\}$. Then, it is obvious that if there exist $v \in F_A$ and $u \in F_B$ such that $\langle v, u\rangle = 0$, then ϕ is satisfiable. However, at first glance, this reduction with $O(2^{n/2})$ running time is not fine-grained since we require the cost of the reduction to be at most $2^{n(1-\delta)/2}$ for some $\delta > 0$ by Definition 25, but writing down elements in F_A and F_B already takes $\Omega(2^{n/2})$.

Nevertheless, as in Definition 24, a quantum fine-grained reduction only needs to realize the functions f_A and f_B, which takes $O(mkn)$ time where k is the width of clauses. This is much less than $O(2^{n(1-\delta)/2})$. More specifically, f_A and f_B are oracles for F_A and F_B, and for any quantum query to elements in F_A or F_B, the reduction can implement oracles f_A and f_B:

$$|e, x\rangle |0\rangle \xrightarrow{f_e} |e, x\rangle |f_e(x)\rangle , \tag{13}$$

where $e \in \{A, B\}$, and the time $c(f_e)$ for the reduction to implement f_e for one quantum query is at most $O(kmn)$. Finally, this reduction only uses one oracle (F_A, F_B). If there is an algorithm for OV which succeeds with probability $2/3$, we can boost the success probability of the reduction by repetition. Therefore, $(\mathsf{CNF\text{-}SAT}, 2^{n/2})$ is quantum reducible to $(\mathsf{OV}_{n_1,d_1}, n_1)$. ◀

Then, to prove $(\mathsf{CNF\text{-}SAT}, 2^{n/2}) \leq_{\mathrm{QFG}} (\mathsf{CP}_{n_3,d_3}, n_3)$, we show that $(\mathsf{BCP}_{n_2,d_2}, n_2) \leq_{\mathrm{QFG}} (\mathsf{CP}_{n_3,d_3}, n_3)$ and $(\mathsf{OV}_{n_1,d_1}, n_1) \leq_{\mathrm{QFG}} (\mathsf{BCP}_{n_2,d_2}, n_2)$, where n_2, n_3, d_2, d_3 are some functions of n specified in the following lemmas.

▶ **Lemma 28.** *For $d = \Theta(\log n)$,*

$$(\mathsf{BCP}_{n,d}, n) \leq_{\mathrm{QFG}} (\mathsf{CP}_{n',d'}, n'), \tag{14}$$

where $n' = n^{O(1)}$ and $d' = (\log n)^c$ for some constant c and all points have $\{0,1\}$ entries with the Hamming metric.

▶ Remark 29. The points have coordinate entries in $\{0,1\}$, and the Hamming metric is equivalent to distance in ℓ_2-metric (up to power of 2) in this case. Therefore, in the proof of Lemma 28, we can consider the Hamming distance between points instead of ℓ_2 distance without loss of generality.

We first introduce the classical reductions in [29] and some results we will use to prove Lemma 28.

Classical reduction

We can consider an instance of BCP with two sets of points A and B as a weighted complete bipartite graph $K_{n,n}$, where the vertices are the points in these two sets and edges' weights are equal to the distances between the corresponding points. Then, solving BCP is equivalent to find an edge with the minimum weight in this graph. However, we cannot directly apply the algorithm for CP on this graph since there could be two points in the same set (no edge connecting them) that have a smaller distance than any pairs of points in two sets (connected by an edge). To overcome this difficulty, we can "stretch" the points to make the points in the same set far from each other, which is characterized by the contact dimension of a graph:

▶ **Definition 30** (Contact Dimension). *For any graph $G = (V, E)$, a mapping $\tau : V \to \mathbb{R}^d$ is said to realize G if for some $\beta > 0$, the following holds for every distinct vertices u, v:*

$$\begin{aligned} &\|\tau(u) - \tau(v)\|_2 = \beta \text{ if } \{u, v\} \in E, \\ &\|\tau(u) - \tau(v)\|_2 > \beta \text{ otherwise.} \end{aligned} \tag{15}$$

The contact dimension of G, denoted by $\mathrm{cd}(G)$, is the minimum $d \in \mathbb{N}$ such that there exists $\tau : V \to \mathbb{R}^d$ realizing G.

That is, with the help of τ, we can restrict the optimal solution of CP to be the points connected by an edge in G. But we cannot realize the whole complete bipartite graph since $\mathrm{cd}(K_{n,n}) = \Theta(n)$, which makes the dimension of the CP instance too large. [29] showed that we can realize a subgraph of $K_{n,n}$ and apply permutations to its vertices such that the union of these subgraphs cover $K_{n,n}$. In this way, BCP can be computed by solving CP on each subgraph and outputting the best solution. More specifically, the reduction in [29] relies on the following theorem:

▶ **Theorem 31** (Theorem 4.2 in [29]). *For every $0 < \delta < 1$, there exists a log-dense sequence $(n_i)_{i\in\mathbb{N}}$ such that, for every $i \in \mathbb{N}$, there is a bipartite graph $G_i = (A_i \dot\cup B_i, E_i)$ where $|A_i| = |B_i| = n_i$ and $|E_i| \geq \Omega(n_i^{2-\delta})$, such that $\mathrm{cd}(G_i) = (\log n_i)^{O(1/\delta)}$. Moreover, for all $i \in \mathbb{N}$, a realization $\tau : A_i \dot\cup B_i \to \{0,1\}^{(\log n_i)^{O(1/\delta)}}$ of G_i can be constructed in $n_i^{2+o(1)}$ time.*

The log-dense sequence is defined as follows:

▶ **Definition 32.** *A sequence $(n_i)_{i\in\mathbb{N}}$ of increasing positive integers is log-dense if there exists a constant $c \geq 1$ such that $\log n_{i+1} \leq c \cdot \log n_i$ for all $i \in \mathbb{N}$.*

They also showed that, the permutations for covering the complete bipartite graph can be efficiently found, as shown in the following lemma.

▶ **Lemma 33** (Lemma 3.11 in [29])**.** *For any bipartite graph $G(A\dot\cup B, E_G)$ where $|A| = |B| = n$ and $E_G \neq \emptyset$, there exist side-preserving permutations $\pi_1, \ldots, \pi_k : A \cup B \to A \cup B$ where $k \leq \frac{2n^2 \ln n}{|E_G|} + 1$ such that*

$$\bigcup_{i\in[k]} E_{G_{\pi_i}} = E_{K_{n,n}}. \tag{16}$$

Moreover, such permutations can be found in $O(n^6 \log n)$ time.

Now, we are ready to state the quantum fine-grained reduction by "quantizing" the classical reduction.

Proof of Lemma 28. Let A, B be the two sets of input points of BCP. Suppose for BCP, there is an input oracle $\mathcal{O}_{\mathsf{BCP}}$ which, given an index, returns the corresponding point:

$$|b\rangle|i\rangle|0\rangle \xrightarrow{\mathcal{O}_{\mathsf{BCP}}} \begin{cases} |b\rangle|i\rangle|x_i\rangle & \text{if } b = 0, \\ |b\rangle|i\rangle|y_i\rangle & \text{if } b = 1, \end{cases} \tag{17}$$

where x_i is the i-th point in the set A and y_i is the i-th point in the set B. The sizes of A and B are both equal to n and each point is in $\{0,1\}^{d_1}$, where $d_1 = \Theta(\log n)$ is the dimension of BCP.

For CP, suppose there is a quantum algorithm $\mathcal{A}$ such that for m points in $\{0,1\}^{d_2}$ given by an oracle $\mathcal{M}_{CP}$, $\mathcal{A}^{\mathcal{M}_{CP}}$ returns the closest pair of these n points with probability at least $2/3$.

Then we need to transform $\mathcal{O}_{\mathsf{BCP}}$ to some oracles $\mathcal{M}_i$ for CP, such that by running $\mathcal{A}$ with $\mathcal{M}_i$ as input oracles, we can get the bichromatic closest pair between A and B. The reduction has four steps:

1. Pre-processing

We first follow the classical reduction to pre-process the input points of BCP. For some integer $n' \leq n^{0.1}$, we can partition A and B into n'-size subsets:

$$\begin{aligned} A &= A_1 \dot\cup \cdots \dot\cup A_r, \\ B &= B_1 \dot\cup \cdots \dot\cup B_r, \end{aligned} \tag{18}$$

where $r = \lfloor n/n' \rfloor$. Here, we assume that n is divisible by n'. It follows that

$$\mathsf{BCP}(A, B) = \min_{i,j\in[r]} \mathsf{BCP}(A_i, B_j). \tag{19}$$

Then, we use the algorithm in [29] to construct k mappings $f_1, \ldots, f_k : [2n'] \to \{0,1\}^{d'}$ such that

$$\mathsf{BCP}(A_i, B_j) = \min_{t\in[k]} \mathsf{CP}(f_t(A_i) \cup f_t(B_j)) \quad \forall i, j \in [\lfloor n/n' \rfloor]. \tag{20}$$

More specifically, we pick n' to be the largest number in a log-dense sequence that is smaller than $n^{0.1}$. Then, we apply Theorem 31 to classically construct a bipartite graph $G(A \cup B, E)$ with n' vertices in each side and a realization τ. By choosing $\delta = \epsilon/2$ in Theorem 31, the graph G has $|E| = \Omega(n'^{2-\epsilon/2})$ edges. And we can get $2n'$ 0/1-strings of length $(\log n')^{O(2/\epsilon)}$:

$$\tau_i^A = \tau(u_i) \quad \forall u_i \in A, \quad \text{and} \quad \tau_i^B = \tau(v_i) \quad \forall v_i \in B. \tag{21}$$

In order to cover the complete bipartite graph, we run the classical algorithm (Lemma 33) to find k permutations $\pi_1, \ldots, \pi_k : [n'] \to [n']$, where k is a parameter to be chosen later.

Then, we can define the mappings as follows:

$$f_t(u) = \begin{cases} x_v \circ \left(\tau^A_{\pi_t(w)}\right)^{d+1} & \text{if } 1 \le u \le n' \\ y_v \circ \left(\tau^B_{\pi_t(w)}\right)^{d+1} & \text{if } n' < u \le 2n' \end{cases} \qquad \forall t \in [k], u \in [2n'], \tag{22}$$

where $\circ$ means string concatenation and $(s)^{d+1}$ denotes $d+1$ copies of the string s. For a point $p \in A_i \cup B_j$, $u \in [2n']$ is the index in this union-set, $v \in [n]$ is the index in the ground set A or B, and $w \in [n']$ is the index in the subset A_i or B_j. Further, if $1 \le u \le n'$, then $w := u$; otherwise, $w := u - n'$.

2. Oracle construction

For $i, j \in [r], t \in [k]$, we then construct the input oracle $\mathcal{M}_{i,j,t}$ for the problem $\mathsf{CP}(f_t(A_i) \cup f_t(B_j))$. For a query index $u \in [2n']$,

$$M_{i,j,t} |u\rangle |0\rangle = |u\rangle |f_t(u)\rangle . \tag{23}$$

With the help of the input oracle $\mathcal{O}_{\mathsf{BCP}}$, we can implement $\mathcal{M}_{i,j,t}$ in the following way:

1. Prepare an ancilla qubit $|b\rangle$ such that $b = 1$ if $u > n'$.
2. Transform $|u\rangle$ to $|v\rangle$, the index of the point in A or B, based on the value of b. Note that the index is unique. Hence, this transformation is unitary and can be easily achieved by a small quantum circuit.
3. Query $\mathcal{O}_{\mathsf{BCP}}$ with input $|b\rangle |v\rangle$. Assume $b = 0$. Then,

$$|b\rangle |v\rangle |0\rangle \xmapsto{\mathcal{O}_{\mathsf{BCP}}} |b\rangle |v\rangle |x_v\rangle . \tag{24}$$

4. Similar to the second step, the index w of the point in A_i and B_j can be computed from v by a unitary transformation:

$$|b\rangle |v\rangle |x_v\rangle \mapsto |b\rangle |w\rangle |x_v\rangle \tag{25}$$

5. Since each w corresponds to a unique string $\tau^A_{\pi_t(w)}$, we can attach $d+1$ copies of this string to the remaining quantum registers:

$$|b\rangle |w\rangle |x_v\rangle \mapsto |b\rangle |w\rangle |x_v\rangle \left| \left(\tau^A_{\pi_t(w)}\right)^{d+1} \right\rangle . \tag{26}$$

6. By recovering u from w, we get the final state:

$$|u\rangle |f_t(u)\rangle = |u\rangle \left| x_v, \left(\tau^A_{\pi_t(w)}\right)^{d+1} \right\rangle . \tag{27}$$

3. Query process

By Equations (19) and (20), we have

$$\mathsf{BCP}(A,B) = \min_{i,j\in[r],t\in[k]} \mathsf{CP}(f_t(A_i)\cup f_t(B_j)). \tag{28}$$

Hence, we can use quantum minimum-finding algorithm in Theorem 15 over the sub-problems to find the minimum solution. For each sub-problem, we can run the algorithm for CP with $\mathcal{M}_{i,j,t}$ as the input oracle.

4. Post-processing

In case that n is not divisible by n', let the remaining points in A and B be A_{res}, B_{res}, respectively. Then, we can use Grover search to find the closest pair between A_{res} and B, and between B_{res} and A. Then, compare the answer to the previously computed result and pick the smaller one.

Correctness

In this reduction, we do not change the constructions of the mappings $\{f_i\}_{i\in[k]}$. By [29], Equation (28) is correct in the classical setting. Hence, it also holds in the quantum setting, and we can use Grover search to find the minimum solution. However, since the algorithm $\mathcal{A}$ for CP has success probability 2/3, for each tuple $(i,j,t)\in[r]\times[r]\times[k]$, we need to run $\mathcal{A}^{\mathcal{M}_{i,j,t}}$ $O(\log n)$ times to boost the success probability to at least $1-\frac{1}{n}$. Then, by the union bound, the probability that all queries in the Grover search are correct is at least 99/100. Hence, by Theorem 9, the overall success probability is at least 2/3.

Running Time of the Reduction

The running time of the pre-processing step consists of two parts: (1) constructing the graph G and its realization τ; (2) finding k permutations. For the first part, by Theorem 31, it can be done in $n'^{2+o(1)}$ time. For the second part, we pick $k = O(\frac{2n'^2\log n'}{n'^{2-\epsilon/2}}) = O(n'^{\epsilon/2}\log n')$, and by Lemma 33, it can be done in $O(n'^6\log n')$ time. Hence, the total running time of pre-processing step is $n'^{2+o(1)} + O(n'^6\log n') = \widetilde{O}(n^{0.6})$.

The oracle construction can be done "on-the-fly". More specifically, given the strings $\{\tau_i^A,\tau_i^B\}_{i\in[n']}$, and permutations $\{\pi_i\}_{i\in[k]}$, for each query index u, we can simulate the oracle $\mathcal{M}_{i,j,t}$ defined in Equation (23) in $c(\mathcal{M}_{i,j,t}) = O(d_2) = (\log n')^{\Omega(1)} = \widetilde{O}(1)$ time.

In the query process, for each CP instance indexed by (i,j,t), suppose $\mathcal{A}^{\mathcal{M}_{i,j,t}}$ gets the answer in time $q(n') = n'$. Moreover, for each time $\mathcal{A}$ querying the input oracle $\mathcal{M}_{i,j,t}$, we need to spend $c(\mathcal{M}_{i,j,t})$ time to simulate the oracle. And we also have $O(\log n)$ runs for each instance. Hence, the total running time for each CP is at most

$$n'^{1-\epsilon}\cdot\widetilde{O}(1)\cdot O(\log n) = \widetilde{O}(n'^{1-\epsilon}). \tag{29}$$

Then, we use Grover's search algorithm over $r^2\cdot k$ instances, which can be done by querying $\widetilde{O}(\sqrt{r^2\cdot k})$ instances by Theorem 9. Therefore, for any $\epsilon>0$, we have

$$\widetilde{O}(\sqrt{r^2k})\cdot q(n')^{1-\epsilon}\cdot c(\mathcal{M}_{i,j,t})\cdot O(\log n) = \widetilde{O}(\sqrt{(n/n')^2k}\cdot(n')^{1-\epsilon}) \tag{30}$$

$$\leq \widetilde{O}(n\cdot(n')^{-\epsilon}) \leq \widetilde{O}(n\cdot n^{-\epsilon/2}) \leq n^{1-\delta}, \tag{31}$$

where the first inequality follows from $k = O(n'^{\epsilon/2}\log n')$ as shown in [29] and the last inequality follows by setting $\delta = \epsilon/10$.

For the post-processing step, the sizes of A_{res} and B_{res} are at most n'. The running time is

$$O(\sqrt{n \cdot n'} \cdot \log n) \leq \widetilde{O}(n^{0.55}). \tag{32}$$

Therefore, for any $\epsilon > 0$, there exists a $\delta > 0$ such that the Equation (30) holds and the total reduction time is $O(n^{1-\delta})$. By Definition 25, BCP_{n,d_1} can be quantum fine-grained reduced to CP_{n,d_2}. This completes the proof of this lemma. ◀

Finally, we show that $(\mathsf{OV}_{n,d}, n) \leq_{\mathrm{QFG}} (\mathsf{BCP}_{n,d'}, n)$ by quantizing the reduction in [29] following the same idea.

▶ **Lemma 34.** *For $d = \Theta(\log n)$,*

$$(\mathsf{OV}_{n,d}, n) \leq_{\mathrm{QFG}} (\mathsf{BCP}_{n,d'}, n), \tag{33}$$

where $d' = \Theta(\log n)$.

Proof. For an OV instance with sets of vectors A and B, let $\mathcal{O}_{\mathsf{OV}}$ be the input oracle such that:

$$\mathcal{O}_{\mathsf{OV}} |i\rangle |0\rangle = \begin{cases} |i\rangle |a_i\rangle & \text{if } i \in A, \\ |i\rangle |b_i\rangle & \text{if } i \in B. \end{cases} \tag{34}$$

where $a_i, b_i \in \{0,1\}^d$.

Then, similar to the classical reduction, we can construct mappings $f_A, f_B : \{0,1\}^d \to \{0,1\}^{5d}$ such that

$$f_A(a_i)_{5j-4:5j} = \begin{cases} 11000 & \text{if } a_i(j) = 0 \\ 00110 & \text{if } a_i(j) = 1 \end{cases} \quad \forall j \in [d], \tag{35}$$

and

$$f_B(b_i)_{5j-4:5j} = \begin{cases} 10100 & \text{if } b_i(j) = 0, \\ 01001 & \text{if } b_i(j) = 1. \end{cases} \quad \forall j \in [d]. \tag{36}$$

By the classical reduction, we have

$$\mathsf{OV}(A, B) = 1 \text{ if and only if } \mathsf{BCP}(f_A(A), f_B(B)) = 2d \tag{37}$$

under Hamming distance.

Also, note that we can simulate the input oracle $\mathcal{O}_{\mathsf{BCP}}$ by first querying the oracle $\mathcal{O}_{\mathsf{OV}}$ to get the vector, then applying the corresponding mapping f_A or f_B, which can be done in $c(\mathcal{O}_{\mathsf{BCP}}) = O(d)$ time. Let the running time of the algorithm for BCP be $q(n) = n$. Then for any $\epsilon > 0$,

$$q(n)^{1-\epsilon} \cdot c(\mathcal{O}_{\mathsf{BCP}}) = n^{1-\epsilon} \cdot \Theta(\log n) \leq n^{1-\delta} \tag{38}$$

for some small $\delta > 0$. Hence, by Definition 25, $(\mathsf{OV}_{n,d}, n) \leq_{\mathrm{QFG}} (\mathsf{BCP}_{n,d'}, n)$. ◀

Proof of Theorem 26. We can prove the theorem by contradiction following Lemma 27, Lemma 34, and Lemma 28. Specifically, suppose that there exists an $\epsilon > 0$, for all $d = \Theta(\log n)$, there exists a quantum algorithm which can solve OV in time $O(n^{1-\epsilon})$. Then, we can obtain a quantum algorithm for $\mathsf{CNF\text{-}SAT}$, which runs in time $O(2^{n/2(1-\epsilon)})$ by Lemma 27. This contradicts QSETH. The proof for CP is the same. ◀

3.3 Quantum lower bound for BCP in nearly-constant dimensions under QSETH

A byproduct of the previous subsection is a quantum lower bound for BCP in higher dimensions (i.e., $d = \text{polylog}(n)$) under QSETH (Lemma 34). In this subsection, we show that this quantum lower bound for BCP even holds for nearly-constant dimensions (i.e., $d = c^{\log^*(n)}$). The main result of this subsection is the following theorem.

▶ **Theorem 35.** *Assuming QSETH, there is a constant c such that* BCP *in $c^{\log^*(n)}$ dimensions requires $n^{1-o(1)}$ time for any quantum algorithm.*

We will "quantize" the results by Chen [17] to prove this theorem. More specifically, we first show a quantum fine-grained self-reduction of OV from $\log n$ dimensions with binary entries to $2^{\log^*(n)}$ dimensions with integer entries ($\mathbb{Z}$-OV). Then, we give a quantum fine-grained reduction from $\mathbb{Z}$-OV to BCP in nearly-constant dimensions.

▶ **Definition 36** (Integral Orthogonal Vector, $\mathbb{Z}$-OV)**.** *Given two sets A, B of n vectors in $\mathbb{Z}^d$, find a pair of vectors $a \in A$ and $b \in B$ such that $\langle a, b\rangle = 0$, where the inner product is taken in $\mathbb{Z}$.*

We use $\mathbb{Z}\text{-OV}_{n,d}$ to denote $\mathbb{Z}$-OV with n vectors of d dimension in each set. We then recap a theorem in [17]:

▶ **Theorem 37** ([17, Theorem 4.1])**.** *Let b, ℓ be two sufficiently large integers. There is a classical reduction $\psi_{b,\ell} : \{0,1\}^{b\cdot\ell} \to \mathbb{Z}^\ell$ and a set $V_{b,\ell} \subseteq \mathbb{Z}$, such that for every $x, y \in \{0,1\}^{b\cdot\ell}$,*

$$\langle x, y\rangle = 0 \;\Leftrightarrow\; \langle \psi_{b,\ell}(x), \psi_{b,\ell}(y)\rangle \in V_{b,\ell} \tag{39}$$

and

$$0 \le \psi_{b,\ell}(x)_i < \ell^{6^{\log^*(b)}\cdot b} \tag{40}$$

for all possible x and $i \in [\ell]$. Moreover, the computation of $\psi_{b,\ell}(x)$ takes $\text{poly}(b\cdot\ell)$ time, and the set $V_{b,\ell}$ can be constructed in $O\left(\ell^{O(6^{\log^*(b)}\cdot b)} \cdot \text{poly}(b\cdot\ell)\right)$ time.

Note that the size of $V_{b,\ell}$ is at most $\ell^{2\cdot 6^{\log^*(b)}\cdot b+1}$. The following lemma gives a quantum fine-grained reduction from OV to $\mathbb{Z}$-OV:

▶ **Lemma 38.** *For $d = \Theta(\log n)$,*

$$(\text{OV}_{n,d}, n) \le_{\text{QFG}} (\mathbb{Z}\text{-OV}_{n,d'}, n). \tag{41}$$

where $d' = 2^{O(\log^* n_2)}$.

Proof. Consider an $\text{OV}_{n,d}$ with $d = c\cdot\log n$, where c is an arbitrary constant. We choose $\ell := 7^{\log^* n}$ and $b := d/\ell$. Then, we can apply Theorem 37 to get the mapping function $\psi_{b,\ell}$ and the set $V_{b,\ell}$. For each $v \in V_{b,\ell}$, we'll construct an instance of $\mathbb{Z}\text{-OV}_{n,\ell+1}$ as follows:

1. Let $|i\rangle$ be the input query index of $\mathbb{Z}\text{-OV}_{n,\ell+1}$.
2. Query $\text{OV}_{n,d}$'s input oracle $\mathcal{O}_{\text{OV}}$ and get the vector $|i, x\rangle$.
3. Compute the mapping $\psi_{b,\ell}$ and get $|i, x\rangle|\psi_{b,\ell}(x)\rangle$.
4. If $x \in A$, then attach 1 to the end of the register: $|i, x\rangle|\psi_{b,\ell}(x), 1\rangle$. If $x \in B$, then attach $-v$ to the end: $|i, x\rangle|\psi_{b,\ell}(x), -v\rangle$.
5. Use $\mathcal{O}_{\text{OV}}$ to erase x and return the final input state $|i\rangle|\psi_{b,\ell}(x), 1\rangle$ or $|i\rangle|\psi_{b,\ell}(x), -v\rangle$.

For each instance, we can use the quantum oracle for $\mathbb{Z}\text{-OV}_{n,\ell+1}$ to check the orthogonality. $\text{OV}_{n,d}$ is YES if and only if there exists a YES-instance of $\mathbb{Z}\text{-OV}_{n,\ell+1}$.

Correctness

The correctness follows from Equation (39):

$$\langle x, y\rangle = 0 \Leftrightarrow \langle \psi_{b,\ell}(x), \psi_{b,\ell}(y)\rangle = v \in V_{b,\ell} \Leftrightarrow \langle [\psi_{b,\ell}(x),\ 1], [\psi_{b,\ell}(y),\ -v]\rangle = 0. \tag{42}$$

Reduction time

Note that for $\ell = 7^{\log^* n}$ and $b = d/\ell$, we have:

$$\log\left(\ell^{O(6^{\log^*(d)}\cdot b)}\right) = \log \ell \cdot O\left(6^{\log^*(d)} \cdot (d/\ell)\right) \tag{43}$$

$$= O\left(\log^*(n) \cdot 6^{\log^* n} \cdot c \log n / 7^{\log^* n}\right) \tag{44}$$

$$= o(\log n). \tag{45}$$

This implies that $|V_{b,\ell}| \leq \ell^{2\cdot 6^{\log^*(b)}\cdot b+1} \leq n^{o(1)}$. Hence, the number of $\mathbb{Z}$-$\mathsf{OV}_{n,\ell+1}$ instances is $n^{o(1)}$ and the running time for compute $V_{b,\ell}$ is $n^{o(1)}$. And for each input query, the oracle for $\mathbb{Z}$-$\mathsf{OV}_{n,\ell+1}$ can be simulated in $c(\mathcal{O}_{\mathbb{Z}\text{-}\mathsf{OV}}) = \mathrm{poly}(d) = \mathrm{poly}(\log n)$ time. We can show that for every $\epsilon > 0$, if $\mathbb{Z}$-$\mathsf{OV}_{n,\ell+1}$ can be decided in $n^{1-\epsilon}$ time, then

$$\sum_{v\in V_{b,\ell}} n^{1-\epsilon} \cdot c(\mathcal{O}_{\mathbb{Z}\text{-}\mathsf{OV}}) = n^{o(1)} \cdot n^{1-\epsilon} \cdot \mathrm{poly}(\log n) \leq n^{1-\delta} \tag{46}$$

for some $\delta > 0$, which satisfies the definition of quantum fine-grained reduction (Definition 25).

Therefore, $\mathsf{OV}_{n,O(\log n)}$ is quantum fine-grained reducible to $\mathbb{Z}$-$\mathsf{OV}_{n,2^{O(\log^*(n))}}$. ◀

Then, we give a quantum fine-grained reduction from $\mathbb{Z}$-OV to BCP:

▶ **Lemma 39.** *For $d = 2^{O(\log^* n)}$,*

$$(\mathbb{Z}\text{-}\mathsf{OV}_{n,d}, n) \leq_{\mathrm{QFG}} (\mathsf{BCP}_{n,d'}). \tag{47}$$

where $d' = d^2 + 2$.

Proof. We remark here that this proof closely follows that for Theorem 4.3 in [17]. We nonetheless give it here as some details are different.

For an $\mathbb{Z}$-$\mathsf{OV}_{n,d}$ instance with $(k \cdot \log n)$-bit entries, we construct a BCP instance as follows:

1. For $x \in A$, construct a vector $x' \in \mathbb{Z}^{d^2}$ such that $x'_{i,j} = x_i \cdot x_j$. Here, we index a d^2-dimensional vector by $[d] \times [d]$. Similarly, for $y \in B$, construct a vector $y' \in \mathbb{Z}^{d^2}$ such that $y'_{i,j} = -y_i \cdot y_j$.
2. Choose $W := (d^2+1) \cdot n^{4k}$. For each x', construct a vector $x'' \in \mathbb{R}^{d^2+2}$ such that

$$x'' = \left[x',\ \sqrt{W - \|x'\|_2^2},\ 0\right]. \tag{48}$$

For each y', construct a vector $y'' \in \mathbb{R}^{d^2+2}$ such that

$$y'' = \left[y',\ 0,\ \sqrt{W - \|y'\|_2^2}\right]. \tag{49}$$

Then, we claim that the $\mathbb{Z}$-OV instance is YES if and only if the BCP instance has the minimum distance $\leq \sqrt{2W}$.

Correctness

First note that $\|x'\|_2^2 \leq d^2 \cdot (2^{k \log n})^4 = d^2 \cdot n^{4k}$. Hence, $W - \|x'\|_2^2 > 0$ and $W - \|y'\|_2^2 > 0$. For any x'' and y'' in the new constructed instance of BCP, we have

$$\begin{aligned} \|x'' - y''\|_2^2 &= \|x''\|_2^2 + \|y''\|_2^2 - 2 \cdot \langle x'', y'' \rangle && (50) \\ &= 2 \cdot W - 2 \cdot \langle x', y' \rangle && (51) \\ &= 2 \cdot W - 2 \cdot \sum_{(i,j) \in [d] \times [d]} x_i \cdot x_j \cdot (-y_j \cdot y_j) && (52) \\ &= 2 \cdot W + 2 \cdot (\langle x, y \rangle)^2. && (53) \end{aligned}$$

Hence,

$$\langle x, y \rangle = 0 \;\Leftrightarrow\; \|x'' - y''\|_2^2 = 2W. \tag{54}$$

Reduction time

We can see from the above description that the input mapping function is simple and can be computed by a small quantum circuit in $O(d^2) = O(2^{O(\log^*(n))})$ time. Hence, we have $c(\mathcal{O}_{\mathsf{BCP}}) = O(2^{O(\log^*(n))})$. Also, by Definition 25, it's easy to check that this is indeed a quantum fine-grained reduction from $\mathbb{Z}$-OV to BCP. ◀

Now Theorem 35 follows immediately from Lemma 38 and Lemma 39:

Proof of Theorem 35. Let $\epsilon > 0$ be some constant. Suppose we can solve $\mathsf{BCP}_{n,c^{\log^*(n)}}$ in $n^{1-\epsilon}$ time for all constant $c > 0$. Then, by Lemma 38 and Lemma 39, we can also solve $\mathsf{OV}_{n,c' \log n}$ in $n^{1-\delta}$ time for some $\delta > 0$ and any $c' > 0$. However, this contradicts QSETH by Theorem 26. Therefore, assuming QSETH, there exists a constant c such that $\mathsf{BCP}_{n,c^{\log^*(n)}}$ requires $n^{1-o(1)}$ time. ◀

4 Closest pair in constant dimension

In this section, we show that there exist almost-optimal quantum algorithms for CP in constant dimension. The main result is the following theorem, which is a direct consequence of Corollary 55 and Theorem 56.

▶ **Theorem 40.** *For any constant dimension, the quantum time complexity for* CP *is* $\widetilde{\Theta}(n^{2/3})$.

Our approach to solve CP is first reducing to the decision version of the problem, and then apply quantum walk algorithms to solve the decision version. We define the decision version of CP, CP_ϵ, as follows.

▶ **Definition 41** (CP_ϵ)**.** *Given a set of points $P \subset \mathbb{R}^d$ and $\epsilon \in \mathbb{R}$, find a pair $a, b \in P$ such that $\|a - b\| \leq \epsilon$ if there is one and returns* no *is no such pair exists.*

The reduction from CP to CP_ϵ is given by the following lemma.

▶ **Lemma 42.** *Let m be the number of bits needed to encode each coordinate as a bit string and d be the dimension. Given an oracle $\mathcal{O}$ for CP_ϵ, there exists an algorithm $A^{\mathcal{O}}$ that runs in time and query complexity $O(m + \log d)$ that solves the* CP.

Proof. Let (P, δ) be an instance of the CP. We first pick an arbitrary pair $a_0, b_0 \in P$ and compute $\Delta(a_0, b_0)$. Then, we set ϵ to be $\Delta(a_0, b_0)/2$ and run the oracle $\mathcal{O}$ to check whether there exists a distinct pair with distance less than $\Delta(a_0, b_0)/2$ or not. If there exists such a pair, which we denote as (a_1, b_1), then we set $\epsilon = \Delta(a_1, b_1)$ and call $\mathcal{O}$ to check again. If there is no such pair, then we set $\epsilon = 3\Delta(a_0, b_0)/4$ and call $\mathcal{O}$. We run this binary search for $m + \log d$ iterations. Finally, the algorithm outputs the closest pair. ◀

In classical setting, point location is an important step in solving the closest-pair problem, especially the dynamic version. For the quantum algorithm, as walking on the Markov chain, we repeatedly delete a point and add a new point. Hence, in each step, the first thing is to determine the location of the new added point.

For simplicity, we assume that $m = O(\log n)$, which is the number of digits of each coordinate of the points. By translation, we can further assume that all the points are lying in $[0, L]^d$, where $L = O(2^m) = \text{poly}(n)$.

Since we are considering CP_ϵ, one simple way of point location is to discretize the whole space into a hypergrid, which is defined as follows:

▶ **Definition 43.** *Let $d, \epsilon, L > 0$. A hypergrid $G_{d,\epsilon,L}$ in the space $[0, L]^d$ consists of all ϵ-boxes*

$$g := [a_1, b_1) \times [a_2, b_2) \times \cdots \times [a_d, b_d), \tag{55}$$

such that $b_1 - a_1 = \cdots = b_d - a_d = \epsilon/\sqrt{d}$ [VII], and a_i is divisible by ϵ for all $i \in [d]$.

For each point $p_i \in [0, L]^d$, we can identify the ϵ-box that contains it using the function $\text{id}(p_i) : [0, L]^d \to \{0, 1\}^{d \log(L/\epsilon)}$:

$$\text{id}(p_i) = \left(\lfloor p_i(1)/w \rfloor, \lfloor p_i(2)/w \rfloor, \ldots, \lfloor p_i(d)/w \rfloor \right), \tag{56}$$

where $w = \frac{\epsilon}{\sqrt{d}}$ is the width of the ϵ-box. The number of bits to store $\text{id}(p_i)$ is $d \cdot \log(L/w) = O(d \cdot \log(L))$. Since all the points in an ϵ-box have the same id, we also use this $g(\text{id}(p))$ to denote this ϵ-box containing p.

For the ease of our analysis, we define the neighbors of a hypergrid.

▶ **Definition 44.** *Let $\epsilon \in \mathbb{R}$. Let g_1, g_2 be two ϵ-boxes in a hypergrid where $\text{id}(g_1) = (x_1, \ldots, x_d)$ and $\text{id}(g_2) = (x'_1, \ldots, x'_d)$. We say that g_1 and g_2 are each other's* ϵ-neighbor *if*

$$\sqrt{\sum_{i=1}^{d} \|x_i - x'_i\|^2} \leq \epsilon \tag{57}$$

Note that the number of ϵ-neighbors of a ϵ-box is at most $(2\sqrt{d} + 1)^d$. We also have the following observation:

▶ **Observation 45.** *Let $p_1, p_2 \in [0, L]^d$ be any two distinct points.*

- *If p_1 and p_2 are in the same ϵ-box, then $\Delta(p_1, p_2) \leq \epsilon$.*
- *If $\Delta(p_1, p_2) \leq \epsilon$, then $g(\text{id}(p_1))$ must be an ϵ-neighbor of $g(\text{id}(p_2))$.*

To solve CP_ϵ with quantum walk, we need data structures to keep track of the pairs that have distance at most ϵ. The desired data structure should have size $\widetilde{O}(n^{2/3})$, insertion/deletion time $O(\log n)$, and one should be able to check whether there exist pairs of distance at most ϵ in time $O(\log n)$. In addition, as pointed out in [5], the data structure should have the following two properties:

[VII] The diagonal length of an ϵ-box is ϵ.

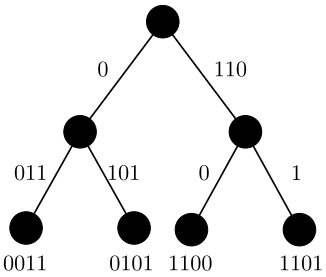

Figure 2 The uniquely represented radix tree that stores the keys $\{0011, 0101, 1100, 1101\}$.

- the data structure should have the bounded *worst-case* performance rather than *average-case* performance;
- the representation of the data structure should be history-independent, i.e., the data is uniquely represented regardless of the order of insertions and deletions.

We need the first property since the data structure may take too long for some operations, and this is not acceptable. The second property is required because, otherwise, the interference of quantum states would be messed up. In [5], a hash table and a skip list is used to for solving the element distinctness problem using quantum walks. In [11], a simpler data structure, namely, a radix tree, is used to achieve the same performance. More details of using a radix tree to solve the element distinctness can be found in [28]. Similar to the quantum data structure model in [5, 11, 28], we need the *quantum random access gate* to efficiently access data from a quantum memory, whose operation is defined as:

$$|i, b, z_1, \ldots, z_m\rangle \mapsto |i, z_i, z_1, \ldots, z_{i-1}, b, z_{i+1}, z_m\rangle\,, \tag{58}$$

where $|z_1, \ldots, z_m\rangle$ is some data in a quantum memory with m qubits. We assume this operation takes $O(\log m)$ time.

In the remainder of this section, we present two quantum algorithms for solving CP_ϵ. The data structures of both versions are based on the *augmented radix tree*, which we discuss in detail in the following subsection.

4.1 Radix tree for at most one solution

The purpose of the augmented radix tree is to quickly locate the points in an ϵ-box given its id. An ordinary radix tree is a binary tree that organizes a set of keys which are represented as binary strings. Each edge is labeled by a substring of a key and each leaf is labeled by a key such that concatenating all the labels on the path from the root to a leaf yields the key for this leaf. In addition, for each internal node, the labels of the two edges connecting to two children start with different bit. Note that in this definition, we implicitly merge all internal nodes that have only one child. The radix tree is uniquely represented for any set of keys. An example of a radix tree is shown as Figure 2.

Our basic radix tree is essentially the one in [11, 28] with modification on the nodes' internal structure. We highlight the extra information stored in the radix tree. First we use a *local counter* to store the number of points in this ϵ-box; second, we use a *flag* in each leaf node to indicate whether there is a point in this ϵ-box that is in some pair with distance at most ϵ. The flag bit in an internal node is the OR of the ones in its children. The local counter in each internal node is the sum of the local counters in its children. We also store at

most two points that are in the ϵ-box corresponding to this node. More precisely, let S be a subset of indices of the input points. We use $\tau(S)$ to denote the radix tree associated with S. Then, $\tau(S)$ consists of at most $r\lceil \log r \rceil$ nodes. Each node consists of the following registers:

$$\mathcal{D} \times \mathcal{M}_1 \times \mathcal{M}_2 \times \mathcal{M}_3 \times \mathcal{C} \times \mathcal{F} \times \mathcal{P}_1 \times \mathcal{P}_2, \tag{59}$$

where $\mathcal{D}$ stores the id of an ϵ-box for a leaf (and a substring of an id for an internal node) using $O(d \log(L/\epsilon))$ bits. $\mathcal{M}_1, \mathcal{M}_2$, and $\mathcal{M}_3$ use $O(\log n)$ bits to store the pointers to its parent, left child, and right child, respectively as well as the labels of the three edges connecting them to this node, $O(\log n)$ bits to store the labels of the three edges incident to it. $\mathcal{C}$ uses $O(\log n)$ bits to store the local counter. $\mathcal{F}$ stores the flag bit. $\mathcal{P}_1$ and $\mathcal{P}_2$ stores the coordinates of at most two points in this ϵ-box, which takes $O(d \log L)$ bits. The two points are stored in ascending order of their indices.

We need to pay attention to the layout of $\tau(S)$ in memory. We use three times more bits than needed to store $\tau(S)$, this will ensure that there are always more than $1/3$ of the bits that are free. We divide the memory into cells where each cell is large enough to store one leaf node of $\tau(S)$. Besides $\tau(S)$, we also store a bitmap $\mathcal{B}$, which takes $O(\log n)$ bits to encode the current free cells (with "1" indicating occupied and "0" indicating free). To make the radix tree history-independent, we use a quantum state which is the uniform superposition of basis states $|\tau(S), B\rangle$ for all possible valid layout of $\tau(S)$ and it corresponds to the bitmap $\mathcal{B}$.

Insertion and deletion from $\tau(S)$ takes $O(\log n)$ time. Checking the presence of an ϵ-close pair takes constant time – we just need to read the flag bit in the root. Preparing the uniform superposition of all $i \in S$ can be done in $O(\log n)$ time by performing a controlled-rotation on each level of the radix tree where the angles are determined by the local counters in the two children of a node.

In the following subsections, we present the two versions of our algorithms. The first version invokes the quantum walk framework only once and its data structure maintains the existence of an ϵ-close pair. The second version uses a much simpler data structure, but it is only capable of handling CP_ϵ with a unique solution. Hence it requires invoking the quantum walk framework multiple times to solve the general CP_ϵ. These two quantum algorithms have almost the same time complexity.

4.2 Single-shot quantum walk with complicated data structure

To handle multiple solutions, our data structure is a composition of an augmented radix tree, a hash table, and a skip list. We give a high-level overview of our data structure as follows. Recall that by the discretization of the space into ϵ-boxes, it is possible that a pair of points in different ϵ-boxes have distance at most ϵ, but one only needs to check $(2\sqrt{d}+1)^d$ ϵ-neighbors to detect such a case. We maintain a list of points for each nonempty ϵ-box in an efficient way. A hash table is used to store the tuple (i, p_i) which is used to quickly find the point p_i, given its index i. The points are also stored in a skip list for each nonempty ϵ-box, ordered by its index i, which allows for quick insertion and deletion of points. Each ϵ-box is encoded into a unique key, and a radix tree is used to store such key-value pairs, where the value is associated with a skip list. The flag bits in this radix tree maintain the presence of an ϵ-close pair.

In the following, we present the details of the data structure and show it has all the desired properties.

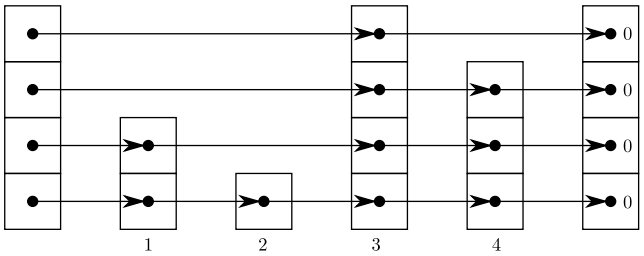

Figure 3 An example of a skip list that stores $\{1, 2, 3, 4\}$.

Hash table

The hash table we use is almost the same as the one used in [5], except that we do not store the $\lfloor \log r \rfloor$ counters in each bucket to facilitate the diffusion operator (which is handled easily here in the quantum walk on a Johnson graph). Our hash table has r buckets, where each bucket contains $\lceil \log n \rceil$ entries. We use a fixed hash function $h(i) = \lfloor ir/n \rfloor + 1$ to hash $\{1, \ldots, n\}$ to $\{1, \ldots, r\}$. That is, for $j \in [r]$, the j-th bucket contains the entries for (i, p_i) in ascending order of i, where $i \in S$ and $h(i) = j$.

The entry for (i, p_i) contains the tuple (i, p_i) and $\lceil \log n \rceil + 1$ pointers to other entries. These pointers are used in the skip list which we will describe below. The memory size of each entry is hence $O(\log^2 n + d \log L)$ and there are $O(r \log n)$ entries. Therefore, the hash table uses $O(rd \log^3(n + dL))$ qubits.

It is possible that more than $\lceil \log n \rceil$ points are hashed into the same bucket. However, as shown in [5], this probability is small.

Skip list

The skip list we use closely follows that in [5], except that the elements p_i in our skip list is ordered by its index i. We construct a skip list for each ϵ-box containing at least one point to store the points in it. For each $i \in S$, p_i belongs to exactly one skip list. Also, for $i \in S$, we randomly assign a level $\ell_i \in [0, \ldots, \ell_{\max}]$ where $\ell_{\max} = \lceil \log n \rceil$. The skip list associated with a ϵ-box has $\ell_{\max} + 1$ lists, where the level-ℓ list consists of all $i \in S$ such that $\ell_i \geq \ell$ and p_i is in this ϵ-box. Hence, the level-0 list consists of all $i \in S$ for p_i in this ϵ-box. Each element of the level-ℓ list has a specific pointer to the next element in this level, or to 0 if there is no next element. Each skip list contains a start entry that does not contain any (i, p_i) information but $\ell_{\max} + 1$ pointers to the first element of the each level. This start entry is stored in a leaf node of the augmented radix tree (which we will describe below) corresponding to this ϵ-box. In each skip list, we do not allocate memory for each node. Instead, each pointer is pointing to an entry of the hash table. The pointers are stored in the hash table (for the internal entries of each level) and in the radix tree (for the start entry). An example of a skip list is shown in Figure 3.

Given $i \in S$, we can search for p_i as follows. We start from the start entry of the level-$\ell_{\max}$ list and traverse each element until we find the last element $j_{\ell_{\max}}$ such that $j_{\ell_{\max}} < i$. Repeat this for levels $\ell_{\ell_{\max}-1}, \ldots, \ell_0$ and at each level start from the element that ended the previous level. At level-0, we obtain the element j_0. Then, the next element of j_0 is where p_i should be located (if it is stored in this skip list) or be inserted.

Each $i \in S$ is randomly assigned a level ℓ_i at the beginning of computation that does not change during the computation. More specifically, $\ell_i = \ell$ with probability $1/2^{\ell+1}$ for $\ell < \ell_{\max}$ and with probability $1/2^{\ell_{\max}}$ for $\ell = \ell_{\max}$. This can be achieved using $\ell_{\max}$

hash functions $h_1, \ldots, h_{\ell_{\max}} : [n] \to \{0,1\}$. In this way, each $i \in [n]$ has level $\ell < \ell_{\max}$ if $h_1(i) = \cdots = h_\ell(i) = 1$ but $h_{\ell+1}(i) = 0$; and it has level $\ell_{\max}$ if $h(i) = \cdots = h_{\ell_{\max}}(i) = 1$. In this quantum algorithm, we use an extra register to hold the state $|h_1, \ldots, h_{\ell_{\max}}\rangle$ which is initialized to a uniform superposition of all possible such functions from a d-wise independent family of hash functions (see [5, Theorem 1]) for $d = \lceil 4 \log n + 1 \rceil$. During the execution of the quantum algorithm, a hash function from the hashing family is chosen depending on the state in this register.

At first glance, the skip list has the same role as the hash table – finding p_i given index i. However, they have very different purposes in our algorithm. Recall that each nonempty ϵ-box is associated with a skip list, which is used to quickly insert and delete a point in this ϵ-box. The number of points in this ϵ-box can be as small as one and as large as r (in the extreme case where all the points are in the same ϵ-box). Hence, we cannot afford to have a fixed length data structure (such as a hash table or a sorted array) to store these points. In addition, to support quick insertion and deletion, a skip list is a reasonable choice (against an ordinary list). The purpose of the hash table can be viewed as a uniquely represented memory storing all the r points that can be referred to by the skip lists.

Augmented radix tree

We augment the radix tree described in Section 4.1 to handle multiple solution. In this augmented radix tree, we do not need the registers $\mathcal{P}_1$ and $\mathcal{P}_2$. Instead, we use $\lceil \log n \rceil$ pointers $\mathcal{L}_1, \ldots, \mathcal{L}_{\lceil \log n \rceil}$ as the start entry of a skip list. These pointers uses $O(\log^2 n)$ bits. In addition, we use an *external counter* in the leaf nodes to record whether there is a point in other ϵ-boxes that is at most ϵ-away from a point in this ϵ-box, which uses $O(\log n)$ bits. More formally, let $\tau'(S)$ be the augmented radix tree associated with S. Each node of $\tau'(S)$ consists of the following registers

$$\mathcal{D} \times \mathcal{M}_1 \times \mathcal{M}_2 \times \mathcal{M}_3 \times \mathcal{E} \times \mathcal{C} \times \mathcal{F} \times \mathcal{E} \times \mathcal{L}_1 \times \cdots \times \mathcal{L}_{\lceil \log n \rceil}. \tag{60}$$

Next, we present how to perform the required operations on S with our data structure.

Checking for ϵ-close pairs

To check the existence of an ϵ-close pair, we just read the flag in the root of the radix tree. If the flag is set, there is at most one ϵ-close pair in S, and no such pairs otherwise. This operation takes $O(1)$ time.

Insertion

Given (i, p_i), we perform the insertion with the following steps:

1. Insert this tuple into the hash table.
2. Compute the id, $\mathrm{id}(p_i)$, of the ϵ-box which p_i belongs to. Denote this ϵ-box by $g(\mathrm{id}(p_i))$.
3. Using $\mathrm{id}(p_i)$ as the key, check if this key is already in $\tau'(S)$, if so, insert i into the skip list corresponding to $g(\mathrm{id}(p_i))$; otherwise, first create a uniform superposition of the addresses of all free cells into another register, then create a new tree node in the cell determined by this address register and insert it into the tree. The pointers for the start entry of the skip list is initially set to 0. Insert i into this skip list. Let $\tau'(S, g(\mathrm{id}(p_i))$ denote the leaf node in $\tau'(S)$ corresponding to $g(\mathrm{id}(p_i))$.
4. Increase the local counter $\mathcal{C}$ in $\tau'(S, g(\mathrm{id}(p_i)))$ by 1.
5. Use Procedure 1 to update the external counters $\mathcal{E}$ and flags $\mathcal{F}$ in $\tau'(S, g(\mathrm{id}(p_i)))$ as well as in the leaf nodes corresponding to the neighbor ϵ-boxes of $g(\mathrm{id}(p_i))$.

Note that the first step takes at most $O(\log n)$ time. The second step can be done in $O(d)$ time. In Procedure 1, the number of ϵ-neighbors to check is at most $(2\sqrt{d}+1)^d$.

Procedure 1 Updating nodes for insertion.

input : (i, p_i), The leaf node in $\tau'(S)$ corresponding to the ϵ-box $g(\mathrm{id}(p_i))$, denoted by, $\tau'(S, g(\mathrm{id}(p_i)))$.

if *the local counter* $\mathcal{C} = 1$ *in* $\tau'(S, \mathrm{id}(p_i))$ **then**
  for *all* ϵ*-box* g' *that is a* ϵ*-neighbor (see Definition 44) of* $g(\mathrm{id}(p_i))$ *where the local counter* $\mathcal{C} = 1$ *in* $\tau'(S, g')$ *and the distance between* p_i *and the point in* g' *is at most* ϵ **do**
    Increase the external counter $\mathcal{E}$ of $\tau'(S, g')$ by 1;
    Increase the external counter $\mathcal{E}$ of $\tau'(S, g(\mathrm{id}(p_i)))$ by 1;
    if *the external counter* $\mathcal{E}$ *in* $\tau'(S, g')$ *was increased from 0 to 1* **then**
      Set the flag $\mathcal{F}$ in $\tau'(S, g')$;
      Update the flag $\mathcal{F}$ in the nodes along the path from $\tau'(S, g')$ to the root of $\tau'(S)$;
    end
  end
  if *the external counter* $\mathcal{E} > 1$ *in* $\tau'(S, g(\mathrm{id}(p_i)))$ **then**
    Set the flag $\mathcal{F}$ in $\tau'(S, g(\mathrm{id}(p_i)))$;
    Update the flag $\mathcal{F}$ in the nodes along the path from $\tau'(S, \mathrm{id}(p_i))$ to the root of $\tau'(S)$;
  end
else if *the local counter* $\mathcal{C} = 2$ *in* $\tau'(S, \mathrm{id}(p_i))$ **then**
  Set the flag $\mathcal{F}$ in $\tau'(S, g(\mathrm{id}(p_i)))$;
  Update the flag $\mathcal{F}$ in the nodes along the path from $\tau'(S, g(\mathrm{id}(p_i)))$ to the root of $\tau'(S)$;
  Set the external counter $\mathcal{E} = 0$ in $\tau'(S, \mathrm{id}(p_i))$;
  Let i' be the other index (than i) stored in the skip list corresponding to $g(\mathrm{id}(p_i))$;
  for *all* ϵ*-box* g' *that is a* ϵ*-neighbor of* $g(\mathrm{id}(p_i))$ *where the local counter* $\mathcal{C} = 1$ *in* $\tau'(S, g')$ *and the distance between* $p_{i'}$ *and the point in* g' *is at most* ϵ **do**
    Decrease the external counter of $\tau'(S, g')$ by 1;
    if *the external counter* $\mathcal{E}$ *in* $\tau'(S, g')$ *was decreased from 1 to 0* **then**
      Unset the flag $\mathcal{F}$ in $\tau'(S, g')$;
      Update the flag $\mathcal{F}$ in the nodes along the path from $\tau'(S, g')$ to the root of $\tau'(S)$;
    end
  end
end

To obtain a uniform superposition of the addresses of all free cells, we first create a uniform superposition of all possible addresses to access to the bitmap $|\mathcal{B}\rangle$. We also use an auxiliary register that is initialized to $|0\rangle$. Then, the quantum random access gate defined in Equation (58) is applied on the register holding the uniform superposition of all addresses, the auxiliary register, and the bitmap register, which is effectively a SWAP operation on the second register and the corresponding bit in $|\mathcal{B}\rangle$. The auxiliary register remains $|0\rangle$ if and

only if the address in the first register is free. Since the size of memory space is chosen so that the probability of seeing a free cell is at least 1/3 (and we know exactly this probability based on how many cells have already been used), we can add an extra register and apply a one-qubit rotation to make the amplitude of the second register being $|0\rangle$ exactly 1/2. Hence, using *one* iteration of the oblivious amplitude amplification (which is a generalized version of Grover's search algorithm. See [12] and [34]) with the second register being the indicator, we obtain the uniform superposition of the addresses of all free cells. This cost if $O(\log n)$.

In [5], it was shown that with high probability, insertion into the skip list can be done in $O(d + \log^4(n + L))$ time. Hence, with high probability, the insertion costs $O(d + \log^4(n + L) + d(2\sqrt{d}+1)^d)$ time, where $O(d(2\sqrt{d}+1)^d)$ is the time for checking neighbors. To further reduce the running time, we can just stop the skip list's insertion and deletion procedures after $O(d + \log^4(n + L))$ time, which only causes little error (see Lemma 47).

Deletion

Given (i, p_i), we perform the following steps to delete this tuple from our data structure.

1. Compute the id, $\mathrm{id}(p_i)$, of the ϵ-box which p_i belongs to, and denote this ϵ-box by $g(\mathrm{id}(p_i))$.
2. Using $\mathrm{id}(p_i)$ as the key, we find the leaf node in $\tau'(S)$ that is corresponding to $g(\mathrm{id}(p_i))$.
3. Remove i from the skip list, and decrease the local counter $\mathcal{C}$ in $\tau'(S, g(\mathrm{id}(p_i)))$ by 1.
4. Use Procedure 2 to update the external counters $\mathcal{E}$ and flags $\mathcal{F}$ in $\tau'(S, g(\mathrm{id}(p_i)))$ as well as in leaf nodes corresponding to the neighbor ϵ-boxes of $g(\mathrm{id}(p_i))$.
5. If the local counter $\mathcal{C} = 0$ in this leaf node, remove $\tau'(S, g(\mathrm{id}(p_i)))$ from $\tau'(S)$, and update the bitmap $\mathcal{B}$ in $\tau'(S)$ that keeps track of all free memory cells.
6. Remove (i, p_i) from the hash table.

Note that the first step can be done in $O(d)$ time. The second step can be done in $O(\log n)$ time. Procedure 2 has the same time complexity with Procedure 1. Hence, the cost for the deletion procedure is the same as that for insertion.

Finding a ϵ-close pair

We just read the flag in the root of the radix tree and then go to a leaf whose flag is 1. Check the local counter $\mathcal{C}$ of the node. if it is at least 2, output the first two elements in skip list. Otherwise, we find the ϵ-neighbor of the current node whose $C = 1$ and then output the points in that ϵ-neighbor and the current node.

Uniqueness

The uniqueness of our data structure follows from the analysis of [5, 11, 28]. More specifically, the hash table is always stored in the same way, as each $i \in S$ is stored in the same bucket for the fixed hash function and in each bucket, elements are stored in ascending order of i. The skip list is uniquely stored once the hash functions $h_1, \ldots, h_{\ell_{\max}}$ is determined. The shape of the radix tree is unique for S, but each node can be stored in different locations in memory. We use a uniform superposition of all possible memory organizations (by keeping track of the bitmap for free cells) to keep the quantum state uniquely determined by S.

Correctness

In the following, we argue that our data structure has the desired properties. First, we prove the correctness.

Procedure 2 Updating nodes for deletion.

input : (i, p_i), The leaf node in $\tau'(S)$ corresponding to the ϵ-box $g(\mathrm{id}(p_i))$, denoted by, $\tau'(S, g(\mathrm{id}(p_i)))$.

if *the local counter* $\mathcal{C} = 0$ *in* $\tau'(S, \mathrm{id}(p_i))$ **then**
  Unset the flag $\mathcal{F}$ in $\tau'(S, g(\mathrm{id}(p_i)))$;
  Update the flag $\mathcal{F}$ in the nodes along the path from $\tau'(S, \mathrm{id}(p_i))$ to the root of $\tau'(S)$;
  Set the external counter $\mathcal{E} = 0$ in $\tau'(S, \mathrm{id}(p_i))$;
  for *all* ϵ*-box* g' *that is a* ϵ*-neighbor (see Definition 44) of* $g(\mathrm{id}(p_i))$ *where the local counter* $\mathcal{C} = 1$ *in* $\tau'(S, g')$ *and the distance between* p_i *and the point in* g' *is at most* ϵ **do**
    Decrease the external counter $\mathcal{E}$ of $\tau'(S, g')$ by 1;
    if *the external counter* $\mathcal{E}$ *in* $\tau'(S, g')$ *was decreased from 1 to 0* **then**
      Unset the flag $\mathcal{F}$ in $\tau'(S, g')$;
      Update the flag $\mathcal{F}$ in the nodes along the path from $\tau'(S, g')$ to the root of $\tau'(S)$;
    end
  end
else if *the local counter* $\mathcal{C} = 1$ *in* $\tau'(S, \mathrm{id}(p_i))$ **then**
  Let i' be the only index stored in the skip list corresponding to $g(\mathrm{id}(p_i))$;
  for *all* ϵ*-box* g' *that is a* ϵ*-neighbor of* $g(\mathrm{id}(p_i))$ *where the local counter* $\mathcal{C} = 1$ *in* $\tau'(S, g')$ *and the distance between* $p_{i'}$ *and the point in* g' *is at most* ϵ **do**
    Increase the external counter $\mathcal{E}$ of $\tau'(S, g')$ by 1;
    Increase the external counter $\mathcal{E}$ of $\tau'(S, g(\mathrm{id}(p_i)))$ by 1;
    if *the external counter* $\mathcal{E}$ *in* $\tau'(S, g')$ *was increased from 0 to 1* **then**
      Set the flag $\mathcal{F}$ in $\tau'(S, g')$;
      Update the flag $\mathcal{F}$ in the nodes along the path from $\tau'(S, g')$ to the root of $\tau'(S)$;
    end
  end
  if *the external counter* $\mathcal{E} = 0$ *in* $\tau'(S, g(\mathrm{id}(p_i)))$ **then**
    Unset the flag $\mathcal{F}$ in $\tau'(S, g(\mathrm{id}(p_i)))$;
    Update the flag $\mathcal{F}$ in the nodes along the path from $\tau'(S, \mathrm{id}(p_i))$ to the root of $\tau'(S)$;
  end
end

▶ **Lemma 46.** *The flag bit in the root of* $\tau'(S)$ *is set if and only if there exist distinct* $i, j \in S$ *such that* $|p_i - p_j| \leq \epsilon$.

Proof. We show that after each insertion and deletion, the data structure maintains the following conditions, and then lemma follows.

1. The flag bit of each leaf node of $\tau'(S)$ is set if and only if either its local counter is at least 2, or its external counter is at least 1.
2. The external counter of a leaf node $\tau'(S, g')$ is nonzero if and only if g' contains only one point p, and there exists another p' in another ϵ-box g'' such that $|p - p'| \leq \epsilon$.

It is easy to check that the first condition is maintained for each insertion and deletion. We show the second condition holds during insertions and deletion. For insertions, we consider the first case where a point p is just inserted into the ϵ-grid g', and p is its only point. The first for-loop in Procedure 1 updates other ϵ-boxes that have only one point to maintain the second condition. We consider the second case where g' already contains p' and p is just inserted, then the external counter in g' should be 0, and the second for-loop in Procedure 1 updates other ϵ-boxes that have only one point using the information of p'. This maintains the second condition. For deletions, there are also two cases. First, consider p is the only one point in g' and it is just deleted. We use the first for-loop in Procedure 2 to update the ϵ-boxes that has only one point using the information of p to maintain the second condition. Second, there is another point p' left in g' after deleting p. In this case, we start to check the external counter in g'. We use the second for-loop in Procedure 2 to check other ϵ-boxes that have only one point using the information of p' and update the corresponding external counter to maintain the second condition. ◀

Imperfection of the data structures and error analysis

Our data structure is not perfect. As indicated by Ambainis [5], there are two possibilities that it will fail. First, the hash table might overflow. Second, it might take more that $\lceil \log n \rceil$ time to search in a skip list. To resolve the first problem, we fix the number of entries in each bucket to be $\lceil \log n \rceil$ and treat any overflow as a failure. For the second problem, we stop the subroutine for accessing the skip list after $O(\log n)$ steps, and it causes an error in some cases. The original error analysis can be directly used in our case, as our hash table doesn't change the structure or the hash function, and our skip lists can be viewed as breaking the skip list in [5] into several pieces (one for each nonempty ϵ-box), and each insertion/deletion only involves one of them. Hence, the discussion in [5, Section 6] can be directly adapted to our case:

▶ **Lemma 47** (Adapted from [5]). *Let $|\psi\rangle$ be the final state of our algorithm (with imperfect data structures) and $|\psi'\rangle$ be the final state with the perfect data structure. Then $\| |\psi\rangle - |\psi'\rangle \| \leq O(1/\sqrt{n})$.*

Sketch of proof. There are two places where the data structure may give error: first, the hash table may have overflow, and second, the algorithm cannot find the required element in the skip lists in the desired time. The original proof showed that the probability that any of these errors happens is negligible, and thus the two-norm distance between $|\psi\rangle$ and $|\psi'\rangle$ can be bounded. Here, our data structure combining hash table, skip list, and radix tree, only has errors from hash tables and skip lists. The radix tree which has no error can be viewed as applying additional unitaries on $|\psi\rangle$ and $|\psi'\rangle$, and this does not change the distance between the two states. Since the probability that the errors from hash tables and skip lists happen are negligible by following the same analysis in [5], we can conclude that the two-norm distance between $|\psi\rangle$ and $|\psi'\rangle$ is small. ◀

Time complexity for CP_ϵ

We use the quantum walk framework reviewed in Section 2.5 to solve CP_ϵ. We first build the Johnson graph for CP_ϵ, which is similar to that for ED in Section 2.5. The vertices of the Johnson graph are $S \subseteq [n]$ with $|S| = n^{2/3}$ and S is marked if there exist distinct $i, j \in S$ such that $\Delta(p_i, p_j) \leq \epsilon$. We use $|S, d(S)\rangle$ to represent the quantum state corresponding to S, where $d(S)$ is the data structure of S defined in Section 4.1. Consider the three operations:

- **Steup:** with cost S, preparing the initial state

$$|\pi\rangle = \frac{1}{\sqrt{\binom{n}{n^{2/3}}}} \sum_{S\subseteq[n]:|S|=n^{2/3}} \sqrt{\pi_S}\,|S, d(S)\rangle\,. \tag{61}$$

- **Update:** with cost U, applying the transformation

$$|S, d(S)\rangle\,|0\rangle \mapsto |S, d(S)\rangle \sum_{S'\subseteq[n]:|S\cap S'|=n^{2/3}-1} \sqrt{p_{SS'}}\,|S', d(S')\rangle\,, \tag{62}$$

  where $p_{SS'} = \frac{1}{n^{2/3}(n-n^{2/3})}$.
- **Checking:** with cost C, applying the transformation:

$$|S, d(S)\rangle \mapsto \begin{cases} -\,|S, d(S)\rangle & \text{if } S \in M \\ |S, d(S)\rangle & \text{otherwise,} \end{cases} \tag{63}$$

  where M is the set of marked states.

We have the following result.

▶ **Theorem 48.** *There exists a quantum algorithm that with high probability solves* CP_ϵ *with time complexity* $O(n^{2/3}(d + \log^4(n+L) + d(2\sqrt{d}+1)^d))$.

Proof. The Johnson graph has $\lambda \geq 1/n^{2/3}$ and the Markov chain has spectral gap $\delta \geq 1/n^{2/3}$. For the setup operation, $\mathsf{S} = O(n^{2/3}(d + \log^4(n+L) + d(2\sqrt{d}+1)^d))$, since preparing the uniform superposition for all $|S\rangle$ costs $O(\log n)$ Hadamard gates and we need to do $n^{2/3}$ insertions to prepare the data structure. Each insertion costs $O(d+\log^4(n+L)+d(2\sqrt{d}+1)^d)$. For the update operation, we can implement the quantum walk operator as described in [28]: we use $|S, d(S)\rangle\,|i,j\rangle$ to represent $|S, d(S)\rangle\,|S', d(S')\rangle$ where S' is obtained from S by adding index i and removing index j. Then the diffusion can be implemented by preparing a uniform superposition of all $i \in S$ and a uniform superposition of all $j \notin S$, which takes time $O(\log n)$, and the "SWAP" operation can be implemented by a unitary that maps $|S, d(S)\rangle\,|i,j\rangle$ to $|S', d(S')\rangle\,|j,i\rangle$. In this way, the update operation uses $O(1)$ insertion and deletion to construct $d(S')$ from $d(S)$, and hence $\mathsf{U} = O(d + \log^4(n+L) + d(2\sqrt{d}+1)^d)$. The checking operation can be done in $O(1)$ time with the data structure. Therefore, by Lemma 22, the time complexity is $O(\mathsf{S} + \frac{1}{\sqrt{\lambda}}(\frac{1}{\sqrt{\delta}}\mathsf{U} + \mathsf{C})) = O(n^{2/3}(d + \log^4(n+L) + d(2\sqrt{d}+1)^d))$. ◀

By Lemma 42, we have the following corollary.

▶ **Corollary 49.** *There exists a quantum algorithm that with high probability solves CP with time complexity* $O(n^{2/3} \cdot (d + \log^4(n+L) + d(2\sqrt{d}+1)^d) \cdot (m + \log d))$.

▶ Remark 50. For $d = O(1)$ dimension and $m = O(\log n)$ digits of each coordinate of the points, the running time of the single-shot quantum algorithm is $O(n^{2/3} \cdot \log^5 n)$.

4.3 Multiple-trial quantum walks with simple data structure

In the previous subsection, we provide a quantum algorithm which solves CP_ϵ in time $O(n^{2/3}(d + \log^4 n + d(2\sqrt{d}+1)^d))$, where the logarithmic cost is mainly from the cost of the skip list. In this section we present a quantum algorithm which only requires the radix tree, and thus improve the running time. The caveat is that, with only the radix tree data structure, the insertion would fail if there are more than one ϵ-close pairs. As a result,

we need to keep shrinking the size of the problem using [5, Algorithm 3] until there is a unique solution with high probability, and then run the $\widetilde{O}(n^{2/3})$ quantum algorithm for this unique-solution case.

In the following, we first show how to solve the unique-solution CP_ϵ, and then show the reduction from the multiple-solution case to the unique-solution case.

▶ **Lemma 51.** *There exists a quantum algorithm that with high probability solves the unique-solution* CP_ϵ *with time complexity* $O(n^{2/3}(\log n + d(2\sqrt{d}+1)^d))$.

Data structure for unique-solution

We use the radix tree $\tau(S)$ for S defined in Section 4.1. In the following, we describe the necessary operations on $\tau(S)$.

Checking for ϵ-close pair

To check the existence of an ϵ-close pair, we just read the flag bit in the root of $\tau(S)$, which takes $O(1)$ time.

Insertion

Given (i, p_i), we perform the following steps for insertion. First compute the id, $\mathrm{id}(p_i)$, of the ϵ-box which p_i belongs to. Denote this ϵ-box by $g(\mathrm{id}(p_i))$. Using $\mathrm{id}(p_i)$ as the key, check if this key is already in $\tau(S)$. There are two cases:

- $\mathrm{id}(p_i)$ is already in $\tau(S)$: insert p_i into $\tau(S, g(\mathrm{id}(p_i)))$, increase the local counter in $\tau(S, g(\mathrm{id}(p_i)))$ by 1 and also set the flag. Then update the flag and local counter of the nodes along the path from $\tau(S, g(\mathrm{id}(p_i)))$ to the root.
- $\mathrm{id}(p_i)$ is not in $\tau(S)$: create a new leaf node for $\mathrm{id}(p_i)$ and insert it into $\tau(S)$. Insert p_i into this new leaf node, and increase the local counter in $\tau(S, g(\mathrm{id}(p_i)))$ by 1. Then, check the ϵ-neighbors g' of $\tau(S, g(\mathrm{id}(p_i)))$ that contains only one point p' and set both flags if p_i is ϵ-close to p', and in this case, update the flag bit and local counter on the nodes along the paths from $\tau(S, g(\mathrm{id}(p_i)))$ and g'.

Deletion

Given (i, p_i), we first compute the id, $\mathrm{id}(p_i)$ of the ϵ-box that p_i belongs to, and locate the corresponding leaf node $\tau(S, g(\mathrm{id}(p_i)))$. Decrease the local counter in $\tau(S, g(\mathrm{id}(p_i)))$ by 1 and update the local counter in the nodes along the path from $\tau(S, g(\mathrm{id}(p_i)))$ to the root. Check the number of points stored in $\tau(S, g(\mathrm{id}(p_i)))$. There are two possibilities:

- There are two points in $\tau(S, g(\mathrm{id}(p_i)))$: unset the flag in $\tau(S, g(\mathrm{id}(p_i)))$ and update the flag bit in the nodes along the path to the root and delete p_i from $\tau(S, g(\mathrm{id}(p_i)))$.
- p_i is the only point in $\tau(S, g(\mathrm{id}(p_i)))$: check the ϵ-neighbors g' of $\tau(S, g(\mathrm{id}(p_i)))$ that contains only one point p' and unset both flags if p_i is ϵ-close to p', and in this case, update the flag bit on the nodes along the path from $\tau(S, g(\mathrm{id}(p_i)))$ and g' to the root. Delete p_i from $\tau(S, g(\mathrm{id}(p_i)))$ and delete $\tau(S, g(\mathrm{id}(p_i)))$ from $\tau(S)$.

As in Section 4.2, we use a bitmap register $|\mathcal{B}\rangle$ to keep track of the free cells in $\tau(S)$. For insertion, we maintain a uniform superposition of all possible free cells to insert a new radix tree node. For deletion, we update the bitmap $|\mathcal{B}\rangle$ accordingly. This ensures the uniqueness of the quantum data structure.

The correctness of the data structure is straightforward, and the time complexity is $O(\log n + d(2\sqrt{d}+1)^d)$ for both insertion and deletion. Also, preparing a uniform superposition for all $i \in S$ costs $O(\log n)$ using the local counter in each node. By a similar analysis of Theorem 48, we prove Lemma 51 as follows.

Proof of Lemma 51. The algorithm uses the framework in Lemma 22 with the data structure we just described in this subsection, where $\mathsf{U} = O(\log n + d(2\sqrt{d}+1)^d))$, $\mathsf{C} = O(1)$ and $\mathsf{S} = O(n^{2/3}(\log n + d(2\sqrt{d}+1)^d))$. Therefore, the running time of the algorithm is as claimed. ◀

Next, we show how to reduce multiple-solution CP_ϵ to unique-solution CP_ϵ. A high-level overview of Ambainis's reduction in [5] is the following. We run the algorithm for unique-solution CP_ϵ several times on some random subsets of the given input. If the given subset contains solutions, then with constant probability there exists a subset which contains exactly one solution.

▶ **Definition 52** ([5, 27]). *Let $\mathcal{F}$ be a family of permutations on $f : [n] \to [n]$. $\mathcal{F}$ is ϵ-approximate d-wise independent if for any $x_1, \ldots, x_d \in [n]$ and for all $y_1, \ldots, y_d \in [n]$,*

$$\frac{1-\epsilon}{n\cdot(n-1)\cdot(n-d+1)} \leq \Pr\left[\bigwedge_{i=1}^{n} f_i(x_i) = y_i\right] \leq \frac{1+\epsilon}{n\cdot(n-1)\cdot(n-d+1)}. \tag{64}$$

▶ **Lemma 53** ([5, 27]). *Let n be an even power of a prime number. For any $t \leq n$, $\epsilon > 0$, there exists an ϵ-approximate t-wise independent family $\mathcal{F} = \{\pi_j | j \in [R]\}$ of permutations $\pi_j : [n] \to [n]$ such that:*

- $R = O\left(\left(n^{t^2}\cdot\epsilon^{-t}\right)^{3+o(1)}\right)$;
- *given i, j, $\pi_j(i)$ can be computed in time $O(t\log^2 n)$.*

The multiple-solution algorithm from [5] is as follows:

Algorithm 3 The algorithm for multiple ϵ-close pair.

input : Let (S, ϵ) be the input, and $|S| = n$.

Let $T_1 = S$ and $j = 1$;

while $|T_j| > n^{2/3}$ **do**

  Run the algorithm described in Lemma 51 on T_j, and Measure the final state. If there is a pair with distance less than ϵ, output the pair and stop ;

  Let q_j be an even power of a prime with $|T_j| \leq q_j \leq (1+\frac{1}{8})|T_j|$. Select a random permutation π_j on $[q_j]$ from the $\frac{1}{n}$-approximately $4\log n$-wise independent family of permutations as in Lemma 53 ;

  Let

$$T_{j+1} := \left\{\pi_1^{-1}\cdot\pi_2^{-1}\cdots\pi_j^{-1}(i), \quad i \in \left[\left\lceil\frac{4q_j}{5}\right\rceil\right]\right\}. \tag{65}$$

  $j \leftarrow j+1$;

end

If $|T_j| \leq n^{\frac{2}{3}}$, then run Grover's search algorithm on T_j for a pair with distance at most ϵ ;

We have the main result of this subsection:

▶ **Theorem 54.** *There exists a quantum algorithm that with high probability solves* CP_ϵ *with time complexity* $O(n^{2/3} \cdot (\log n + d(2\sqrt{d}+1)^d) \cdot \log^3 n) = O(n^{2/3} \cdot \log^4 n)$ *for* $d = O(1)$.

Proof. We prove the running time of the algorithm here. For the correctness, one can check [5] for the detail.

By Equation (65), the size of T_{j+1} will be at most

$$\frac{4}{5} \cdot (1 + \frac{1}{8})|T_j| = \frac{9}{10}|T_j|. \tag{66}$$

Therefore, the while-loop takes at most $O(\log n)$ iterations in the worst case. Let $n_j = |T_j|$ be the size of the instance in the j-th iteration. Then, the unique-solution algorithm in Procedure 3 runs in $O(n_j^{2/3} \cdot (\log n_j + d(2\sqrt{d}+1)^d))$-time (Lemma 51), given an $O(1)$-time access to the set T_j. However, in Procedure 3 each element of the random permutation can be computed in time $O(\log^3 n)$ according to Lemma 53 with $t = 4 \log n$, which means the unique-solution algorithm will take $O(\log^3 n)$ time for each query to T_j. Note that we will not actually compute the whole set T_{j+1}, as shown in Procedure 3, which takes too much time. Hence, the running time for the j-th iteration is $O(n_j^{2/3} \cdot (\log n_j + d(2\sqrt{d}+1)^d) \cdot \log^3 n)$. And the total running time for the while-loop is

$$\sum_{j=1}^{O(\log n)} O(n_j^{2/3} \cdot (\log n_j + d(2\sqrt{d}+1)^d) \cdot \log^3 n) \tag{67}$$

$$\leq O(n^{2/3} \cdot (\log n + d(2\sqrt{d}+1)^d) \cdot \log^3 n) \cdot \sum_{j=0}^{O(\log n)} \left(\frac{9}{10}\right)^{2j/3} \tag{68}$$

$$= O(n^{2/3} \cdot (\log n + d(2\sqrt{d}+1)^d) \cdot \log^3 n), \tag{69}$$

where the first inequality follows from $n_j \leq (\frac{9}{10})^{j-1} \cdot n$. Finally, Procedure 3 runs in time $O(n^{2/3} \log n)$. This completes the proof of the running time. ◀

To conclude the quantum algorithms for solving CP in constant dimension, we have the following corollary that is a direct consequence of either Theorem 48 or Theorem 54.

▶ **Corollary 55.** *For any* $d = O(1)$*, there exists a quantum algorithm that, with high probability, solves* $\mathsf{CP}_{n,d}$ *in time* $\widetilde{O}(n^{2/3})$.

4.4 Quantum lower bound for CP in constant dimensions

We can easily get an $\Omega(n^{2/3})$ lower bound for the quantum time complexity of CP in constant dimension by reducing the element distinctness problem (ED) to CP.

▶ **Theorem 56** (Folklore). *The quantum time complexity of* CP *is* $\Omega(n^{2/3})$.

Proof. We reduce ED to one dimensional CP by mapping the point i with value $f(i)$ in ED the point $f(i) \in \mathbb{R}$ in CP. If the closest pair has distance zero, we know there is a collision $f(i) = f(j)$. If the closest pair has distance greater or equal to one, we know there is no collision. Therefore, ED's $\Omega(n^{2/3})$ query lower bound by [1] translates into $\Omega(n^{2/3})$ time lower bound for CP. ◀

5 Bichromatic closest pair in constant dimensions

Classically, bichromatic closest pair problem is harder than the closest pair problem. In constant dimension, the best algorithms for the closest pair problem are "nearly linear", while the algorithm by [3] for bichromatic closest pair problem is "barely subquadratic", running in $O(n^{2-1/\Theta(d)})$-time. In quantum, we found that BCP is still harder than CP in constant dimension. In particular, we cannot adapt the quantum algorithm in previous section for solving BCP because the data structure cannot distinguish the points from two sets efficiently. We can only get a sub-linear time quantum algorithm for BCP using different approach, which is a quadratic speed-up for the classical algorithm.

Nevertheless, we show that we can find an approximate solution for BCP with multiplicative error $1+\xi$ with quantum time complexity $\widetilde{\Theta}(n^{2/3})$. The following theorem is a direct consequence of Theorems 64 and 67.

▶ **Theorem 57.** *For any fixed dimension and error ξ, there is a quantum algorithm which can find an approximate solution for* BCP *with multiplicative error $1+\xi$ in time $\widetilde{O}(n^{2/3})$. Moreover, all quantum algorithms which can find an approximate solution for* BCP *with arbitrary multiplicative error requires time $\Omega(n^{2/3})$.*

Similar to solving CP, we reduce BCP to its decision version of the problem, and then apply quantum algorithms to solve the decision problem. We define the decision problem as BCP_ϵ.

▶ **Definition 58** (BCP_ϵ)**.** *In BCP_ϵ, we are given two sets A, B of n points $\in \mathbb{R}^d$ and a distance measure Δ. The goal is to find a pair of points $a \in A$, $b \in B$ such that $\Delta(a,b) \leq \epsilon$ if it exists and returns* no *if no such pair exists.*

To address the approximate version of BCP, we also define the approximation version of BCP_ϵ as follows:

▶ **Definition 59** ($(1+\xi)$-BCP_ϵ)**.** *In $(1+\xi)$-BCP_ϵ, we are given two sets A, B of n points $\in \mathbb{R}^d$, a distance measure Δ, and ξ. The goal is to do the following*

1. *If there exists a pair of points $a \in A$, $b \in B$ such that $\Delta(a,b) \leq \epsilon$, output the pair (a,b).*
2. *If for all pairs of points $a \in A$, $b \in B$, $\Delta(a,b) > (1+\xi)\epsilon$, returns* no.

Again, we consider $\Delta(a,b) = \|a-b\|$ as the distance measure in this work. We show that BCP reduce to BCP_ϵ in time $O(m + \log d)$, where m is the number of digits of each coordinate and d is the dimension.

▶ **Lemma 60.** *Given an oracle $\mathcal{O}$ for $(1+\xi)$-BCP_ϵ, there exists an algorithm $A^{\mathcal{O}}$ that runs in time and query complexity $O(m + \log d)$ solves the $(1+\xi)$-*BCP*.*

Proof. Let (A, B, δ) be an instance of the $(1+\xi)$-BCP. We first pick an arbitrary pair $a_0 \in A, b_0 \in B$ and computes $\Delta(a_0, b_0)$. Then, we set ϵ to be $\Delta(a_0,b_0)/2$ and run the oracle $\mathcal{O}$ to check whether there exists a distinct pair which distance is less than $\Delta(a_0,b_0)/2$ or not. If there exists such a pair, which we denote as (a_1, b_1), then we set $\epsilon = \Delta(a_1, b_1)$ and call $\mathcal{O}$ to check again. If there is no such a pair, then we set $\epsilon = 3\Delta(a_0,b_0)/4$ and call $\mathcal{O}$. We continuously run this binary search for $m + \log d$ iterations. Finally, the algorithm outputs the bichromatic closest pair. ◀

In the subsections, we present a quantum algorithm for solving $(1+\xi)$-BCP and a quantum algorithm for exact BCP. To complement the algorithmic results, we also give quantum lower bound for BCP.

5.1 Quantum algorithm for $(1+\xi)$-BCP

The quantum algorithm is based on the quantum walk framework on a tensor product of Johnson graphs. To begin with, we define the Johnson graphs J_A and J_B for A and B, respectively. The vertices of J_A, denoted by X_A, is defined as the set $\{S \subseteq A : |S| = n^{2/3}\}$. There is an edge connecting S and S' if and only if $|S \cap S'| = n^{2/3} - 1$. The Markov chain M_A is defined on X_A with $p_{SS'} = \frac{1}{n^{2/3}(n-n^{2/3})}$ when S and S' are connected by an edge. The Johnson graph for J_B for B and its corresponding Markov chain can be defined similarly. The *tensor product* $M_A \otimes M_B$ is defined as the Markov chain based on $X_A \times X_B$ defined as

$$X_A \times X_B := \{(S_A, S_B) : S_A \in X_A, S_B \in X_B\}, \tag{70}$$

with transition probability

$$p_{(S_A,S_B)(S'_A,S'_B)} = p_{S_A S'_A} \cdot p_{S_B S'_B}. \tag{71}$$

A state (S_A, S_B) is marked if there exists a pair $a \in S_A$ and $b \in S_B$ such that $\Delta(a, b) \le \epsilon$.

Now, we examine the properties of $M_A \otimes M_B$. It is easy to see that $\lambda = \frac{\binom{n-1}{n^{2/3}-1}^2}{\binom{n}{n^{2/3}}^2} = \frac{1}{n^{2/3}}$. Let δ_A and δ_B be the spectral gap of M_A and M_B respectively. As a result of [6, Lemma 21.17], $\delta \ge \min\{\delta_A, \delta_B\} = \frac{1}{n^{2/3}}$. By Lemma 22, the cost for solving $(1+\xi)$-BCP$_\epsilon$ is $O(\mathsf{S} + n^{1/3}(n^{1/3}\mathsf{U} + \mathsf{C}))$, where S, U and C are the cost of quantum operations defined in Section 4.2. Before describing the data structure to achieve meaningful S, U, and C, we first introduce a finer discretization scheme. In Section 4, we used a hypergrid consisting of ϵ-boxes. Here, we discretize the space $[0, L]^d$ as a hypergrid consisting of $\frac{\xi\epsilon}{2\sqrt{d}}$-boxes. The following lemma guarantees that distance between a $\frac{\xi\epsilon}{2\sqrt{d}}$-box and its ϵ-neighbor is at most $(1+\xi)\epsilon$.

▶ **Lemma 61.** *Let g and g' be $\frac{\xi\epsilon}{2\sqrt{d}}$-boxes. If g and g' are ϵ-neighbors, then for all $p \in g$ and $p' \in g'$, $\Delta(p, p') \le (1+\xi)\epsilon$.*

Proof. Recall the definition of the id function in Equation (56). $\mathrm{id}(g)$ can be treated as a point, and we can measure the distance between $\mathrm{id}(g)$ and other points. The lemma can be proven via the triangle inequality:

$$\Delta(p, p') \le \Delta(p, \mathrm{id}(g)) + \Delta(\mathrm{id}(g), \mathrm{id}(g')) + \Delta(p', \mathrm{id}(g') \le \frac{\xi\epsilon}{2} + \epsilon + \frac{\xi\epsilon}{2} \le (1+\xi)\epsilon. \tag{72}$$

◀

In our algorithm, we need to search for all ϵ-neighbors that contain the other color to report an ϵ-close pair (with an multiplicative error ξ). It's easy to see that the number of neighbors of a box is bounded in terms of d and ξ:

▷ Claim 62. For each $\frac{\xi\epsilon}{2\sqrt{d}}$-box, the number of ϵ-neighbors is at most $\left(4\sqrt{d}/\xi + 1\right)^d$.

Based on this finer discretization scheme, we use the data structure defined in Section 4.2 but with simple modifications on the radix tree. Instead of using $\mathcal{L}_1, \ldots, \mathcal{L}_{\lceil \log n \rceil}$ as the start entry of the skip list, we use $\lceil \log n \rceil$ pointers for both sets A and B. We also need local counters $\mathcal{C}^A$ and $\mathcal{C}^B$ for both colors. Now, each node in the radix tree has the following registers:

$$\begin{aligned}&\mathcal{D} \times \mathcal{M}_1 \times \mathcal{M}_2 \times \mathcal{M}_3 \times \mathcal{E}^A \times \mathcal{E}^B \times \mathcal{C}^A \times \mathcal{C}^B \times \\ &\mathcal{F} \times \mathcal{L}_1^A \times \cdots \times \mathcal{L}_{\lceil \log n \rceil}^A \times \mathcal{L}_1^B \times \cdots \times \mathcal{L}_{\lceil \log n \rceil}^B.\end{aligned} \tag{73}$$

The points in A (or B, respectively) is organized by the skip list for A (or B, respectively). The insertion and deletion operations are similar to the data structure in Section 4.2, but in the procedure for updating the local and external counters and checking ϵ-neighbors, we need to consider points of the other color. We formally describe the two procedures as follows.

Insertion

Given a point (i, p_i, x), where $x \in \{A, B\}$ denotes the color. We perform the insertion with the following steps:

1. Insert this tuple into the hash table corresponding to x.
2. Compute the id, $\mathrm{id}(p_i)$, of the $\frac{\xi\epsilon}{\sqrt{d}}$-box which p_i belongs to and denote it by $g(\mathrm{id}(p_i))$.
3. Using $\mathrm{id}(p_i)$ as the key, check if this key is already in $\tau'(S)$, if so, insert i into the skip list for color x corresponding to $g(\mathrm{id}(p_i))$; otherwise, first create a uniform superposition of the addresses of all free cells into another register, then create a new tree node in the cell determined by this address register and insert it into the tree. The pointer for the start entry of the skip list is initially set to 0. Insert i into this skip list. Let $\tau'(S, g(\mathrm{id}(p_i))$ denote the leaf node in $\tau'(S)$ corresponding to $g(\mathrm{id}(p_i))$.
4. Increase the local counter $\mathcal{C}^x$ in $\tau'(S, g(\mathrm{id}(p_i)))$ by 1.
5. Use Procedure 4 to update the external counters $\mathcal{E}^x, \mathcal{E}^{\bar{x}}$ (here $\bar{x}$ denotes the other color than x) and flags $\mathcal{F}$ in $\tau'(S, g(\mathrm{id}(p_i)))$, the leaf nodes which are corresponding to the ϵ-neighbors of $g(\mathrm{id}(p_i))$, and their parent nodes.

Note that the first step takes at most $O(\log n)$ time. The second step can be done in $O(d)$ time. In Procedure 4, the number of ϵ-neighbors to check is at most $(\frac{4\sqrt{d}}{\xi} + 1)^d$ by Claim 62.

Deletion

Given (i, p_i, x), we perform the following steps to delete this tuple from our data structure.

1. Compute the id, $\mathrm{id}(p_i)$, of the $\frac{\epsilon\xi}{\sqrt{d}}$-box which p_i belongs to and denote it by $g(\mathrm{id}(p_i))$.
2. Using $\mathrm{id}(p_i)$ as the key, we find the leaf node in $\tau'(S)$ that is corresponding to $g(\mathrm{id}(p_i))$.
3. Remove i from the skip list for color x, and decrease the local counter $\mathcal{C}^x$ in $\tau'(S, g(\mathrm{id}(p_i)))$ by 1.
4. Use Procedure 2 to update the external counters $\mathcal{E}^x$ and $\mathcal{E}^{\bar{x}}$ (here $\bar{x}$ denote the other color than x) and flags $\mathcal{F}$ in $\tau'(S, g(\mathrm{id}(p_i)))$ as well as in leaf nodes corresponding to the ϵ-neighbors of $g(\mathrm{id}(p_i))$.
5. If both local counters $\mathcal{C}^x, \mathcal{C}^{\bar{x}}$ in this leaf node are 0, remove $\tau'(S, g(\mathrm{id}(p_i)))$ from $\tau'(S)$, and update the bitmap $\mathcal{B}$ in $\tau'(S)$ that keeps track of all free memory cells.
6. Remove (i, p_i, x) from the hash table.

Note that the first step can be done in $O(d)$ time. The second step can be done in $O(\log n)$ time. Procedure 5 has the same time complexity with Procedure 4. Hence, the cost for the deletion procedure is the same with that for insertion.

Checking for $(1 + \xi)\epsilon$-close pairs

To check the existence of an $(1 + \xi)\epsilon$-close pair, we just read the flag in the root of the radix tree. If the flag is set, there is at most one ϵ-close pair in S, and no such pairs otherwise. This operation takes $O(1)$ time.

Procedure 4 Updating nodes for insertion for the bichromatic case.

input : (i, p_i, x), the leaf node in $\tau'(S)$ corresponding to $g(\mathrm{id}(p_i))$, denoted by $\tau'(S, g(\mathrm{id}(p_i)))$.

Let $\bar{x} \in \{A, B\}$ and $\bar{x} \neq x$;
if $\mathcal{C}^x = 1$ *in* $\tau'(S, \mathrm{id}(p_i))$ *and* $\mathcal{C}^{\bar{x}} = 0$ **then**
  for *all* ϵ*-neighbor* g' *(see Definition 44) of* $g(\mathrm{id}(p_i))$ *where* $\mathcal{C}^{\bar{x}} \geq 1$ *in* $\tau'(S, g')$ **do**
    Increase $\mathcal{E}^x$ of $\tau'(S, g')$ by 1;
    Increase $\mathcal{E}^{\bar{x}}$ of $\tau'(S, g(\mathrm{id}(p_i)))$ by 1;
    if $\mathcal{E}^x$ *in* $\tau'(S, g')$ *was increased from 0 to 1* **then**
      Set the flag $\mathcal{F}$ in $\tau'(S, g')$;
      Update the flags $\mathcal{F}$ in the nodes along the path from $\tau'(S, g')$ to the root of $\tau'(S)$;
    end
  end
  if $\mathcal{E}^{\bar{x}} \geq 1$ *in* $\tau'(S, g(\mathrm{id}(p_i)))$ **then**
    Set the flag $\mathcal{F}$ in $\tau'(S, g(\mathrm{id}(p_i)))$;
    Update the flags $\mathcal{F}$ in the nodes along the path from $\tau'(S, \mathrm{id}(p_i))$ to the root of $\tau'(S)$;
  end
else if $\mathcal{C}^x = 1$ *and* $\mathcal{C}^{\bar{x}} \geq 1$ *in* $\tau'(S, g(\mathrm{id}(p_i)))$ **then**
  Set the flag $\mathcal{F}$ in $\tau'(S, g(\mathrm{id}(p_i)))$;
  Update the flags $\mathcal{F}$ in the nodes along the path from $\tau'(S, g(\mathrm{id}(p_i)))$ to the root of $\tau'(S)$;
  Set $\mathcal{E}^{\bar{x}} = 0$ in $\tau'(S, \mathrm{id}(p_i))$;
  for *all* g' *that is an* ϵ*-neighbor of* $g(\mathrm{id}(p_i))$ *where the the local counter* $\mathcal{C}^{\bar{x}} \geq 1$ *in* $\tau'(S, g')$ **do**
    Decrease $\mathcal{E}^x$ of $\tau'(S, g')$ by 1;
    if $\mathcal{E}^x$ *in* $\tau'(S, g')$ *was decreased from 1 to 0* **then**
      Unset the flag $\mathcal{F}$ in $\tau'(S, g')$;
      Update the flags $\mathcal{F}$ in the nodes along the path from $\tau'(S, g')$ to the root of $\tau'(S)$;
    end
  end
end

Finding a $(1+\xi)\epsilon$-close pair

We just read the flag in the root of the radix tree and then go to a leaf which flag is 1. Check the local counters of the node. If both local counters are at least 1, output the first elements in skip lists for A and the first element in the skip list for B. Otherwise, check the external counters. Suppose $\mathcal{E}^A$ is non-zero. Then we find the ϵ-neighbor of the current node whose $\mathcal{C}^B > 0$ and output the first point in the skip list of A of the current node and the first element in the skip list of B of the ϵ-neighbor.

We have the following result.

▶ **Theorem 63.** *For any fixed dimension and fixed* ξ*, there exists a quantum algorithm that, with high probability, can solve* $(1+\xi)$-BCP_ϵ *in time* $O(n^{2/3}(d + \log^4(n+L) + d(\frac{4\sqrt{d}}{\xi} + 1)^d))$.

Procedure 5 Updating nodes for deletion for the bichromatic case.

input : (i, p_i, x) from A, the leaf node in $\tau'(S)$ corresponding to $g(\mathrm{id}(p_i))$, which we denote as $\tau'(S, g(\mathrm{id}(p_i)))$.

Let $\bar{x} \in \{A, B\}$ and $\bar{x} \neq x$;
if $\mathcal{C}^x$ *and* $\mathcal{C}^{\bar{x}}$ *in* $\tau'(S, \mathrm{id}(p_i)) = 0$ **then**
  Unset the flag $\mathcal{F}$ in $\tau'(S, g(\mathrm{id}(p_i)))$;
  Update the flags $\mathcal{F}$ in the nodes along the path from $\tau'(S, \mathrm{id}(p_i))$ to the root of $\tau'(S)$;
  Set $\mathcal{E}^x = 0$ and $\mathcal{E}^{\bar{x}} = 0$ in $\tau'(S, \mathrm{id}(p_i))$;
  for *all* g' *that is an* ϵ*-neighbor (see Definition 44) of* $g(\mathrm{id}(p_i))$ *where the local counter* $\mathcal{C}^{\bar{x}} \geq 1$ *in* $\tau'(S, g')$ **do**
    Decrease $\mathcal{E}^x$ of $\tau'(S, g')$ by 1;
    if $\mathcal{E}^x$ *in* $\tau'(S, g')$ *was decreased from 1 to 0* **then**
      Unset the flag $\mathcal{F}$ in $\tau'(S, g')$;
      Update the flags $\mathcal{F}$ in the nodes along the path from $\tau'(S, g')$ to the root of $\tau'(S)$;
    end
  end
else if $\mathcal{C}^x = 0$ *and* $\mathcal{C}^{\bar{x}} \geq 1$ **then**
  for *all* g' *that is an* ϵ*-neighbor of* $g(\mathrm{id}(p_i))$ *where the local counter* $\mathcal{C}^x \geq 1$ *in* $\tau'(S, g')$ **do**
    Increase $\mathcal{E}^{\bar{x}}$ of $\tau'(S, g')$ by 1;
    Increase $\mathcal{E}^x$ of $\tau'(S, g(\mathrm{id}(p_i)))$ by 1;
    if $\mathcal{E}^{\bar{x}}$ *in* $\tau'(S, g')$ *was increased from 0 to 1* **then**
      Set the flag $\mathcal{F}$ in $\tau'(S, g')$;
      Update the flags $\mathcal{F}$ in the nodes along the path from $\tau'(S, g')$ to the root of $\tau'(S)$;
    end
  end
  if $\mathcal{E}^x = 0$ *in* $\tau'(S, g(\mathrm{id}(p_i)))$ **then**
    Unset the flag $\mathcal{F}$ in $\tau'(S, g(\mathrm{id}(p_i)))$;
    Update the flags $\mathcal{F}$ in the nodes along the path from $\tau'(S, \mathrm{id}(p_i))$ to the root of $\tau'(S)$;
  end
end

Proof. The proof closely follows the analysis for Theorem 48, and the correctness of the data structure and the time complexity of its operations follow from the discussion in Section 4.2. Note that our algorithm will output a pair which belong to the same $\frac{\xi\epsilon}{2\sqrt{d}}$-box or two of them that are ϵ-neighbors. Based on Lemma 61, two points which corresponding hyercubes are ϵ-neighbors have distance at most $(1+\xi)\epsilon$. Therefore, our algorithm could output a pair of points which distance is at most $(1+\xi)\epsilon$. Another difference is that here we need to search at most $(4\sqrt{d}/\xi + 1)^d$ neighbors during insertions and deletions. As a result, $\mathsf{U} = O(d + \log^4(n+L) + d(4\sqrt{d}/\xi+1)^d)$, and $\mathsf{S} = O(n^{2/3}(d + \log^4(n+L) + d(4\sqrt{d}/\xi+1)^d)$. Again, $\mathsf{C} = O(1)$, $\delta \geq 1/n^{2/3}$, and $\lambda \geq 1/n^{2/3}$. Therefore, by Lemma 22, the total cost is $O(\mathsf{S} + \frac{1}{\sqrt{\lambda}}(\frac{1}{\sqrt{\delta}}\mathsf{U} + \mathsf{C})) = O(n^{2/3}(d + \log^4(n+L) + (4\sqrt{d}/\xi+1)^d))$. ◀

By Lemma 60 and the above Theorem 63, we have the following theorem:

▶ **Theorem 64.** *For an fixed dimension and fixed ξ, there exists a quantum algorithm that, with high probability, can solve $(1+\xi)$-BCP in time $\widetilde{O}(n^{2/3})$.*

5.2 Quantum algorithm for solving BCP exactly

In this subsection, we present a quantum algorithm for solving BCP exactly. The main idea of this algorithm is to partition A into smaller subsets. Then we build data structures which support nearest-neighbor search on all subsets in superposition. We use the quantum minimum finding algorithm to find the smallest distances from B to each subset, among which we use the quantum minimum finding algorithm again to find the smallest distance.

Unlike the data structure for solving CP, the data structure for BCP does not have to be uniquely represented, as no insertion and deletion are performed in this algorithm. The data structure can have expected running time instead of the worst-case running time. The total worst-case running time can be bounded by standard techniques. The nearest-neighbor search data structure we use is from [19], and is reformulated in the following lemma.

▶ **Lemma 65** ([19]). *For any fixed dimension, there exists a data structure for n points in $\mathbb{R}^d$ that can be built in expected time complexity $O(n^{\lceil d/2\rceil+\delta})$ for arbitrarily small δ and the nearest-neighbor search can be performed in worst-case time complexity $O(\log n)$.*

This data structure is based on the Voronoi diagram and its triangulation in higher dimensions. Using this data structure, we have a quantum algorithm for solving BCP exactly, which yields the following theorem.

▶ **Theorem 66.** *There exists a quantum algorithm that, with high probability, solves BCP for dimension d with time complexity $\widetilde{O}\left(n^{1-\frac{1}{2d}+\delta}\right)$ for arbitrarily small δ.*

Proof. We first partition A into $\lceil n/r\rceil$ subsets $S_1,\ldots,S_{\lceil n/r\rceil}$, where $|S_i|=r$ for $i\in\left[\lceil n/r\rceil\right]$. (The value of r will be determined later). For all $i\in\left[\lceil n/r\rceil\right]$, we can find a closest pair between S_i and B as follows. First, a data structure as in Lemma 65 for S_i is built in expected time $O\left(r^{\lceil d/2\rceil+\delta}\right)$, which supports nearest-neighbor search in time $O(\log n)$. Then, we use the quantum minimum finding subroutine (Theorem 9) which uses the distance reported by the nearest-neighbor search as the oracle. The closest pair between S_i and B can be found in time complexity $\widetilde{O}(\sqrt{n})$. Note that the time complexity for building the data structure is not bounded for the worst case. However, using Markov's inequality, we know that with high probability, say, at least $9/10$, the time complexity is bounded by $O\left(r^{\lceil d/2\rceil+\delta}\right)$. Hence, fixing a constant $c\geq 10$, and stop the data structure construction after $c\cdot r^{\lceil d/2\rceil+\delta}$ steps. With at most $1/10$ probability, the construction will fail and this event can be detected by checking the solution returned by the quantum minimum finding subroutine. We run $O(\log n)$ instances of above procedure in parallel and use take the quantum minimum of all the $O(\log n)$ results. The probability that all these instances fail is at most $(1/10)^{O(\log n)}=O(1/n)$. We refer to the above procedure as the "inner search", and its time complexity is $O\left(r^{\lceil d/2\rceil+\delta}+\sqrt{n}\right)$.

Next, we use the distance of the output of the inner search as the oracle and perform another quantum minimum finding subroutine for $i\in\left[\lceil n/r\rceil\right]$. We refer to this procedure as the "outer search". The probability that the closest pair between A and B lies in S_i and B is r/n. As a result, the number of the oracle queries for the quantum minimum finding subroutine is $\widetilde{O}(\sqrt{n/r})$. The time complexity for each query is $O\left(r^{\lceil d/2\rceil+\delta}+\sqrt{n}\right)$. Therefore,

the total time complexity is $\widetilde{O}\big((r^{\lceil d/2\rceil+\delta}+\sqrt{n})\cdot\sqrt{n/r}\big)$. A simple calculation shows that this achieves the minimum (ignoring the δ term in the exponent) when $r = n^{1/d}/(d-1)^{2/d}$, which yields the total time complexity

$$\widetilde{O}\left(n^{1-\frac{1}{2d}+\delta}\right). \tag{74}$$

The failure probability for each query is at most $O(1/n)$. Therefore, the total failure probability is at most $O(\sqrt{n/r}/n) = O(n^{-(1/2-1/2d)})$ for $d > 1$, which can be smaller than any constant. ◀

5.3 Quantum lower bound for BCP in constant dimensions

Now, we give a lower bound for $(1+\xi)$-BCP, which trivially holds for BCP.

▶ **Theorem 67.** *The quantum query complexity for solving* BCP *is* $\Omega(n^{2/3})$. *Furthermore, the quantum query complexity for solving* $(1+\xi)$-BCP *with an arbitrary* ξ *is also* $\Omega(n^{2/3})$.

Proof. Recall that we have shown in Section 4.4 that ED reduces to CP by viewing ED as one-dimensional CP with the minimum distance 0. It is not hard to see that ED also reduces to approximate CP with multiplicative error $1+\xi$ since 0 times $1+\xi$ is still 0. For simplicity, we denote approximate CP with multiplicative error $1+\xi$ as $(1+\xi)$-CP. Given a set S as a $(1+\xi)$-CP instance, we choose $A, B \subset S$ uniformly at random such that $A = S \setminus B$ and $|A| = |B|$. Then, with $1/2$ probability, a closest pair in S has one point in A and another in B. Therefore, if (a, b) be a valid solution for $(1+\xi)$-BCP on (A, B), (a, b) is also a a valid solution for $(1+\xi)$-CP on S with probability $1/2$.

It is obvious that following the same proof, CP reduces to BCP. Hence, the quantum query complexity for BCP and $(1+\xi)$-BCP are both $\Omega(n^{2/3})$. This completes the proof. ◀

6 Orthogonal vectors in constant dimensions

▶ **Theorem 68.** *The time complexity of* $\mathsf{OV}_{n,d}$ *(Definition 10) in quantum query model is* $\Theta(\sqrt{n})$ *when the dimension* d *is constant .*

Proof. We show lower and upper bounds for $\mathsf{OV}_{n,d}$:

Lower bound

We reduce the search problem to an instance of 2-dimensional OV. Let all vectors in A be $(0, 1)$. We map an element of the search instance with value 0 as a vector in B with value $(0, 1)$ in $\mathsf{OV}_{n,2}$, and 1 as $(1, 0)$. An orthogonal pair must contain the vector in B with value $(1, 0)$ in this construction. Therefore, if we find an orthogonal pair, we find the corresponding marked (value 1) element in the search instance. The $\Omega(\sqrt{n})$ lower bound of Grover's search algorithm gives an $\Omega(\sqrt{n})$ lower bound to $\mathsf{OV}_{n,d}$.

Upper bound

The vectors only have 2^d possible values, $\{0, 1\}^d$, in the d-dimensional OV. For a particular value $v \in \{0, 1\}^d$, we can use Grover search to check whether there exist vector $a \in A$ such that $a = v$ in time $O(\sqrt{n})$, and similarly for vectors in B. Therefore we can, for all $v \in \{0, 1\}^d$, check whether there exist $a \in A$ such that $a = v$ and $b \in B$ such that $b = v$ in $O(2^{d+1}\sqrt{n})$ time, recording the results as two 2^d bit strings S_A and S_B. Then we check all

2^{2d} pairs of values (v, w) whether $\langle v, w\rangle = 0$, $S_A(v) = 1$, and $S_B(w) = 1$. When we found such a pair (v, w), we use Grover's search algorithm again to output a corresponding pair of vectors. The total running time is $O(2^{d+1}\sqrt{n} + 2^{2d} + 2\sqrt{n}) = \widetilde{O}(\sqrt{n})$. ◀

References

1 Scott Aaronson and Yaoyun Shi. Quantum Lower Bounds for the Collision and the Element Distinctness Problems. *J. ACM*, 51(4):595–605, July 2004.

2 Amir Abboud, Ryan Williams, and Huacheng Yu. More Applications of the Polynomial Method to Algorithm Design. In *Proceedings of the Twenty-sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '15, pages 218–230, 2015.

3 Pankaj K. Agarwal, Herbert Edelsbrunner, Otfried Schwarzkopf, and Emo Welzl. Euclidean Minimum Spanning Trees and Bichromatic Closest Pairs. *Discrete & Computational Geometry*, 6(3):407–422, September 1991.

4 A. Ambainis. Quantum Search Algorithms. *SIGACT News*, 35(2):22–35, June 2004. doi:10.1145/992287.992296.

5 Andris Ambainis. Quantum Walk Algorithm for Element Distinctness. *SIAM Journal on Computing*, 37(1):210–239, 2007.

6 Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009.

7 Arturs Backurs, Piotr Indyk, and Ludwig Schmidt. On the Fine-Grained Complexity of Empirical Risk Minimization: Kernel Methods and Neural Networks. In *Advances in Neural Information Processing Systems*, pages 4308–4318, 2017.

8 Marshall Ball, Alon Rosen, Manuel Sabin, and Prashant Nalini Vasudevan. Proofs of Useful Work. *IACR Cryptology ePrint Archive*, 2017:203, 2017.

9 C. Bennett, E. Bernstein, G. Brassard, and U. Vazirani. Strengths and Weaknesses of Quantum Computing. *SIAM Journal on Computing*, 26(5):1510–1523, 1997.

10 Jon Louis Bentley and Michael Ian Shamos. Divide-and-Conquer in Multidimensional Space. In *Proceedings of the eighth annual ACM symposium on Theory of computing*, pages 220–230. ACM, 1976.

11 Daniel J. Bernstein, Stacey Jeffery, Tanja Lange, and Alexander Meurer. Quantum Algorithms for the Subset-Sum Problem. In *Post-Quantum Cryptography*, pages 16–33. Springer Berlin Heidelberg, 2013. doi:10.1007/978-3-642-38616-9_2.

12 Dominic W. Berry, Andrew M. Childs, Richard Cleve, Robin Kothari, and Rolando D. Somma. Exponential Improvement In Precision for Simulating Sparse Hamiltonians. *Forum of Mathematics, Sigma*, 5, 2017. doi:10.1017/fms.2017.2.

13 Sergei N Bespamyatnikh. An Optimal Algorithm for Closest-Pair Maintenance. *Discrete & Computational Geometry*, 19(2):175–195, 1998.

14 Harry Buhrman, Christoph Durr, Mark Heiligman, Peter Hoyer, Frédéric Magniez, Miklos Santha, and Ronald De Wolf. Quantum Algorithms for Element Distinctness. In *Proceedings 16th Annual IEEE Conference on Computational Complexity*, pages 131–137. IEEE, 2001.

15 Harry Buhrman, Subhasree Patro, and Florian Speelman. The Quantum Strong Exponential-Time Hypothesis. *arXiv preprint*, 2019. arXiv:1911.05686.

16 Timothy M Chan and Ryan Williams. Deterministic APSP, orthogonal vectors, and more: Quickly derandomizing Razborov-Smolensky. In *Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete algorithms*, pages 1246–1255. Society for Industrial and Applied Mathematics, 2016.

17 Lijie Chen. On the Hardness of Approximate and Exact (Bichromatic) Maximum Inner Product. *arXiv preprint*, 2018. arXiv:1802.02325.

18 Lijie Chen and Ryan Williams. An equivalence class for orthogonal vectors. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 21–40. SIAM, 2019.

19 Kenneth L Clarkson. A Randomized Algorithm for Closest-Point Queries. *SIAM Journal on Computing*, 17(4):830–847, 1988.

20 Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT press, 2009.

21 Evgeny Dantsin, Vladik Kreinovich, and Alexander Wolpert. On Quantum Versions of Record-breaking Algorithms for SAT. *SIGACT News*, 36(4):103–108, December 2005. doi:10.1145/1107523.1107524.

22 R. David, K. S., and B. Laekhanukit. On the Complexity of Closest Pair via Polar-Pair of Point-Sets. *SIAM Journal on Discrete Mathematics*, 33(1):509–527, 2019.

23 Christoph Durr and Peter Hoyer. A quantum algorithm for finding the minimum. *arXiv preprint*, 1996. arXiv:quant-ph/9607014.

24 Lov K. Grover. A Fast Quantum Mechanical Algorithm for Database Search. In *Proceedings of the Twenty-eighth ACM Symposium on Theory of Computing - STOC '96*. ACM Press, 1996. doi:10.1145/237814.237866.

25 Timon Hertli. *Improved Exponential Algorithms for SAT and ClSP*. PhD thesis, ETH Zurich, 2015.

26 Russell Impagliazzo and Ramamohan Paturi. On the Complexity of k-SAT. *J. Comput. Syst. Sci.*, 62(2):367–375, March 2001. doi:10.1006/jcss.2000.1727.

27 Toshiya Itoh, Tatsuya Nagatani, and Jun Tarui. Explicit Construction for k-Wise Nearly Random Permutations by Iterated Feistel Transform. Workshop on Randomness and Computation, 2005.

28 Stacey Jeffery. *Frameworks for Quantum Algorithms*. PhD thesis, University of Waterloo, 2014.

29 CS Karthik and Pasin Manurangsi. On Closest Pair in Euclidean Metric: Monochromatic is as Hard as Bichromatic. *10th Innovations in Theoretical Computer Science*, 2019.

30 S. Khuller and Y. Matias. A Simple Randomized Sieve Algorithm for the Closest-Pair Problem. *Information and Computation*, 118(1):34–37, 1995.

31 Victor Klee. On the Complexity of d-Dimensional Voronoi Diagrams. *Archiv der Mathematik*, 34(1):75–80, 1980. doi:10.1007/bf01224932.

32 Jon Kleinberg and Eva Tardos. *Algorithm Design*. Pearson Education India, 2006.

33 Frédéric Magniez, Ashwin Nayak, Jérémie Roland, and Miklos Santha. Search via Quantum Walk. *SIAM Journal on Computing*, 40(1):142–164, 2011.

34 Chris Marriott and John Watrous. Quantum Arthur–Merlin games. *Computational Complexity*, 14(2):122–152, 2005. doi:10.1007/s00037-005-0194-x.

35 Abdullah Mueen, Eamonn Keogh, Qiang Zhu, Sydney Cash, and Brandon Westover. Exact Discovery of Time Series Motifs. In *Proceedings of the 2009 SIAM international conference on data mining*, pages 473–484. SIAM, 2009.

36 Alexandros Nanopoulos, Yannis Theodoridis, and Yannis Manolopoulos. C2P: Clustering based on Closest Pairs. In *VLDB*, 2001.

37 Michael A Nielsen and Isaac Chuang. Quantum Computation and Quantum Information, 2002.

38 Ramamohan Paturi and Pavel Pudlák. On the Complexity of Circuit Satisfiability. In *Proceedings of the forty-second ACM symposium on Theory of computing*, pages 241–250. ACM, 2010.

39 Ramamohan Paturi, Pavel Pudlák, Michael E Saks, and Francis Zane. An Improved Exponential-Time Algorithm for k-SAT. *Journal of the ACM (JACM)*, 52(3):337–364, 2005.

40 Michael O Rabin. Probabilistic Algorithms Algorithms and Complexity: New Directions and Recent Results, 1976.

41 Kunihiko Sadakane, Norito Sugawara, and Takeshi Tokuyama. Quantum Algorithms for Intersection and Proximity Problems. In Peter Eades and Tadao Takaoka, editors, *Algorithms and Computation*, pages 148–159, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

42 Dominik Scheder and John P Steinberger. PPSZ for General k-SAT-Making Hertli's Analysis Simpler and 3-SAT Faster. In *32nd Computational Complexity Conference (CCC 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

43 T Schoning. A Probabilistic Algorithm for k-SAT and Constraint Satisfaction Problems. In *40th Annual Symposium on Foundations of Computer Science (Cat. No. 99CB37039)*, pages 410–414. IEEE, 1999.

44 M. I. Shamos and D. Hoey. Closest-Point Problems. In *16th Annual Symposium on Foundations of Computer Science (sfcs 1975)*, pages 151–162, 1975.

45 Mario Szegedy. Quantum Speed-Up of Markov Chain Based Algorithms. In *45th Annual IEEE symposium on foundations of computer science*, pages 32–41. IEEE, 2004.

46 Virginia Vassilevska Williams. Hardness of Easy Problems: Basing Hardness on Popular Conjectures such as the Strong Exponential Time Hypothesis (invited talk). In *10th International Symposium on Parameterized and Exact Computation (IPEC 2015)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2015.

47 Nilton Volpato and Arnaldo Moura. A fast quantum algorithm for the closest bichromatic pair problem, 2010.

48 Ryan Williams. Pairwise comparison of bit vectors. Theoretical Computer Science Stack Exchange. URL: `https://cstheory.stackexchange.com/q/37369`.

49 Ryan Williams. A New Algorithm for Optimal 2-Constraint Satisfaction and Its Implications. *Theoretical Computer Science*, 348(2-3):357–365, 2005.

50 Ryan Williams and Huacheng Yu. Finding Orthogonal Vectors In Discrete Structures. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*, pages 1867–1877. SIAM, 2014.

51 Raymond Chi-Wing Wong, Yufei Tao, Ada Wai-Chee Fu, and Xiaokui Xiao. On Efficient Spatial Matching. In *Proceedings of the 33rd International Conference on Very Large Data Bases*, VLDB '07, pages 579–590, 2007.

52 A. C. . Yao. Lower Bounds for Algebraic Computation Trees with Integer Inputs. In *30th Annual Symposium on Foundations of Computer Science*, pages 308–313, 1989.