Optimal Query Bounds for Orthogonal Range Search Problems

Muhammad Abdullah, Andrew Gu, Anders Olsen

December 2021

1 Introduction

In orthogonal range searching problems, we are given a static set P of n points in $\mathbb{R}^d_{\geq 0}$, and asked to answer several queries online about these points. The requirement that all points have non-negative coordinates is a convention; if this is not the case then all points can easily be translated to fulfill this. Each query consists of a description of a d-dimensional cuboid R, and we are asked to return some information about the points contained in that cuboid, depending on the type of query:

- **Range emptiness:** return TRUE if R contains no points of P.
- **Range counting:** return the number of points of P contained in R.
- **Range reporting:** return the location of each point of P contained in R.
- **Range minima:** each point is initially associated with a static priority value. Return the lowest priority point in $P \cap R$.

Typically data structures are designed to respond to only one such type of query efficiently. We may also consider queries that are unbounded in some directions. These are known as k-sided queries, where k is the number of sides of the cuboid that are bounded. For example, a three-dimensional four-sided query might consist of the rectangle $[a, b] \times [0, y] \times [0, z]$.

Typically, for orthogonal range search problems, only data structures which use $n^{1+o(1)}$ space are of interest. Also, range reporting queries will necessarily need time at least equal to the number of points that are reported, so time complexities are typically written as $O(f(n) + k)$ where k is size of the answer.

Our paper is centered around the results from a 2021 paper by Nekrich. In Section 2, we describe his data structure that is able to answer the closely related two-dimensional range minima and three-dimensional five-sided query problems in the same asymptotic time and space complexities as have been previously achieved for the range emptiness problem, which closes a complexity gap between the two problems. Another result from Nekrich's paper is a data structure to answer range reporting in four dimensions with $O(\log n / \log \log n + k)$ query time, which is known to be optimal from older results. In Section 3, we mention some of those older results which prove lower bounds and show a proof that reduces the problem to explicitly constructing "hard" inputs for the range reporting problem.

2 Closing the Complexity Gap

2.1 Previous results

In the 1988, Chazelle [4] introduced a data structure that performed range minima queries in two dimensions in $O(n)$ space and $O(\log^{1+\epsilon}(n))$ time. At the time, there was a known solution for two-dimensional range emptiness queries in $O(n)$ space and $O(\log^{\epsilon}(n))$ time, leaving a gap between the two problems. Intuitively, this makes sense, as it should be harder to find the lowest-priority point in a rectangle than determining whether that rectangle is empty or not. Surprisingly, this gap was recently closed in 2021 by Nekrich [6], which gave a data structure for range minima queries matching the above bounds for range emptiness queries. The data structure relies on the concept of *shallow cuttings* in three dimensions. In the following sections, we will give a general overview of the data structure, first explaining shallow cuttings and their coverings. From there, we will describe a dominance data structure that is central to the final construction. Finally we will use the dominance data structure to construct the data structure that will answer range minima queries in $O(\log^{\epsilon}(n))$ time.

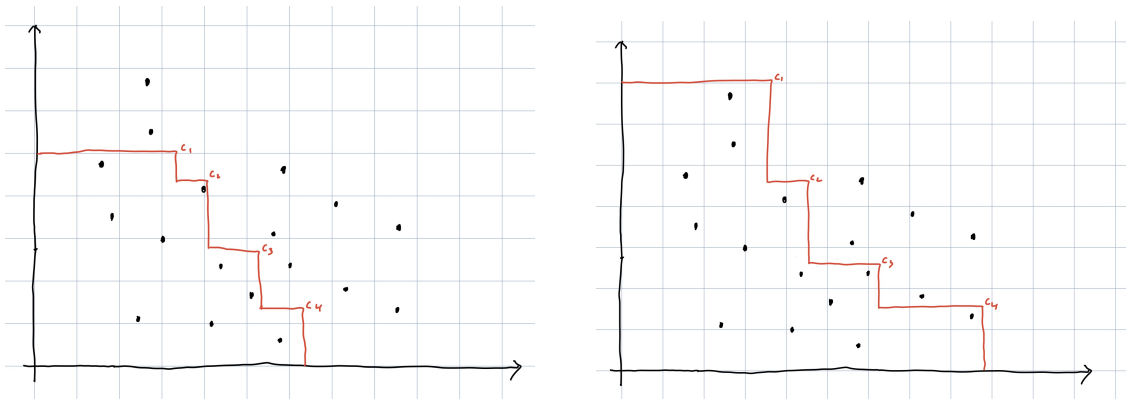
A recurring theme of this data structure is that we subdivide our query into several parts. In some cases, we will handle parts of the query with data structures from other authors, as they have a satisfactory complexity bounds for our purposes. These areas are not essential to the improvements made in the recent Nekrich paper, so we do not discuss these data structures in detail. These other data structures essentially serve to handle edge cases, where as the real meat of Nekrich's work is the clever use of shallow cuttings and recursive grid structures.

2.2 Shallow Cuttings

A point (x_0, x_1, x_2) **dominates** another point $y = (y_0, y_1, y_2)$ if $y \in [0, x_0] \times [0, x_1] \times [0, x_2]$ (that is, each coordinate of y is at most the corresponding coordinate of x). Given a set of points P in $\mathbb{R}^3$ ($|P| = m$), a **t-shallow cutting** of P is a set of $O(m/t)$ cuboids C (also called cells), each of the form $[0, a] \times [0, b] \times [0, c]$, satisfying the following properties;

1. $\forall C_i \in C$, $|C_i \cap P| \leq 2t$. (Each cell has a bounded number of points.)
2. $\forall q \in P$, if q dominates at most t points from P then q is covered by a cell.

Intuitively, we can cover points of P with low dominance in $O(m/t)$ cells such that each cell does not cover too many points. The set of points covered by C_i is called the **conflict list** of C_i and is denoted as $list(C_i) = P \cap C_i$.



(a) 2-shallow cutting

(b) 3-shallow cutting

Figure 1: Two shallow cuttings of the same point set (in two dimensions) but with different cut threshold. In each, c_i is the top right corner of cell C_i. The red line shows portions of the upper boundaries of each cell, which extend to both axes.

The diagrams of Figure 1 are not strictly representative of the case we are working with, since the data structure uses three-dimensional cuttings, but they adequately illustrate the concept of a cutting.

Given the above definitions, we give the main theorem for this data structure. Intuitively, this theorem asserts that we can construct an 'almost' covering of a t-shallow cut using unbounded rectangular columns, such that $O(m/d)$ points are not covered and each column does not contain too many points.

Theorem 1. Let C denote a t-shallow cutting of a three-dimensional point set P, $|P| = m$. For any integer $d > 0$ there exists a subset P' of P and a set of three-dimensional rectangles $R = \{R_1, R_2, \ldots, R_s\}$, such that

- $|P'| = O(m/d)$. (The set of points that are not covered, P', is bounded.)
- The rectangles are not bounded in the z direction, i.e. they only have 4 sides.
- $|R_i \cap P| = O(td^4)$. (Each rectangle has a bounded number of points.)
- The conflict list of every cell (minus P') is covered by $O(d^3)$ rectangles:

$$list(C_i) \cap (P/P') \subseteq \bigcup_{j=1}^{O(d^3)} list(C_i) \cap R_{i_j}.$$

We will treat this theorem as a given, [6] contains a detailed construction of the covering for the interested reader. In particular, the proof of this theorem is constructive, so we may use the same process to find the set of covering rectangles R_i when constructing our data structure.

2.3 Dominance Data Structure

Using the theorem for shallow cuttings, we can construct a data structure that answers a special type of query called $\log n$**-capped dominance query**. In this type of query, for a query point q, we return all k points that q dominates if $k \leq \log n$. Otherwise, we return $NULL$. We only need to consider capped queries because if the data structure returns $NULL$, we can query a 'slower' data structure of Chazelle and report all points in the query range in the same asymptotic time bounds. This data structure of Chazelle has a runtime $O(\log^{1+\epsilon} n + k \log^{\epsilon} n)$, which is desirable only when $k \geq \log n$.

Using the shallow cutting theorem with $t = \log^6(n)$ (the exponent 6 is arbitrary, what matters is that it is polylogarithmic and thus its log is $O(\log \log n)$) and $d = \log(n)$, we get a set of uncovered points P'. These uncovered points are handled separately in a data structure [2] that takes $O(\log(n))$ bits per word and $O(\log \log(n) + k)$ time per query. Defining the **covered conflict list** P_j of each cell C_j as $P_j = list(C_j)/P'$, this data structure will answer $\log n$-capped dominance queries on P_j as follows:

1. For the point q, find the cell C_j which contains it. If it exists then we can find the cell in $\log \log n$ time. Otherwise, if q is not contained in any cell then we know that q dominates more than $t = \log^6 n$ points and we can return $NULL$.

2. Look at the covered conflict list of that cell P_j. We reduce our query point q to the rank space of P_j. Using section 2.3.1, we can enumerate each point that is dominated by q in $O(\log^{\epsilon} n)$ time.

3. Lastly we query the data structure on P' and answer all points dominated by q in $O(\log \log n + k)$ time.

2.3.1 Rank Reduction

We use the idea of rank reduction, turning coordinates into integer ranks for faster look up, to answer dominance enumeration queries. Considering the properties of coverings of shallow cuts, we can enumerate the ranks directly since each cell has $\leq 2t =$ polylog points. The process of turning those ranks back into coordinates is described next. Using Theorem 1 we know that each cell has a list of rectangles that cover it, $rlist(C_i) = \{R_{j_k} | P_i \subset \cup_k R_{j_k}\}$. We know that $|rlist(C_i)| = O(d^3)$, so in $O(\log(d))$ bits we can store a reference to all rectangles for that cell. For each rectangle, we know $|R_s| = O(td^4)$, thus in $O(\log d + \log t)$ bits we can store the rank of every point in R_s. So if we want to find the rank of a point in a cell, we first find the rectangle that contains it and then we find the rank of the point inside the rectangle. For the latter problem, we impose yet another data structure on the rectangle that answers another type of query, a **capped range selection** query. This query is done on a two-dimensional plane and consists of a rectangle Q and integer v. We return the v-th point inside Q, ordered via x-coordinate. This is a known problem and we can use a range tree, with some modifications, to answer this.

2.4 Putting it all together

Using the capped dominance data structure, we can describe a way to recursively divide the grid such that we are able to answer three-dimensional five-sided range reporting queries. However, this is different from our end goal which is a data structure for two-dimensional range minima queries.

It turns out that both of these queries are deeply linked and most data structures that can answer one could be modified, in the same bounds, to answer the other. This transformation is given in section 2.5. Then we only need to handle the three-dimensional five-sided reporting case, which will be the data structure we have been building towards.

Given a set of n points in space, we divide the xy-plane into a set of vertical and horizontal slabs, each parallel to the grid, which extend in the z direction. This division is done such that there are $\sqrt{n/\log^3 n}$ slabs of either type, and each slab contains $\sqrt{n \log^3 n}$ points. Computationally, this division can be easily done by a sweep plane.

On each slab, we impose the dominance data structure defined in the previous section. There is a global data structure on the slabs, denoted by D^t. Our eventual query will be handled by D^t and decomposed into a set of queries for the slabs. The five-sided reporting query would be the aggregation of the individual slab calls.

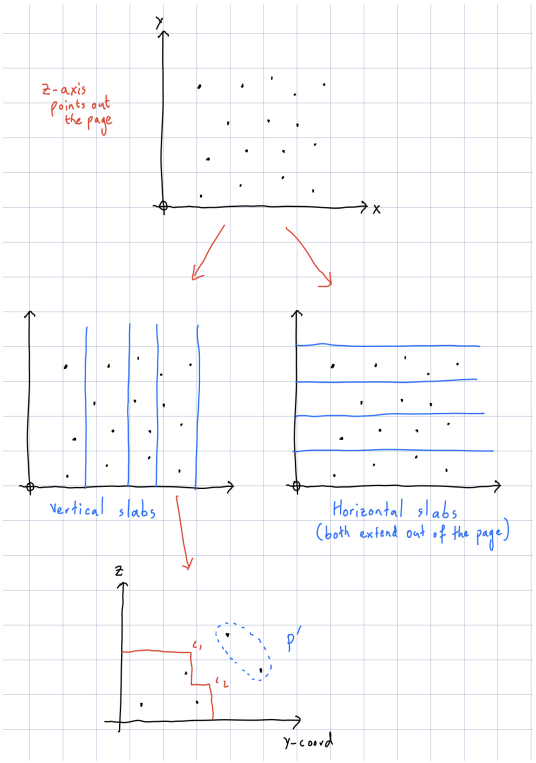


Figure 2: An overview of the data structure, we have described its construction from the bottom up in previous sections. In the base case, dominance queries are divided over C_1, C_2 and P'. In the above diagrams we are looking from top to bottom.

The decomposition is as follows; if the (five-sided) query is contained inside a slab (either a vertical or horizontal slab) then we query that slab. Otherwise the query must cross both vertical and horizontal slab boundaries, and we divide the query like shown in the first diagram below.

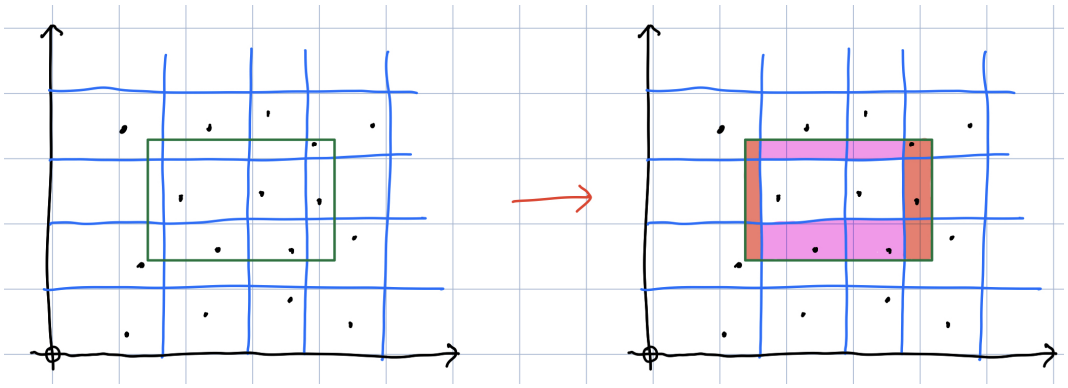


Figure 3: The edges of the five-sided query are decomposed into queries inside vertical (red) and horizontal (pink) slabs, while the inside of the query is handled by D^t. The time and space bounds of the query is dominated by the slab queries, not D^t.

The query to each slab is bounded on four sides. In order to answer a query for a slab, we query its recursive data structure, and take advantage of the fact that we are only bounding four sides. To answer a four-sided query, we again divide the query into a central portion of whole cells and partial portions on the sides, as shown in the second diagram below. In this case, there are only 3 partial portions, two of which are only bounded by 3 parameters. Thus we can use the dominance data structure to find all the points contained in our region, which takes $O(\log^\epsilon n)$ time, in addition to $O(k)$ to report each point. Note that the unbounded sides in these two regions are not the same, but we can deal with this by storing a dominance data structure for each slab in each of the 8 possible directions, which requires only a constant factor increase in space. The region of whole cells can be answered by the top-level data structure in $O(\log^\epsilon n + k)$. Finally, one of the partial portions is another four-sided query, which we must answer recursively. Thus the time required to answer a four-sided query on a data structure of n points is

$$q(n) = O(\log^\epsilon n) + q\left(\sqrt{n \log^3 n}\right) = O(\log^\epsilon n \log\log n),$$

plus $O(k)$ to report the points.

To answer our original five-sided query, we must answer a query on the top-level data structure and 4 four-sided queries on lower level data structures. As before, the top-level query takes $O(\log^\epsilon n)$ time, which is dominated by the 4 four-sided queries, which take $O(\log^\epsilon n \log\log n)$ time. Thus we can answer five-sided queries in this same runtime. Note that we can actually achieve a query time of $O(\log^\epsilon n)$ for any ϵ, as for each ϵ there exists some delta such that $\log^\delta n \log\log n = O(\log^\epsilon n)$. We can use this δ when constructing the data structure instead of the desired ϵ to achieve the desired complexity.

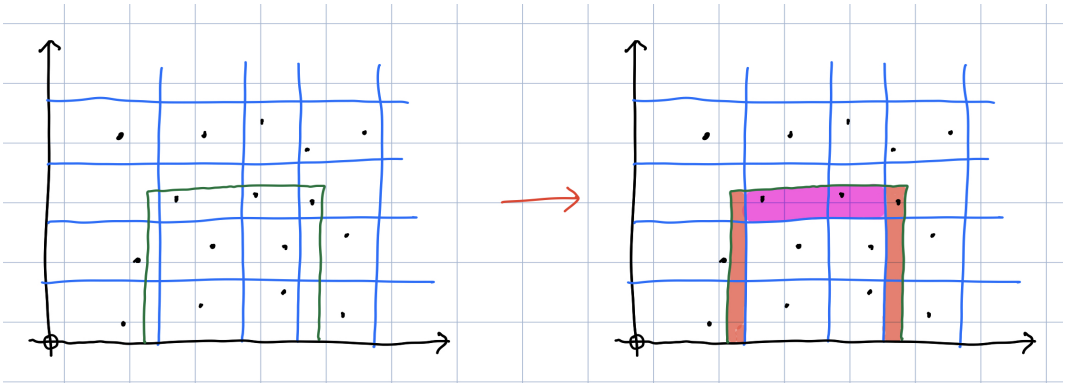


Figure 4: The edges of the four-sided query are decomposed into two three-sided queries inside vertical slabs (red) and one four-sided query inside horizontal (pink) slabs, while the inside of the query is handled by D^t. The time and space bounds of the query is dominated by the slab queries, not D^t.

2.5 Extending to Two-Dimensional Range Minima Queries

In order to support two-dimensional range minima queries instead of five-sided reporting queries, we replace the data structure at each level with one more suited to our needs. The top data structure now needs to know only the lowest priority point in each intersection of two slabs, and should be able to find the lowest priority point quickly in any rectangular region of slab intersections. This is just a specific case of the orthogonal range minima problem, and since we only have $O(n/\log^3 n)$ intersections of slabs, we can use previously-described range minima data structures while staying within our space and time bounds. We can use a data structure from [3] for this purpose, which contains the $O(n/\log^3 n)$ points in $o(n/\log^2 n)$ space, and can answer queries in $O(\log\log n)$ time, both of which are sufficient for our purposes.

We also need to modify the dominance data structure. Instead of supporting three-dimensional dominance reporting queries, we need to support two-dimensional dominance minima queries, which ask for the lowest priority point dominated by a given point q. This modification is non-trivial, but we can achieve the same $O(\log^\epsilon n)$ time bound and $O(n\log\log n)$ bits space bound as for the three-dimensional dominance reporting queries.

Now in order to answer a range minima query for an arbitrary rectangular region, we divide it as before into a middle region with sides given by slab boundaries and four external regions, each of which is bounded on only three sides. As with the three-dimensional four-sided queries before, each of these two-dimensional three-sided queries can be broken up into a middle query on the top-level data structure for that slab, two dominance queries, and another three-sided query on a smaller slab. This still gives the same recursive query time

$$q(n) = O(\log^\epsilon n) + q\left(\sqrt{n\log^3 n}\right) = O(\log^\epsilon n \log\log n)$$

for three-sided minima queries, and thus the same $O(\log^\epsilon n\log\log n) = O\left(\log^{\epsilon'} n\right)$ time for four-sided minima queries. Now when answering queries using this recursive data structure, we must compute $O(\log\log n)$ queries on top level data structures and $O(\log\log n)$ on dominance data

structures. To get the answer to the original range mimima query, we simply take the lowest-priority point between all of the points returned by these queries.

2.6 Space Requirements

Essential to this result is the fact that this data structure uses only linear space in the number of points, as the $O(\log^{\epsilon} n)$ runtime is a significant improvement over all previous results for linear-space data structures for this problem. The top-level data structure consumes $o(n/\log^2 n)$ space in machine words, so it certainly takes $o(n)$ bits. The top level dominance data structure takes $O(n \log\log n)$ bits, as each point requires $O(\log\log n)$ bits. We of course also have the space required for the lower level recursive data structures, which gives a space requirement (in bits) of

$$S(n) = O(n\log\log n) + 2\sqrt{n/\log^3 n}\, S\left(\sqrt{n\log^3 n}\right).$$

Letting $c(n) = S(n)/n$, we have

$$\begin{aligned} n \cdot c(n) &= O(n\log\log n) + 2n \cdot c\left(\sqrt{n\log^3 n}\right) \\ c(n) &= O(\log\log n) + 2c\left(\sqrt{n\log^3 n}\right) \\ &= O(\log n). \end{aligned}$$

Then $S(n) = O(n\log n)$ bits, so we require only $O(n)$ words, as desired.

3 Lower Bounds

Another result from Nekrich's 2021 paper is a data structure to answer four-dimensional orthogonal range reporting with $n\log^{O(1)} n$ space and $O(\log n/\log\log n + k)$ query time. This achieves a lower bound that was set by prior authors. In this section we will give a brief history of lower bounds for orthogonal range reporting and show some of the ideas used to prove lower bounds for this problem.

Chazelle made the first progress on lower bounds for the orthogonal range reporting problem with the following result.

Theorem 2. [5] A pointer machine that solves the problem in d dimensions with a query time of $O(k + P(\log n))$ where P is a polynomial must use $O(n(\log n/\log\log n)^{d-1})$ space.

We will go into more detail on Theorem 2 in Section 3.1. At a high level, the proof argues that if there is a large collection of possible queries whose answer sets overlap in at most one point, then the pointer machine must use a large amount of space to distinguish these possibilities. Then there is a construction that creates such "hard" sets of queries. Chazelle's result lower bounds the space usage when query time is assumed to be small, and in 2010 the same approach was adapted to lower bound the query time when the space usage is assumed to be small. The result is summarized in the following theorem.

Theorem 3. [1] Any data structure which solves the d-dimensional orthogonal range reporting problem with $S(n)$ space must have a query time of at least $O(Q(n) + k)$, where

$$Q(n) = \Omega((\log n/(\log(S(n)/n)))^{\lfloor d/2\rfloor - 1}).$$

When Theorem 3 is applied with $S(n) = n \log^{O(1)} n$, the lower bound on query time is $\Omega((\log n/\log\log n)^{\lfloor d/2 \rfloor - 1})$. The same authors also provided a data structure that can solve the three-dimensional range reporting problem with a query time of $O(\log n + k)$ using $O(n(\log n/\log\log n)^2)$ space. This is optimal by the aforementioned theorem. By generalizing inductively, their data structure can answer queries in $O(\log n(\log n/\log\log n)^{d-3} + k)$ time using $O(n(\log n/\log\log n)^{d-1})$ space.

In summary, if a data structure using $n \log^{O(1)} n$ space can optimally answer range reporting queries in $O(Q_d(n) + k)$ time, then the known bounds are

$$\Omega((\log n/\log\log n)^{\lfloor d/2 \rfloor - 1} + k) \leq Q_d(n) \leq O(\log n(\log n/\log\log n)^{d-3} + k).$$

Nekrich achieved the lower bound for $n = 4$ but the optimal query time for higher dimensions remains unknown.

3.1 The Pointer Machine Model

We outline some of the steps in the proof of Theorem 2, which overlap with those of Theorem 3. The bound is based on the *pointer machine* model, first introduced by Tarjan [7]. This machine consists of some memory and a finite number of registers, which are either data registers or pointer registers. The pointer machine can manipulate data and pointers through a limited set of instructions.

Chazelle's model of computation based on the pointer machine consists of a directed graph with vertex set V with source σ and edge set E. the conversion of a pointer machine to this graphical model consists of associating vertices to records and edges to the fields of those records. Movement through the graph corresponds to the pointer machine executing instructions. The pointer machine has a fixed number of fields per record, so in the graphical model we impose that the outdegree of a vertex is at most 2. Finally, there is a *freelist* of nodes with no data that the program can add to the graph (which correspond to the pointer machine creating new records).

For $v \in V$, let $N(v)$ denote the out-neighborhood of v. There is a set W of vertices, initially containing only σ. The program can execute the following instructions:

- Pick $v \in W$ and add $N(v)$ to W.
- Add a node v from the freelist and add it to W, with $N(v) = \varnothing$.
- Pick $v, w \in W$ and add the edge $v \to w$ if $|N(v)| < 2$.
- Pick $v, w \in W$ and remove the edge $v \to w$ if it exists.

The set W corresponds to states that the pointer machine could have visited. As applied to range reporting on n points, we assume that there is a labeling function L on the vertices with $0 \leq L(v) \leq n$. If $L(v) \neq 0$, then vertex v is associated with the point $p_{L(v)}$ (different vertices may share labels). For a query set q in the range reporting problem, let $W(q)$ be the set W at the termination of the program on input q. Then a necessary condition for the problem to be correctly answered is

$$\{i \mid p_i \in q\} \subseteq \{L(v) \mid v \in W\}.$$

This condition roughly says that in order to decide that a given p_i is in side the query range, the machine must reach a node which adds p_i to the answer. Query time in the pointer machine model corresponds to $W(q)$ (the number of explored states) in the graphical model.

To get a lower bound for this graphical computation model, we produce a set of queries such that no two queries share too many points. This forces the graph to have a large number of nodes and therefore the pointer machine to use a large amount of space. This is encapsulated in the following lemma:

Lemma 1. Let α, a, b be positive real numbers. Suppose that for any query q, the inequality

$$|W(q)| \leq a(|P \cap q| + \log^b n)$$

holds. Additionally suppose there exists a set of queries $|S|$ such that for each $q_i, q_j \in S$ with $i \neq j$, the following properties hold:

- $|P \cap q_i| \geq \log^\alpha n$ (the number of points in the query set is at least $\log^\alpha n$).
- $|P \cap q_i \cap q_j| \leq 1$ (any two queries share at most one point in common).

Then $|V| \geq |S|(\log^b n)/2^{16a+4}$.

With this lemma, it suffices to produce a set of points P and a set of queries S such that the conditions of the lemma hold. We will only cover the two-dimensional case in this paper and say that Chazelle generalizes to higher dimensions with a probabilistic construction based on the two-dimensional construction. For some integers m, λ, we let $n = m^\lambda$ and define $\rho_m(i)$ for $0 \leq i < n$ by

$$\rho_m(k_{\lambda-1}m^{\lambda-1} + \cdots + k_1 m + k_0) = k_0 m^{\lambda-1} + \cdots + k_{\lambda-2}m + k_{\lambda-1}$$

for $0 \leq k_i < m$, i.e. reverse the representation of i in base m with λ digits. The set of points P to be queried is

$$P = \{(i, \rho_m(i)) \mid 0 \leq i < n\}.$$

See Figure 5 for an example. Geometrically, this is the projection of a λ-dimensional lattice with m points on each side. This construction has the property that for any $0 \leq j < \lambda$ and $0 \leq k < m^\lambda$, the points

$$\{(i, \rho_m(i)) \mid \text{the representations of } i \text{ and } k \text{ in base } m \text{ agree except in the } m^j \text{ coefficient}\}$$

can be enclosed by a query rectangle (the range of the x coordinate is chosen to force the digits more significant than m^j to agree, and the range of the y coordinate is chosen to force the digits less significant than m^j to agree). This is a total of $\lambda m^{\lambda-1}$ queries because each such query is determined by a choice of j and a choice of the digits other than the m^j digit. The conditions of the lemma hold with suitable choices of m and λ, and we conclude from the lemma that $O(n \log n / \log\log n)$ space is required for polylogarithmic query time. This proves Lemma 1, which leads back to the lower bound on orthogonal range reporting that was mentioned previously.

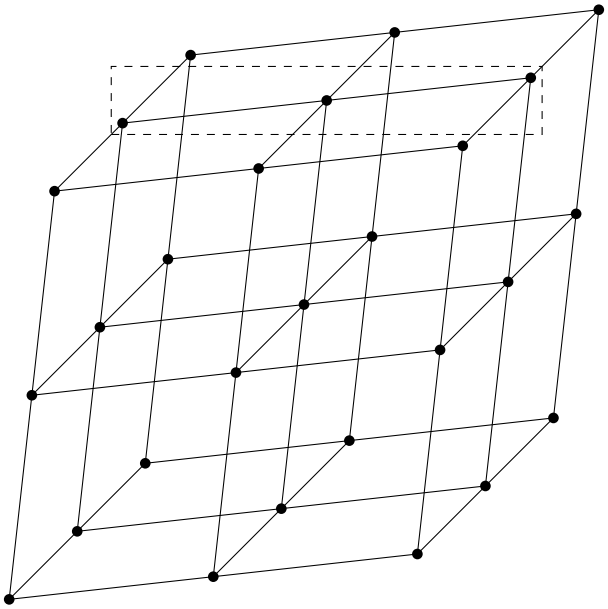


Figure 5: The construction for $m = \lambda = 3$. Each line segment of three points is a possible query.

References

[1] Afshani, P., Arge, L., and Larsen, K. D. Orthogonal range reporting. *Proceedings of the 2010 annual symposium on Computational geometry - SoCG 10* (2010).

[2] Chan, T. M. Persistent predecessor search and orthogonal point location on the word ram. *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms* (2013).

[3] Chan, T. M., Larsen, K. G., and Pătraşcu, M. Orthogonal range searching on the ram, revisited. *Proceedings of the 27th annual ACM symposium on Computational geometry - SoCG 11* (2011).

[4] Chazelle, B. A functional approach to data structures and its use in multidimensional searching. *SIAM Journal on Computing 17*, 3 (1988), 427–462.

[5] Chazelle, B. Lower bounds for orthogonal range searching: I. the reporting case. *Journal of the ACM 37*, 2 (1990), 200–212.

[6] Nekrich, Y. New data structures for orthogonal range reporting and range minima queries. *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)* (2021), 1191–1205.

[7] Tarjan, R. E. A class of algorithms which require nonlinear time to maintain disjoint sets. *Journal of Computer and System Sciences 18*, 2 (1979), 110–127.