Quantum Algorithms for Hopcroft's Problem

Vladimirs Andrejevs
Centre for Quantum Computer Science, Faculty of Computing, University of Latvia, Riga, Latvia

Aleksandrs Belovs
Centre for Quantum Computer Science, Faculty of Computing, University of Latvia, Riga, Latvia

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

Abstract

In this work we study quantum algorithms for Hopcroft's problem which is a fundamental problem in computational geometry. Given n points and n lines in the plane, the task is to determine whether there is a point-line incidence. The classical complexity of this problem is well-studied, with the best known algorithm running in $O(n^{4/3})$ time, with matching lower bounds in some restricted settings. Our results are two different quantum algorithms with time complexity $\widetilde{O}(n^{5/6})$. The first algorithm is based on partition trees and the quantum backtracking algorithm. The second algorithm uses a quantum walk together with a history-independent dynamic data structure for storing line arrangement which supports efficient point location queries. In the setting where the number of points and lines differ, the quantum walk-based algorithm is asymptotically faster. The quantum speedups for the aforementioned data structures may be useful for other geometric problems.

2012 ACM Subject Classification Theory of computation → Quantum computation theory; Theory of computation → Computational geometry; Theory of computation → Data structures design and analysis

Keywords and phrases Quantum algorithms, Quantum walks, Computational Geometry

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.9

Related Version *Preprint*: `https://arxiv.org/abs/2405.01160`

Funding *Aleksandrs Belovs*: Supported by QOPT (QuantERA ERA-NET Cofund).
Jevgēnijs Vihrovs: Supported by Latvian Quantum Initiative under European Union Recovery and Resilience Facility project no. 2.3.1.1.i.0/1/22/I/CFLA/001 and QOPT (QuantERA ERA-NET Cofund).

1 Introduction

In this work we investigate the quantum complexity of Hopcroft's problem, a classic problem in computational geometry. Given n lines and n points in the plane, it asks to determine whether some point lies on some line. In a line of research spanning roughly 40 years culminating with a recent paper by Chan and Zheng [19], the classical complexity has settled on $O(n^{4/3})$ time, with matching lower bounds in some models of computation [27]. Along with its natural setting, the problem also captures the essence of a class of other geometric problems with complexity $\widetilde{O}(n^{4/3})$ [26].

There are several reasons why we find Hopcroft's problem interesting in the quantum setting. Firstly, classical algorithms for this problem typically use data structures supporting some fundamental geometric query operations. For example, Hopcroft's problem can be reduced to the *simplex range searching*, in which the data structure stores the given points and each query asks whether a given region contains any of the points [4]. Another approach is to store the given lines instead and answer *point location queries*, that is, for a given point, determine which region of the line configuration it belongs to [40, 25]. Thus, Hopcroft's problem gives a good playground for improving and comparing the complexity

of ubiquitous geometric data structures quantumly. We are also interested in finding new *history-independent* data structures that can be used in quantum walk algorithms, following the ideas started with Ambainis' element distinctness algorithm [7] (see also [1, 15]).

Secondly, Hopcroft's problem is closely related to a large group of geometric tasks that can be solved in the same time $\widetilde{O}(n^{4/3})$. A speedup for Hopcroft's problem may automatically give an improvement for some of those. Erickson studied the class of such problems [26], some examples include detecting/counting intersections in a set of segments and detecting/counting points in a given set of regions. The problem can be also reduced to various other geometric problems, giving fine-grained lower bounds. For example, Hopcroft's problem in d dimensions (replacing lines with hyperplanes) can be reduced to halfspace range checking in $d+1$ dimensions for $d \geq 4$ (are all given points above all given hyperplanes?) and others [26].

In fact, Hopcroft's problem in d dimensions can also be equivalently formulated as follows: given two sets of vectors $\mathcal{A}, \mathcal{B} \in \mathbb{R}^{d+1}$, determine whether there are $a \in \mathcal{A}$, $b \in \mathcal{B}$ such that $\langle a, b \rangle = 0$ [46]. The famous Orthogonal Vectors problem (OV) in fine-grained complexity is a special case of Hopcroft's when $\mathcal{A}, \mathcal{B} = \{0,1\}^d$ [44, 3]. The complexities of these problems differ; if $|A| = |B| = n$, then, classically, the complexity of OV is $\Theta(n)$ in $O(1)$ dimensions [45] and $\Theta(n^{2-o(1)})$ in polylog n dimensions under SETH [2, 18]. In contrast, the complexity of Hopcroft's problem in d dimensions is $O(n^{2d/(d+1)})$ [19]. Quantumly, the complexity of OV was settled in [1]; for $O(1)$ dimensions, it is $\Theta(\sqrt{n})$ and for polylog n dimensions, $\Theta(n^{1+o(1)})$ under QSETH, the quantum analogue of SETH. In this work we also examine the quantum complexity of Hopcroft's problem in an arbitrary number of dimensions d.

In general, we are interested in investigating quantum speedups for computational geometry problems. In recent years, there have been several works researching this topic. First, Ambainis and Larka gave a nearly optimal $O(n^{1+o(1)})$ quantum algorithm for the Point-On-3-Lines (detecting whether three lines are concurrent among the n given) and similar problems [11]. This problem is closely connected to fine-grained complexity as well, as it is an instance of the 3-Sum-Hard problem class. Classically, it is conjectured that 3-Sum cannot be solved faster than $O(n^2)$; the authors also conjectured a quantum analogue that 3-Sum cannot be solved quantumly faster than $O(n)$, and Buhrman et al. used this conjecture to prove conditional quantum lower bounds on various geometrical problems [14]. Aaronson et al. studied the quantum complexity of the Closest Pair problem (finding the closest pair of points among the n given), proving an optimal $\widetilde{\Theta}(n^{2/3})$ running time in $O(1)$ dimensions using a quantum walk algorithm with a dynamic history-independent data structure for storing points that is able to detect ϵ-close pairs of points [1]. For Bipartite Closest Pair problem in d dimensions (finding the closest pair of points between two sets of size n), they gave an $O(n^{1-1/2d+\delta})$ time quantum algorithm for any $\delta > 0$. For more results in quantum algorithms for computational geometry, see [39, 38, 12, 42, 43, 31].

For Hopcroft's problem, the best classical results give complexity $O((nm)^{2/3} + m \log n + n \log m)$, where n and m are the number of lines and points, respectively. In d dimensions, these generalize to $O((nm)^{d/(d+1)} + m \log n + n \log m)$ complexity [19]. The first complexity is unconditionally optimal if the algorithm needs to list all incidences, since there exists a planar construction with $\Omega((nm)^{2/3})$ incidence pairs ([25], Section 6.5.). It is also believed to be optimal for detection as well, with matching lower bounds in some models [27]. The dependence on n and m is symmetric since Hopcroft's problem is self-dual, in the sense that there is a geometric transformation which maps lines to points and vice versa, while preserving the point-line incidences ([25], Section 14.3.).

Finally, quite often the quantum query complexity of a problem matches its time complexity, like in Unstructured Search [29, 13], Element Distinctness [9, 8, 32], Closest Pair [1], Claw Finding [41, 47], just to name a few. In other cases even the precise query

complexity is not yet clear, for example, Triangle Finding [33] or Boolean Matrix Product Verification [16, 24]. In the case of Hopcroft's problem, its quantum query complexity can be easily characterized to be $\Theta((nm)^{1/3}+\sqrt{n}+\sqrt{m})$ from known results, see Theorem 5. The query-efficient algorithm does not immediately generalize to time complexity; therefore, the main focus here falls to improving the performance of the relevant classical data structures quantumly, which we find interesting.

1.1 Our results

In this work we show two quantum algorithms for Hopcroft's problem with time complexity $\widetilde{O}(n^{5/6})$. This constitutes a polynomial speedup over the classical $O(n^{4/3})$ time. We obtain our results by speeding up classical geometric data structures using different quantum techniques. We look at two underlying fundamental problems one usually encounters on the way to solve Hopcroft's problem.

1. **Simplex range searching.** In simplex range searching, the input is a set of n points in the d-dimensional space. A query then asks whether a given simplex contains any of the given points. The query may also ask to list or count the points in the simplex, among other variants [4]. Usually, there is some *preprocessing time* to precompute the data structure and some *query time* to answer each query. Classically, these complexities are well-understood; in a nutshell, a data structure of size m can be constructed in $\widetilde{O}(m)$ time and each query can then be answered in $\widetilde{O}(n/m^{1/d})$ time [35], and this is matched by lower bounds in the semigroup model [23, 22]. If the allowed memory size is linear, then preprocessing and query times become respectively $O(n\log n)$ and $O(n^{1-1/d})$ [17]. In this paper we require a variant of simplex range queries which we call *hyperplane emptiness queries*, where we have to determine whether a query hyperplane contains any of the given points. We show that quantumly we can speed up query time quadratically by using Montanaro's quantum algorithm for searching in the backtracking trees [36]:
   ▶ **Theorem 1.** *There is a bounded-error quantum algorithm that can answer hyperplane emptiness queries in $\widetilde{O}(\sqrt{n^{1-1/d}})$ time.*
   We note that this result is not really specific to the hyperplane emptiness queries, as all what we are doing is speeding up search in the *partition tree* data structure [17], thus this result can applied to other types of queries as well. However, this speedup does not extend to the counting version of simplex queries, since essentially, our procedure implements a search for a marked vertex in a tree using quantum walk. Using this result, we show a quantum speedup for Hopcroft's problem in d dimensions:
   ▶ **Theorem 2.** *There is a bounded-error quantum algorithm which solves Hopcroft's problem with n hyperplanes and $m \le n$ points in d dimensions in time:*
   - $\widetilde{O}(n^{\frac{d}{2(d+1)}}m^{1/2})$, *if* $m \ge n^{\frac{d}{d+1}}$;
   - $\widetilde{O}(n^{1/2}m^{\frac{d-1}{2d}})$, *if* $m \le n^{\frac{d}{d+1}}$.

   If $n=m$, then the algorithm has complexity $\widetilde{O}(n^{1-\frac{1}{2(d+1)}})$.
   In particular, the complexity is $\widetilde{O}(n^{5/6})$ in 2 dimensions for $n=m$.
2. **Planar point location.** The second approach is to use point location queries. One can use usually use classical point location data structures to determine whether a query point lies on the boundary of a planar region it belongs to. More specifically, we consider only *planar* point location data structures in the *line arrangements*. A set of n lines partitions the plane into $O(n^2)$ regions; this an old and well-researched topic, with many approaches to construct a data structure that holds the description of these regions in $O(n^2)$ time, the same amount of space and polylog n point location query time [25] (in fact, $O(n^d)$ preprocessing time and space and $O(\log n)$ query time in d dimensions [21, 20]).

In addition, there are also dynamic data structures with the same preprocessing and query times. More specifically, one can insert or remove a line in time $O(n)$ (or $O(n^{d-1})$ in d dimensions) [37]. We take an opportunity to employ such a data structure in a *quantum walk* algorithm on a Johnson graph to solve Hopcroft's problem. In particular, we develop a history-independent randomized data structure for storing an arrangement of an r-subset of n lines with ability to perform line insertion/removal in $O(r \operatorname{polylog} n)$ time and point location in $\operatorname{polylog} n$ time, requiring $O(r \operatorname{polylog} n)$ memory storage.

To do that, we store k-levels of the line arrangement in the history-independent skip lists a la Ambainis [9]. A k-level of a line arrangement is a set of segments of lines such that there are exactly k lines above each edge. Turns out that skip lists are ideal for encoding the k-levels. For example, when a new line is inserted, it splits each k-level in two parts, one of which will still belong to the k-level, but the other will belong to the $(k+1)$-level. We can then "reglue" these two tails to the correct levels of the arrangements in $\operatorname{polylog} n$ time by utilizing the properties of the skip list, all while keeping the history independence of the data structure. Using this data structure, we show the following quantum speedup for Hopcroft's problem in 2 dimensions:

▶ **Theorem 3.** *There is a bounded-error quantum algorithm that solves Hopcroft's problem with n lines and $m \leq n$ points in the plane in time:*

- $\widetilde{O}(n^{1/3}m^{1/2})$, *if* $n^{2/3} \leq m$;
- $\widetilde{O}(n^{2/5}m^{2/5})$, *if* $n^{1/4} \leq m \leq n^{2/3}$;
- $\widetilde{O}(n^{1/2})$, *if* $m \leq n^{1/4}$.

In particular, the complexity is $\widetilde{O}(n^{5/6})$ when $n = m$.

Both of Theorems 2 and 3 have their pros and cons. Theorem 2 is arguably simpler, since it is a quite direct application of the quantum speedup for backtracking. It also has a lower polylogarithmic factor hidden in the $\widetilde{O}$ notation, only $\log n$ compared to $\log^6 n$ in Theorem 3. However, Theorem 3 gives better asymptotic complexity if the number of lines n differ from the number of points m. On the other hand, Theorem 2 gives a speedup in the case of an arbitrary number of dimensions, while Theorem 3 has something to say only about the planar case; we leave a possible generalization of the quantum walk approach to larger dimensions for future research.

2 Preliminaries

We assume that the sets of points and lines are both in a general position (no two lines are parallel, no three lines intersect at the same point, no three points lie on the same line). Our algorithms work in the standard quantum circuit model augmented with Quantum Random Access Gates that allow to perform read/write operations in superposition; for details, see Appendix A. One of our building blocks is the following version of Grover's search:

▶ **Theorem 4** (Grover's search with bounded-error inputs [5, 30])**.** *Let $\mathcal{A} : [N] \to \{0,1\}$ be a bounded-error quantum procedure with running time T. Then there exists a bounded-error quantum algorithm that computes $\bigvee_{i\in[N]} \mathcal{A}(i)$ with running time $O(\sqrt{N}(T + \log N))$.*

Effectively, this result states that even if the inputs to Grover's search are faulty with constant probability, no error boosting is necessary, which would add another logarithmic factor to the complexity. We say that an algorithm is *bounded-error* if its probability of incorrect output is some constant strictly less than $1/2$.

3 Query complexity

Before examining time-efficient quantum algorithms, we take a look at the quantum query complexity of Hopcroft's problem, which in this case can be fully characterized. We assume that with a single query, we can obtain the description of any given line or point. In Appendix B, we prove the following:

▶ **Theorem 5.** *The quantum query complexity of Hopcroft's problem on n lines and m points in two dimensions is $\Theta(n^{1/3}m^{1/3} + \sqrt{n} + \sqrt{m})$.*

In particular, this proves that Theorem 3 is asymptotically optimal for $m \leq n^{1/4}$. The query complexity and the complexities of our algorithms are shown graphically in Figure 1.

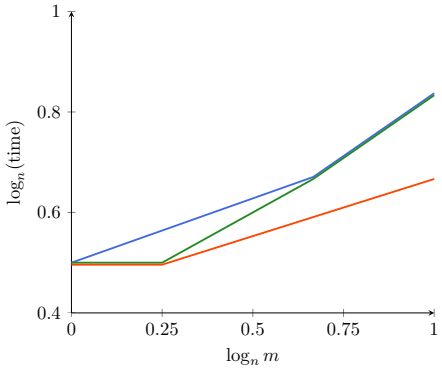

Figure 1 Quantum complexity of Hopcroft's problem in 2 dimensions on n lines and m points, assuming $m \leq n$. The blue line shows the complexity of the quantum algorithm with the partition tree (Theorem 2); the green line shows the complexity of the quantum walk algorithm with the line arrangement data structure (Theorem 3); the red line shows the query complexity (Theorem 5).

The upper bound in Theorem 5 is given by the algorithm of Tani [41]. There is an obvious hurdle in implementing it in the same time complexity here. Their quantum walk would require a history-independent dynamic data structure for storing a set of lines and a set of points supporting the detection of incidence existence. Even assuming the most optimistic quantum versions of the known data structures, the sufficiently powerful speedup looks unfeasible. The next two sections describe the algorithms we have obtained by speeding up two fundamental geometrical query data structures quantumly.

4 Algorithm 1: quantum backtracking with partition trees

We begin with a brief overview of the classical partition tree data structure and its quantum speedup, and then proceed with the description of the quantum algorithm.

4.1 Partition trees

The partition tree is a classical data structure designed for the task of *simplex range searching*. In this problem, one is given n points in a d-dimensional space; the task is to answer queries where the input is a simplex and the answer is the number of points inside that simplex. The *partition tree* is a well-known data structure which can be used to solve this task [4].

This data structure can be described as a tree in the following way. The tree stores n points and each subtree stores a subset of these points. Each interior vertex v is attributed with a simplex $\Delta(v)$ such that all of the points stored in this subtree belong to the interior

of $\Delta(v)$. For each interior vertex v, the subsets of the points stored in its children subtrees form a partition of the points stored in the subtree of v. Each leaf vertex stores a constant number of points in a list. For our purposes, each interior vertex is attributed only with information about its children and no information about the points stored in its subtree.

In this work, we are interested in the *hyperplane emptiness queries*. Given n points in the d-dimensional space, the task is to answer queries where the input is an arbitrary hyperplane and the answer is whether there is a point that lies on the given hyperplane. These queries can also be answered using partition trees:

▶ **Lemma 6** (Hyperplane emptiness query procedure)**.** *Let $\mathcal{T}$ be a partition tree storing a set of points. Let the* tree query cost *$c(\mathcal{T})$ be the maximum number of simplices of $\mathcal{T}$ that intersect an arbitrary hyperplane. Then the hyperplane emptiness query can be answered in $O(c(\mathcal{T}))$ time.*

Proof. The procedure for answering a query is as follows. We start at the root and traverse $\mathcal{T}$ recursively. If the current vertex is an interior vertex v and the query hyperplane is h, then we recurse only in the children of v such that $\Delta(v)$ intersects h. If the current vertex is a leaf vertex, we check whether any of its points lies on h. The running time is evidently linear in the number of simplices intersecting h. ◀

There are different ways to construct partition trees, but a long chain of works in computational geometry resulted in an optimal version of the partition tree [17]. Even though their goal is to answer simplex queries, in fact the main result gives an upper bound on $c(\mathcal{T})$ for their partition tree:

▶ **Theorem 7** (Partition tree [17])**.** *For any set of n points in d dimensions, there is a partition tree $\mathcal{T}$ such that:*

- *it can be built in $O(n \log n)$ time and requires $O(n)$ space;*
- *$c(\mathcal{T}) = O(n^{1-1/d})$; hence, a hyperplane emptiness query requires $O(n^{1-1/d})$ time;*
- *each vertex has $O(1)$ children and the depth of the tree is $O(\log n)$.*

To speed up the emptiness query time of the partition tree quantumly we use the quantum backtracking algorithm [36]. Their quantum algorithm searches for a *marked* vertex in a tree $\mathcal{S}$. The markedness is defined by a black-box function $P : V(\mathcal{T}) \to \{\text{TRUE}, \text{FALSE}, \text{INDETERMINATE}\}$. For leaf vertices v, we have $P(v) \in \{\text{TRUE}, \text{FALSE}\}$. A vertex v is marked if $P(v) = \text{TRUE}$, and the task is to determine whether $\mathcal{S}$ contains a marked vertex.

The root of $\mathcal{S}$ is known and the rest of the tree is given by two other black-box functions. The first, given a vertex v, returns the number of children $d(v)$ of v. The second, given v and an index $i \in [d(v)]$, returns the i-th child of v. The main result is a quantum algorithm for detecting a marked vertex in $\mathcal{S}$:

▶ **Theorem 8** (Quantum algorithm for backtracking [36, 10])**.** *Suppose you are given a tree $\mathcal{S}$ by the black boxes described above and upper bounds T and h on the size and the height of the tree. Additionally suppose that each vertex of $\mathcal{S}$ has $O(1)$ children. Then there is a bounded-error quantum algorithm that detects a marked vertex in $\mathcal{S}$ with query and time complexity $O(\sqrt{Th})$.*

When we apply it to the partition tree from Theorem 7, we get:

▶ **Theorem 1.** *There is a bounded-error quantum algorithm that can answer hyperplane emptiness queries in $\widetilde{O}(\sqrt{n^{1-1/d}})$ time.*

Proof. The procedure of Lemma 6 examines a subtree $\mathcal{S}$ of $\mathcal{T}$. We will apply Theorem 8 to $\mathcal{S}$. Suppose that h is a query hyperplane. The black box P returns INTERMEDIATE for any interior vertex v and for a leaf vertex v returns TRUE iff some point stored in v lies on h. The second black box returns the number of children of v in $\mathcal{T}$ if $\Delta(v)$ intersects h and 0 otherwise. The black box returning the i-th child simply fetches it from the partition tree that is stored in memory. All of these black boxes require only constant time to implement. Since we know that $|\mathcal{S}| = O(n^{1-1/d})$ and the height of $\mathcal{S}$ is $O(\log n)$ from Theorem 7, there is a quantum algorithm that solves the problem in $O(\sqrt{n^{1-1/d} \cdot \log n})$ time by Theorem 8. ◀

4.2 Quantum algorithm

Now we can apply the previous theorem to Hopcroft's problem.

▶ **Theorem 2.** *There is a bounded-error quantum algorithm which solves Hopcroft's problem with n hyperplanes and $m \le n$ points in d dimensions in time:*

- $\widetilde{O}(n^{\frac{d}{2(d+1)}} m^{1/2})$, *if* $m \ge n^{\frac{d}{d+1}}$;
- $\widetilde{O}(n^{1/2} m^{\frac{d-1}{2d}})$, *if* $m \le n^{\frac{d}{d+1}}$.

If $n = m$, then the algorithm has complexity $\widetilde{O}(n^{1-\frac{1}{2(d+1)}})$.

Proof. In the first case, we partition the whole set of points into m/r groups of size $r = n^{\frac{d}{d+1}}$. Using Grover's search, we search for a group that contains a point belonging to some line. To determine whether it's true for a fixed group, first we build a partition tree of Theorem 7 to store these points. Then we run Grover's search over all lines and determine whether a line contains some point from the group using the quantum query procedure from Theorem 1. Overall, the complexity of this algorithm (without logarithmic factors) is

$$O\left(\sqrt{\frac{m}{r}}\left(r + \sqrt{n} \cdot \sqrt{r^{1-1/d} \cdot \log r}\right)\right) = O\left(\sqrt{mr} + \sqrt{\frac{nm}{r^{1/d}} \log r}\right) = O\left(\sqrt{mn^{\frac{d}{d+1}} \log n}\right)$$

If we use the variation of Grover's search with bounded-error inputs (Theorem 4), then we do not incur extra logarithmic factors. If $m \le n^{\frac{d}{d+1}}$, then we simply build the partition tree on all m points, then use Grover's search over all lines and query the partition tree for each of them. The complexity in that case is

$$O\left(m \log m + \sqrt{n} \cdot \left(\sqrt{m^{1-1/d} \cdot \log m} + \log n\right)\right) = O\left(\sqrt{nm^{1-1/d}} \cdot \log n\right),$$

because the second term dominates the first (up to logarithmic factors). ◀

5 Algorithm 2: quantum walk with line arrangements

First we describe a classical history-independent data structure for storing an arrangement of a set of lines. After that, we describe the quantum walk algorithm that uses it for solving Hopcroft's problem.

5.1 Line arrangements

We begin with a few definitions (for a thorough treatment, see e.g. [25]). For a set of lines L, the *line arrangement* $\mathcal{A}(L)$ is the partition of the plane into connected regions bounded by the lines. The convex regions with no other lines crossing them are called *cells* and their sides are called the *edges* of the arrangement (note that some cells may be infinite). The intersection points of the lines are called the *vertices* of the arrangement.

For a set of lines L in a general position, the *k-level* is the set of edges of $\mathcal{A}(L)$ such that there are exactly k lines above each edge (for the special case of a vertical line, we consider points to the left of it to be "above"). By this construction each k-level forms a polygonal chain. Our data structure will store the line arrangement of a subset S of lines by keeping track of all $|S|$ levels, with each level being stored in a skip list. We will be able to support the following operations:

- Answering whether a point lies on some line in $O(\log^6 n)$ time.
- Inserting or removing a line in $O(|S| \log^4 n + \log^6 n)$ time.

5.2 Skip lists

We will need a history-independent data structure which can store a set of elements and support polylogarithmic time insertion/removal operations. For that purpose use the skip list data structure by Ambainis from the Element Distinctness algorithm [9]. Among other applications, it was also used by [1] for the Closest Pair problem, where they also gave a brief description. Here we shortly describe only the details required in our algorithm and rely on the facts already proved in these papers.

Suppose that the skip list stores some set of elements $S \subseteq [N]$, according to some order such that comparing two elements requires constant time. In a skip list, each element $i \in S$ is assigned an integer $\ell_i \in [0, \ldots, \ell_{\max}]$, where $\ell_{\max} = \lceil \log_2 N \rceil$. The skip list itself then consists of $\ell_{\max} + 1$ linked lists where the k-th list contains all $i \in S$ such that $\ell_i \geq k$. We will call the k-th linked list by the k-th layer (to not confuse them with k-levels). In other words, each element $i \in S$ is attributed with $\ell_i + 1$ pointers, where the k-th of them points to the smallest element j such that $j > i$ and $\ell_j \geq k$, or to `Null`, if there is no such j. The first element of the skip list is called the *head* and it only stores the $\ell_{\max}$ pointers, the beginnings of each layer (it is convenient to imagine this element storing value 0, which is smaller then any element of S).

The search of an element $i \in S$ is implemented in the following way. First, we traverse the $\ell_{\max}$-th layer to find the last element j such that $j \leq i$. If $j = i$, we are done; otherwise, traverse the layer below starting from j to find the last element $j' \leq i$ there. By repeating such iterations, we will find i. Insertion of $i \notin S$ is implemented similarly: first we find the last element $j_k \in S$ such that $j < i$, for all layers k. Then we update the pointers: for each layer $k \leq l_i$, we set the pointer from i to be be equal to the pointer from j_k; then we set the pointer from j_k to i. If an operation requires more than $O(\log^4 N)$ time, it is terminated.

To store the elements in memory, a specific hash table is used. Element's entry contains the description of the element together with other data attributed to it (in particular, the values of the pointers). We will not describe the details of the implementation, as it is the same as Ambainis'. The whole data structure can sometimes malfunction (e.g., the hash table buckets can overflow or the operation of the skip list can take too long), but it is shown in [9] that probability of such is small. More specifically, the probability that at least one operation malfunctions among $O(N)$ operations is only $O(1/\sqrt{N})$. Thus, as we are aiming at a sublinear algorithm, we don't need to worry about the error probability of the skip lists. We also note that the memory requirement of Ambainis' skip list is $O(r \log^3 N)$, if at most r elements need to be stored.

5.3 Data structure

Our data structure will operate mainly using the intersection points of the lines. To keep the unique description of the data for history independence, we describe each intersection point in the following way. Suppose the given lines are labeled $\ell_1, \ldots, \ell_n$. For any two lines ℓ_i and

ℓ_j, let $P_{i,j}$ be its intersection point. In an arrangement which includes both of these lines, we describe the left and right edges of ℓ_i connected to $P_{i,j}$ by $\texttt{left}(\ell_i, \ell_j)$ and $\texttt{right}(\ell_i, \ell_j)$. In that case there will be a k such that the k-level contains the edges $\texttt{left}(\ell_i, \ell_j)$ and $\texttt{right}(\ell_j, \ell_i)$, and the $(k+1)$-level contains the edges $\texttt{left}(\ell_j, \ell_i)$ and $\texttt{right}(\ell_i, \ell_j)$. We describe $P_{i,j}$ in the first case by the by the pair of integers $\nu_{i,j} = (i, j)$ and by $\nu_{j,i} = (j, i)$ in the second case, see Figure 2. We call these pairs *path points* of $P_{i,j}$. Note that we can calculate the coordinates of any path point in constant time as it description consists of the indices of the lines.

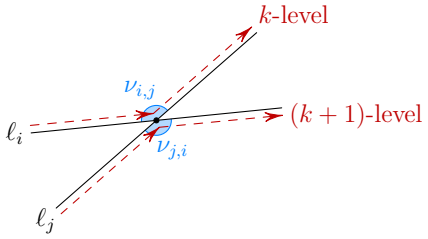

Figure 2 Path points of a line intersection.

Now we will describe the data structure, which stores an arrangement of a subset of given lines. It will operate with multiple skip lists each storing a set of path points of the arrangement. To ensure the unique representation of the data, we encode the pointers of the skip lists with the values of the path points themselves. To represent the beginning of a level from line ℓ_i, we use a "fictitious" starting path point $\nu_{i,i} = (i, i)$. The last element of skip lists we encode with a special "null" path point $\nu_{\texttt{Null}}$. We implement the following skip lists:

- The skip lists that contain the path points of the current k-levels in order from left to right. These skip lists are stored implicitly, since adding and removing lines changes the indexing of the levels and the levels themselves. For each path point ν stored in such a skip list, we additionally store an array $\texttt{Next}_\nu[0 \dots l_{\max}]$ storing the values of next path points of its skip list for each skip list layer.
- `Start` – contains the heads of all level skip lists in the current arrangement. If the first edge of a k-level belongs to ℓ_i, the head of its skip list is $\nu_{i,i}$, and we additionally store $\texttt{Next}_{\nu_{i,i}}$ to access the respective level skip list. The heads are ordered by the slope of the lines with the x-axis corresponding to the head path points.

Further we will describe the implementation of the operations.

5.3.1 Point location

To detect whether a point belongs to some line, we essentially binary search through all k-levels and check whether the given point is strictly above, below or belongs to that level. The binary search is implicitly performed by searching in the skip list `Start`. For a given level, we then search for its edge such that the x-coordinate of the point belongs to the projection of that edge on the x-axis. When this edge is found, we check the relative vertical position of this point in constant time.

Essentially we have two nested searches in the skip list structure, so the complexity of this step is $O(\log^8 n)$, but we can show a better estimate. Ambainis ([9], see the proof of Lemma 6) showed that in a skip list operation, at most $O(\log^2 n)$ pointer accesses are necessary. We run the search in the inner skip list only after accessing a pointer. Therefore, the outer search requires $O(\log^4 n)$ steps and the inner searches altogether require $O(\log^6 n)$ steps, so we improve our estimate to $O(\log^6 n)$.

5.3.2 Line insertion and removal

We will only describe the procedure of inserting a line in the data structure, as removing a line can be implemented by a reverse quantum circuit. Suppose the line to be inserted is ℓ_i; our task is to correctly update the pointers of the skip lists. As we will see, conveniently it suffices to update only the pointers of the new edges created by the insertion of the new line.

Our first step is to construct the edge along the given line. Figure 3 shows an example of the new edges collinear with ℓ_i being created. Suppose that P_{i,j_1} and P_{i,j_2} are two consecutive intersection points with ℓ_i (P_{i,j_1} is left of P_{i,j_2}). Some k-level will pass through an edge connecting P_{i,j_1} and P_{i,j_2}. This level will also pass through $\texttt{left}(\ell_{j_1}, \ell_i)$ and $\texttt{right}(\ell_{j_2}, \ell_i)$, as all edges of a level are directed from left to right. Therefore, the edge should connect $\nu_{j_1,i}$ with ν_{i,j_2}. There are two special cases for the first and the last edge; in the first case, the first path point is $\nu_{i,i}$ and for the second, the second path point is $\nu_{\texttt{Null}}$. According to this order, we insert all path points to the skip list starting with $\nu_{i,i}$ (and using the pointers `Next`). However, we don't insert $\nu_{i,i}$ into `Start` yet.

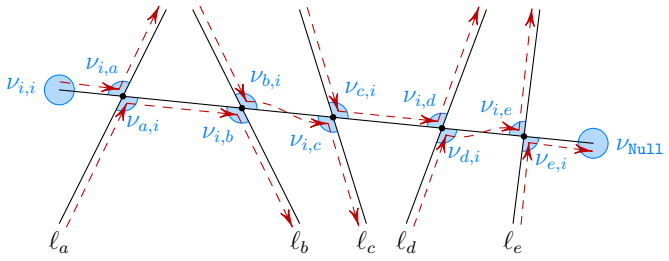

Figure 3 New edges along the inserted line.

Next we will correct all of the level skip lists according to the updated arrangement. Essentially, our algorithm performs a sweep line from right to left which swaps the tails of skip lists at the intersection points of ℓ_i with other lines. First, we create an array containing the same set of path points as the skip list of ℓ_i except $\nu_{i,i}$ and $\nu_{\texttt{Null}}$ and sort them by the x coordinate (at the end of the procedure we null the array by applying this in reverse). We then examine the intersection points of ℓ_i with the other lines from right to left.

Suppose we examine the intersection point $P_{i,j}$ of ℓ_i with ℓ_j.Then there is some edge from the old arrangement from $\nu^{(1)}$ to $\nu^{(2)}$ along ℓ_j which intersects ℓ_i at $P_{i,j}$. The respective pair of path points is $\nu_{i,j}$ and $\nu_{j,i}$. Then some k-level will pass along ℓ_i through $\nu_{i,j}$ and $\nu^{(2)}$, and some adjacent level (either $(k+1)$-level or $(k-1)$-level) will pass along ℓ_j from $\nu^{(1)}$ to $\nu_{j,i}$. Observe that the tails of these levels (from this intersection point to the right) have been correctly updated by the sweep line. Therefore, we just need to swap the tails of these two level skip lists.

To find the edge from $\nu^{(1)}$ to $\nu^{(2)}$, we use the point location operation with $P_{i,j}$. Since we know that this point will belong to some k-level, we modify the point location operation so as to return the head $\nu_{h,h}$ of this k-level. Note that although the level skip lists are partially updated, they still represent the old arrangement to the left of the previously examined intersection point, since the sweepline operates from right to left. Now we have to swap the tails of the skip list with head $\nu_{i,i}$ after $\nu_{i,j}$ and the skip list with head $\nu_{h,h}$ after $\nu^{(1)}$.

Generally, suppose that we wish to swap the tails of skip lists with heads $\nu^{(a_h)}$ and $\nu^{(b_h)}$ after elements $\nu^{(a_t)}$ and $\nu^{(b_t)}$, respectively. By searching $\nu^{(a_t)}$ in the $\nu^{(a_h)}$ skip list, we find all $l_{\max}$ path nodes $\nu^{(a_l)}$ such that $\texttt{Next}_{\nu^{(a_l)}}[l]$ points to a path node after $\nu^{(a_t)}$, for each $l \in [l_{\max}]$. Similarly we define and find $\nu^{(b_l)}$ path nodes. Then we simply swap the values of $\texttt{Next}_{\nu^{(a_l)}}[l]$ and $\texttt{Next}_{\nu^{(b_l)}}[l]$ for all $l \in [0, l_{\max}]$.

To conclude the procedure, we insert $\nu_{i,i}$ (together with $\texttt{Next}_{\nu_{i,i}}$) into `Start`. As we only performed $O(r)$ skip list searching and insertion operations (swapping the tails has the same complexity as an element insertion, as it's only updating $2l_{\max}+2$ pointers) and a point location operation, the complexity of the procedure is $O(r\log^4 n+\log^6 n)$. If $r=n^p$ for some $p>0$, this simplifies to $O(r\log^4 n)$.

5.4 Quantum algorithm

We use the MNRS framework quantum walk on the Johnson graph [9, 34]. In this framework, we search for a marked vertex in an ergodic reversible Markov chain on a state space X defined by the transition matrix $P=(p_{x,y})_{x,y\in X}$. Let the subset of marked states be $M\subseteq X$. To perform the quantum walk, the following procedures need to be implemented:

- Setup operation with complexity S. This procedure prepares the initial state of the quantum walk:

$$|0\rangle|0\rangle \mapsto \sum_{x\in X}\sqrt{\pi_x}\,|x\rangle|0\rangle\,,$$

  where π_x is the stationary distribution of P.
- Update operation with complexity U. This procedure essentially performs a step of the quantum walk by applying the transformation:

$$|x\rangle|0\rangle \mapsto |x\rangle\sum_{y\in X}\sqrt{p_{x,y}}\,|y\rangle\,.$$

- Checking operation with complexity C. This procedure performs the phase flip on the marked vertices:

$$|x\rangle|y\rangle \mapsto \begin{cases} -|x\rangle|y\rangle & \text{if } x\in M,\\ |x\rangle|y\rangle & \text{otherwise.}\end{cases}$$

We examine the Johnson graph on the state space X being the set of all size r subsets of $[n]$. Two vertices $x,y\in X$ are connected in this graph if the intersection of the corresponding subsets has size $r-1$. For the Markov chain, the transition probability is $p_{x,y}=\frac{1}{r(n-r)}$ for all edges. Then we have the following theorem:

▶ **Theorem 9** (Quantum walk on the Johnson graph [9, 34]). *Let P be the random walk on the Johnson graph on size r subsets of $[n]$ with intersection size $r-1$, where $r=o(n)$. Let M be either empty or the set of all size r subsets that contain a fixed element. Then there is a bounded-error quantum algorithm that determines whether M is empty, with complexity*

$$O\left(S+\frac{1}{\sqrt{r/n}}\left(\frac{1}{\sqrt{1/r}}\cdot U+C\right)\right)=O\left(S+\sqrt{n}\cdot U+\sqrt{\frac{n}{r}}\cdot C\right).$$

We can now prove our result:

▶ **Theorem 3.** *There is a bounded-error quantum algorithm that solves Hopcroft's problem with n lines and $m\le n$ points in the plane in time:*

- $\widetilde{O}(n^{1/3}m^{1/2})$, *if* $n^{2/3}\le m$;
- $\widetilde{O}(n^{2/5}m^{2/5})$, *if* $n^{1/4}\le m\le n^{2/3}$;
- $\widetilde{O}(n^{1/2})$, *if* $m\le n^{1/4}$.

In particular, the complexity is $\widetilde{O}(n^{5/6})$ when $n=m$.

Proof. By the duality of Hopcroft's problem, we can exchange n and m; from here on, suppose that $m \geq n$. We do this, since in Theorem 2 it is important that the number of lines is larger and here it is important that the number of points is larger, but we wish to keep the meaning of n and m to avoid confusion. Our algorithm is a quantum walk on the Johnson graph of size r subsets of the given n lines. We will choose r depending on m, but r will always be m^p for some $p > 0$. A set S is marked if it contains a line such that there exists a point from the set of m given points that belongs to this line.

For the implementation, we follow the description of [1] for the quantum algorithm for closest points. For a set S, the state of the walk will be $|S, d(S)\rangle$, where $d(S)$ is our data structure for the line arrangement of S. We then implement the quantum walk procedures:

- For the Johnson graph, π is the uniform distribution. Thus, we first generate a uniform superposition over all subsets S in $O(\log \binom{n}{r}) = O(r \log n)$ time. Then we create $d(S)$ by inserting all lines of S into an initially empty data structure, requiring $O(r^2 \log^4 n)$ time.
- Suppose that S and S' are two size r subsets with $|S \cap S'| = r - 1$ so that $S' = (S \setminus \{i\}) \cup \{j\}$. We then represent a state $|S, d(S)\rangle |S', d(S')\rangle$ with $|S, d(S)\rangle |i, j\rangle$. As the Markov chain probabilities are the same for all edges, we need to implement the transition $|S, d(S)\rangle |0, 0\rangle \mapsto \sum_{i \in S} \sum_{j \notin S} |S', d(S)'\rangle |j, i\rangle$. To do that, first we create a uniform superposition of all $i \in S$ and $j \notin S$ in $O(r \log^4 n)$ time (see Appendix C), obtaining $\sum_{i \in S} \sum_{j \notin S} |S, d(S)\rangle |i, j\rangle$. Then, for fixed i and j, we remove ℓ_i from $d(S)$ and insert ℓ_j, obtaining $d(S')$; this takes $O(r \log^4 n)$ time. Finally, we swap the indices i and j in the second register in $O(\log n)$ time.
- The checking operation runs Grover's search over all m points and for each of them performs the point location operation. The complexity is $O(\sqrt{m} \log^6 n)$.

By Theorem 9 the complexity of the algorithm is

$$O\left(r^2 \log^4 n + \sqrt{n} r \log^4 n + \sqrt{\frac{n}{r}} \sqrt{m} \log^6 n\right).$$

Suppose that $m^{2/3} \leq n$ and pick $r = m^{1/3}$. Then the second term dominates the first and we can simplify the expression to

$$O\left(\sqrt{n} \log^4 n \left(r + \sqrt{\frac{m}{r}} \log^2 n\right)\right) = O(n^{1/2} m^{1/3} \log^6 n).$$

If we have $m^{1/4} \leq n \leq m^{2/3}$ we pick $r = (nm)^{1/5}$. Then we have $r \geq (n^{1+3/2})^{1/5} = \sqrt{n}$, and this time the first term in the complexity dominates the second, and the complexity is

$$O\left(r^2 \log^4 n + \sqrt{\frac{nm}{r}} \log^6 n\right) = O(n^{2/5} m^{2/5} \log^6 n).$$

Finally, for $n \leq m^{1/4}$, we don't have to use either the quantum walk or the history-independent data structure. First, we build any classical data structure for point location in a line arrangement with $O(n^2 \log n)$ build time and space and $O(\log n)$ query time (e.g. see [25], Chapter 11). Then we run Grover's search over all points and for each check whether it belongs to some line. The complexity in this case is

$$O(n^2 \log n + \sqrt{m}(\log m + \log n)) = O(\sqrt{m} \log m).$$ ◀

References

1 Scott Aaronson, Nai-Hui Chia, Han-Hsuan Lin, Chunhao Wang, and Ruizhe Zhang. On the Quantum Complexity of Closest Pair and Related Problems. In Shubhangi Saraf, editor, *35th Computational Complexity Conference (CCC 2020)*, volume 169 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 16:1–16:43, Dagstuhl, Germany, 2020. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.CCC.2020.16.

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 2015, pages 218–230, USA, 2015. Society for Industrial and Applied Mathematics. doi:10.5555/2722129.2722146.

3 Amir Abboud, Virginia Vassilevska Williams, and Oren Weimann. Consequences of Faster Alignment of Sequences. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming*, pages 39–51, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg. doi:10.1007/978-3-662-43948-7_4.

4 Pankaj K. Agarwal. *Simplex Range Searching and Its Variants: A Review*, pages 1–30. Springer International Publishing, Cham, 2017. doi:10.1007/978-3-319-44479-6_1.

5 Jonathan Allcock, Jinge Bao, Aleksandrs Belovs, Troy Lee, and Miklos Santha. On the quantum time complexity of divide and conquer, 2023. arXiv:2311.16401.

6 Jonathan Allcock, Jinge Bao, João F. Doriguello, Alessandro Luongo, and Miklos Santha. Constant-depth circuits for Uniformly Controlled Gates and Boolean functions with application to quantum memory circuits, 2023. arXiv:2308.08539.

7 Andris Ambainis. Polynomial Degree vs. Quantum Query Complexity. In *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science*, FOCS 2003, pages 230–239, Washington, DC, USA, 2003. IEEE Computer Society. doi:10.1109/SFCS.2003.1238197.

8 Andris Ambainis. Polynomial Degree and Lower Bounds in Quantum Complexity: Collision and Element Distinctness with Small Range. *Theory of Computing*, 1(3):37–46, 2005. doi:10.4086/toc.2005.v001a003.

9 Andris Ambainis. Quantum Walk Algorithm for Element Distinctness. *SIAM Journal on Computing*, 37(1):210–239, 2007. doi:10.1137/S0097539705447311.

10 Andris Ambainis and Martins Kokainis. Quantum algorithm for tree size estimation, with applications to backtracking and 2-player games. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2017, pages 989–1002, New York, NY, USA, 2017. ACM. doi:10.1145/3055399.3055444.

11 Andris Ambainis and Nikita Larka. Quantum Algorithms for Computational Geometry Problems. In Steven T. Flammia, editor, *15th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2020)*, volume 158 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 9:1–9:10, Dagstuhl, Germany, 2020. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.TQC.2020.9.

12 A. Bahadur, C. Dürr, T. Lafaye, and R. Kulkarni. Quantum query complexity in computational geometry revisited. In Eric J. Donkor, Andrew R. Pirich, and Howard E. Brandt, editors, *Quantum Information and Computation IV*, volume 6244, page 624413. International Society for Optics and Photonics, SPIE, 2006. doi:10.1117/12.661591.

13 Charles H. Bennett, Ethan Bernstein, Gilles Brassard, and Umesh Vazirani. Strengths and Weaknesses of Quantum Computing. *SIAM Journal on Computing*, 26(5):1510–1523, 1997. doi:10.1137/S0097539796300933.

14 Harry Buhrman, Bruno Loff, Subhasree Patro, and Florian Speelman. Limits of Quantum Speed-Ups for Computational Geometry and Other Problems: Fine-Grained Complexity via Quantum Walks. In Mark Braverman, editor, *13th Innovations in Theoretical Computer Science Conference (ITCS 2022)*, volume 215 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 31:1–31:12, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ITCS.2022.31.

15 Harry Buhrman, Bruno Loff, Subhasree Patro, and Florian Speelman. Memory Compression with Quantum Random-Access Gates. In François Le Gall and Tomoyuki Morimae, editors, *17th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2022)*, volume 232 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 10:1–10:19, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.TQC.2022.10.

16 Harry Buhrman and Robert Špalek. Quantum verification of matrix products. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm*, SODA 2006, pages 880–889, USA, 2006. Society for Industrial and Applied Mathematics. arXiv:quant-ph/0409035.

17 Timothy M. Chan. Optimal Partition Trees. *Discrete & Computational Geometry*, 47:661–690, 2012. doi:10.1007/s00454-012-9410-z.

18 Timothy M. Chan and R. Ryan Williams. Deterministic APSP, Orthogonal Vectors, and More: Quickly Derandomizing Razborov-Smolensky. *ACM Trans. Algorithms*, 17(1), 2021. doi:10.1145/3402926.

19 Timothy M. Chan and Da Wei Zheng. Hopcroft's Problem, Log-Star Shaving, 2D Fractional Cascading, and Decision Trees. *ACM Trans. Algorithms*, 2023. doi:10.1145/3591357.

20 Timothy M. Chan and Da Wei Zheng. *Simplex Range Searching Revisited: How to Shave Logs in Multi-Level Data Structures*, pages 1493–1511. SODA 2023. Society for Industrial and Applied Mathematics, 2023. doi:10.1137/1.9781611977554.ch54.

21 Bernard Chazelle. Cutting hyperplanes for divide-and-conquer. *Discrete & Computational Geometry*, 9:145–158, 1993. doi:10.1007/BF02189314.

22 Bernard Chazelle and Burton Rosenberg. Simplex range reporting on a pointer machine. *Computational Geometry*, 5(5):237–247, 1996. doi:10.1016/0925-7721(95)00002-X.

23 Bernard Chazelle and Emo Welzl. Quasi-optimal range searching in spaces of finite VC-dimension. *Discrete & Computational Geometry*, 4:467–489, 1989. doi:10.1007/BF02187743.

24 Andrew M. Childs, Shelby Kimmel, and Robin Kothari. The Quantum Query Complexity of Read-Many Formulas. In Leah Epstein and Paolo Ferragina, editors, *Algorithms – ESA 2012*, pages 337–348, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. arXiv:1112.0548.

25 Herbert Edelsbrunner. *Algorithms in Combinatorial Geometry*. Springer-Verlag, Berlin, Heidelberg, 1987. doi:10.1007/978-3-642-61568-9.

26 Jeff Erickson. On the relative complexities of some geometric problems. In *Proceedings of the 7th Canadian Conference on Computational Geometry*, pages 85–90. Carleton University, 1995. URL: https://jeffe.cs.illinois.edu/pubs/relative.html.

27 Jeff Erickson. New lower bounds for Hopcroft's problem. *Discrete & Computational Geometry*, 16:389–418, 1996. doi:10.1007/BF02712875.

28 Vittorio Giovannetti, Seth Lloyd, and Lorenzo Maccone. Quantum Random Access Memory. *Phys. Rev. Lett.*, 100:160501, 2008. doi:10.1103/PhysRevLett.100.160501.

29 Lov K. Grover. A Fast Quantum Mechanical Algorithm for Database Search. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, STOC 1996, pages 212–219, New York, NY, USA, 1996. Association for Computing Machinery. doi:10.1145/237814.237866.

30 Peter Høyer, Michele Mosca, and Ronald de Wolf. Quantum Search on Bounded-Error Inputs. In *Automata, Languages and Programming*, ICALP 2003, pages 291–299, Berlin, Heidelberg, 2003. Springer-Verlag. doi:10.1007/3-540-45061-0_25.

31 J. Mark Keil, Fraser McLeod, and Debajyoti Mondal. Quantum Speedup for Some Geometric 3SUM-Hard Problems and Beyond, 2024. arXiv:2404.04535.

32 Samuel Kutin. Quantum Lower Bound for the Collision Problem with Small Range. *Theory of Computing*, 1(2):29–36, 2005. doi:10.4086/toc.2005.v001a002.

33 François Le Gall. Improved Quantum Algorithm for Triangle Finding via Combinatorial Arguments. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, pages 216–225, 2014. doi:10.1109/FOCS.2014.31.

34 Frédéric Magniez, Ashwin Nayak, Jérémie Roland, and Miklos Santha. Search via Quantum Walk. *SIAM Journal on Computing*, 40(1):142–164, 2011. doi:10.1137/090745854.

35 Jiří Matoušek. Range searching with efficient hierarchical cuttings. *Discrete & Computational Geometry*, 10:157–182, 1993. doi:10.1007/BF02573972.

36 Ashley Montanaro. Quantum-Walk Speedup of Backtracking Algorithms. *Theory of Computing*, 14(15):1–24, 2018. doi:10.4086/toc.2018.v014a015.

37 Ketan Mulmuley and Sandeep Sen. Dynamic point location in arrangements of hyperplanes. *Discrete & Computational Geometry*, 8:335–360, 1992. doi:10.1007/BF02293052.

38 Kunihiko Sadakane, Noriko Sugarawa, and Takeshi Tokuyama. Quantum Computation in Computational Geometry. *Interdisciplinary Information Sciences*, 8(2):129–136, 2002. doi:10.4036/iis.2002.129.

39 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. doi:10.1007/3-540-45678-3_14.

40 Neil Sarnak and Robert E. Tarjan. Planar point location using persistent search trees. *Commun. ACM*, 29(7):669–679, 1986. doi:10.1145/6138.6151.

41 Seiichiro Tani. Claw finding algorithms using quantum walk. *Theoretical Computer Science*, 410(50):5285–5297, 2009. Mathematical Foundations of Computer Science (MFCS 2007). doi:10.1016/j.tcs.2009.08.030.

42 Nilton Volpato and Arnaldo Moura. Tight Quantum Bounds for Computational Geometry Problems. *International Journal of Quantum Information*, 07(05):935–947, 2009. doi:10.1142/S0219749909005572.

43 Nilton Volpato and Arnaldo Moura. A fast quantum algorithm for the closest bichromatic pair problem, 2010. URL: https://www.ic.unicamp.br/~reltech/2010/10-03.pdf.

44 Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theoretical Computer Science*, 348(2):357–365, 2005. Automata, Languages and Programming: Algorithms and Complexity (ICALP-A 2004). doi:10.1016/j.tcs.2005.09.023.

45 Ryan Williams. Pairwise comparison of bit vectors, 2017. URL: https://cstheory.stackexchange.com/q/37369.

46 Ryan Williams and Huacheng Yu. Finding orthogonal vectors in discrete structures. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA 2014, pages 1867–1877, USA, 2014. Society for Industrial and Applied Mathematics. doi:10.5555/2634074.2634209.

47 Shengyu Zhang. Promised and Distributed Quantum Search. In Lusheng Wang, editor, *Computing and Combinatorics*, pages 430–439, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. doi:10.1007/11533719_44.

A Model

We use the standard quantum circuit model together with Quantum Random Access Gates (QRAG) (see, for example, [6]). This gate implements the following mapping:

$$|i\rangle\,|b\rangle\,|x_1,\ldots,x_N\rangle \mapsto |i\rangle\,|x_i\rangle\,|x_1,\ldots,x_{i-1},b,x_{i+1},\ldots,x_N\rangle\,.$$

Here, the last register represents the memory space of N bits. Essentially, QRAG gates allow both for reading memory in superposition as well as writing operations. We note that both our algorithms require "read-write" quantum memory, so it is not sufficient to use the weaker "read-only" QRAM gate, which is enough for some quantum algorithms.

To keep the analysis of the algorithms clean, we abstract the complexity of basic underlying operations under the "unit cost". This includes the running time of:

- basic arithmetic operations on $O(\log n)$ bits;
- the implementation of QRAG and elementary gates;
- the running time of a quantum oracle, which with a single query can return the description of any point or line.

In the end, we measure the time complexity in the total amount of unit cost operations. The unit cost can be taken as the largest running time of the operations listed above, which will add a multiplicative factor in the complexity.

Assuming that an application of a QRAG gate takes unit time is also useful for utilizing the existing classical algorithms in the RAM model. The classical algorithms that we use work in the real RAM model, where arithmetic operations and memory calls on $O(\log n)$ bits are considered to be executed in constant time. Thus, we work in the quantum analogue of the real RAM, and if there is a time T classical real RAM algorithm, then we can use it in time $O(T)$ in this model. The actual implementation of QRAG is an area of open research and debate; however, there exist theoretical proposals that realize such operations in time polylogarithmic in the size of the memory, like the bucket brigade architecture of [28].

B Query complexity

Proof of Theorem 1. For the upper bound, Hopcroft's problem can be seen as an instance of the bipartite *subset-finding* problem, in which one is given query access to two sets X and Y of sizes n and m, respectively, and needs to detect whether there is a pair $x \in X$, $y \in Y$ satisfying some predicate $R : X \times Y \to \{0, 1\}$. For this problem, Tani gave a quantum algorithm with query complexity $O(n^{1/3}m^{1/3} + \sqrt{n} + \sqrt{m})$ [41].

For the lower bound, we can reduce the bipartite element distinctness problem (also known as Claw Finding) to Hopcroft's problem. In this problem, we have two sets of variables $x_1, \ldots, x_n \in [N]$ and $y_1, \ldots, y_m \in [N]$ and we need to detect whether $x_i = y_j$ for some i, j. Zhang proved that for this problem $\Omega(n^{1/3}m^{1/3} + \sqrt{n} + \sqrt{m})$ quantum queries are needed [47]. We reduce each x_i to a line $x = x_i$ and each y_j to a point $(y_j, 0)$. Then $x_i = y_j$ only iff some point belongs to some line, so the statement follows. ◀

C Implementation details

To generate a uniform state proportional to $\sum_{i \in S} |i\rangle$ from $|S, d(S)\rangle$, we can proceed as follows. First generate the uniform superposition $\sum_{k \in [r]} |k\rangle$ in $O(\log n)$ time. Then we can find the number of the k-th line in S by iterating over the elements of the `Start` skip list as in the usual linked list until we find the k-th one; this requires $O(r \log^4 n)$ time. We then null the register $|k\rangle$ by applying a reverse procedure and decrementing k in each step. Overall, we obtain the state proportional to $\sum_{i \in S} |i\rangle$ in $O(r \log^4 n)$ time.

To generate a uniform state proportional to $\sum_{j \notin S} |j\rangle$ from $|S, d(S)\rangle$, we can apply a different procedure. First, we can generate the state $\sum_{k \in [n]} |k\rangle |0\rangle$ in $O(\log n)$ time. For a fixed k, we can check whether $k \in S$ using $d(S)$ in $O(\log^4 n)$ time and write 1 in that case in the second register. By measuring the second register, we obtain the required state with probability $1 - r/n > 1/2$. We can use $O(\log n)$ copies of such state to obtain the required state with error probability only $O(1/n)$, which doesn't impact the final constant success probability. Overall, this step requires $O(\log^5 n)$ time, which is negligible compared to $O(r \log^4 n)$.