On Top-k Range Reporting in 2D Space

Saladi Rahul
University of Minnesota
USA
sala0198@umn.edu

Yufei Tao
Chinese University of Hong Kong
Hong Kong
taoyf@cse.cuhk.edu.hk

ABSTRACT

Orthogonal range reporting (ORR) is a classic problem in computational geometry and databases, where the objective is to preprocess a set P of points in $\mathbb{R}^2$ such that, given an axis-parallel rectangle q, all the points in $P \cap Q$ can be reported efficiently. This paper studies a natural variant of the problem called *top-k ORR*, where each point $p \in P$ carries a *weight* $w(p) \in \mathbb{R}$. Besides q, a query also specifies an integer $k \in [1, |P|]$, and needs to report the k points in $q \cap P$ with the largest weights. We present optimal or near-optimal structures for solving the top-k ORR problem in the pointer machine and external memory models. As a side product, our structures give new space-query tradeoff for the *orthogonal range max* problem, which is a special case of top-k ORR with $k = 1$.

Categories and Subject Descriptors

F.2.2 [**Analysis of algorithms and problem complexity**]: Nonnumerical Algorithms and Problems—*computations on discrete structures*; H.3.1 [**Information storage and retrieval**]: Content analysis and indexing—*indexing methods*

Keywords

Top-k; Range Reporting; Approximate Weight Threshold

1. INTRODUCTION

In the *orthogonal range reporting* (ORR) problem, we want to preprocess a set P of points in $\mathbb{R}^2$ into a structure such that, given an axis-parallel rectangle q, all the points in $P \cap q$ can be reported efficiently. This is a classic problem in computational geometry that has been very well understood.

In this work, we study a natural variant of the problem called *top-k orthogonal range reporting* (or top-k ORR) which is defined as follows. Each point $p \in P$ is associated with a distinct *weight* $w(p) \in \mathbb{R}$. Besides q, a query also specifies an integer $k \in [1, n]$ where $n = |P|$. The query returns the k points in $q \cap P$ with the largest weights (see Figure 1).[1] The objective, as before, is to store P in a structure of small size to answer all queries efficiently.

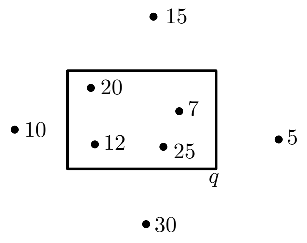


Figure 1: The number beside each point indicates its weight. A top-k ORR query with $k = 2$ and a search region q as shown returns the points with weights 20 and 25.

As an interesting special case, top-1 ORR (i.e., fixing $k = 1$) is also known as the *orthogonal range max* problem.

Pratical Motivation. Top-k ORR is important in applications where a user is interested in only the *best few* objects in terms of weights, as opposed to all the objects in a query region. As a representative example in spatial databases, consider each point as the location of a hotel, with the point's weight corresponding to the restaurant's rating. A top-10 ORR query that would be frequently issued at a website like *hotel.com* is "find the 10 best rated hotel in the Manhattan area". It is not hard to see plenty of other applications of this sort in various domains. In particular, top-k ORR serves as a top-k extension of queries of the following form in relational databases: `select max(`A_1`) from table where` $A_2 \in [x_1, x_2]$ `and` $A_3 \in [y_1, y_2]$.

Computation Models. We are interested in structures of both internal and external memory. For internal memory, our structure works under the pointer machine model. Depending on the types of CPU calculation allowed, Chazelle [9] refined the model into several variants. Unless otherwise stated, by "pointer machine", we refer to the most primitive of his variants—called *elementary pointer machine* (EPM)—which allows only comparisons and +.

For external memory, we resort to the standard model defined by Aggarwal and Vitter [6]. In this model, a machine

[1]That objects have distinct weights is a standard assumption in the previous top-k studies [4, 21]. Otherwise, what would be a top-k result becomes ambiguous (just imagine the extreme case where all objects have the same weight).

problem	space	query	source	model	remark
2-sided top-k ORR	$O(n)$	$O(\log n + k)$	**new**	EPM	optimal
	$O(n/B)$	$O(\log_B n + k/B)$	**new**	EM	optimal
3-sided top-k ORR	$O(n\frac{\log n}{\log\log n})$	$O(\log n + k)$	**new**	EPM	optimal
	$O(\frac{n}{B}\frac{\log n}{\log\log_B n})$	$O(\log_B n + k/B)$	**new**	EM	optimal
4-sided top-k ORR	$O(n\log^2 n)$	$O(\log^2 n + k)$	[18]	EPM	
	$O(n\log^2 n)$	$O(\log n + k\log\log n)$	[18]	EPM	ordered reporting
	$O(n\frac{\log n}{\log\log n})$	$O(\log n + k)$	**new**	EPM	optimal
	$O(\frac{n}{B}\frac{\log n\cdot(\log\log B)^2}{\log\log_B n})$	$O(\log_B n + k/B)$	**new**	EM	
	$O(n/B)$	$O(\sqrt{n/B} + k/B)$	**new**	EM	non-replicate, optimal
orthogonal range max	$O(n)$	$O(\log^3 n)$	[9]	EPM	
	$O(n)$	$O(\log^2 n)$	[9]	APM	
	$O(n\log^2 n)$	$O(\log n)$	folklore	EPM	
	$O(n\frac{\log n}{\log\log n})$	$O(\log n)$	**new**	EPM	
	$O(n/B)$	$O(\log_B^2 n)$	[5]	EM	
	$O(\frac{n}{B}\log n)$	$O(\log_B n)$	[20]	EM	
	$O(\frac{n}{B}\frac{\log n}{\log\log_B n})$	$O(\log_B n)$	**new**	EM	

Table 1: Comparison of our and previous results (EPM = elementary pointer machine, APM = arithmetic pointer machine, and EM = external memory)

has memory of M words, and a disk that has been formatted into *blocks* of size B words (it always holds that $M \geq 2B$). An I/O exchanges a block of data between the disk and the memory. The space of a structure is the number of blocks occupied, whereas the cost of an algorithm is the number of I/Os performed. CPU calculation is free.

1.1 Previous Results

In this subsection, we will review the existing structures on the top-k ORR problem and the orthogonal range max problem. Focus will be placed on the pointer-machine and external memory models, but we will also briefly mention the state-of-the-art relevant results in the RAM (random access machine) model for the two problems.

Top-k ORR. This problem, in spite of its practical importance, has not been extensively studied. On pointer machines, the only results we are aware of are due to Rahul et al. [18]. They propose two structures, both consuming $O(n\log^2 n)$ space[2], but answering a query in $O(\log^2 n + k)$ and $O(\log n + k\log\log n)$ time, respectively. On a RAM, Navarro and Nekrich [15] gave a structure of $O(n)$ space and $O(\log^{1+\epsilon} n + k\log^{\epsilon} n)$ query time.

On the other hand, one-dimensional top-k ORR—where the points of P are on a line and a query region q is an interval—has received more attention. On pointer machines, combining Frederickson's heap selection algorithm [11] with a priority search tree [14] gives a structure of $O(n)$ size and $O(\log n + k)$ query time. In external memory, Afshani et al. [4] presented a structure of $O(n/B)$ space that answers a query in $O(\log_B n + k/B)$ I/Os. In the scenario where the n points of P are in the range $[1, O(n)]$, Brodal et al. [8] developed an optimal RAM structure of $O(n)$ space and $O(k)$ query time. See also the work of Karpinski and Nekrich [13] for a colored version of 1d top-k ORR.

In this work, the k points in the result of a top-k query can be reported in an arbitrary order. In the *ordered* version of the problem, the result points must be output in ascending order of weight. In 2D space, the $O(\log n + k\log\log n)$ query time structure of [18] and the structure of [15] (both mentioned earlier) were actually designed for the ordered version. See [4, 8, 13] for the corresponding results in 1D space. Furthermore, while the current paper assumes that the input set is static, update-efficient top-k structures have also appeared in the literature recently, e.g., [19, 23].

Orthogonal Range Max. For this problem, Chazelle [9] developed a pointer-machine structure of $O(n)$ size and $O(\log^3 n)$ query time. In a more powerful model—*arithmetic pointer machine* (APM) which allows comparisons, $+$, $-$, $\times$, $\div$, and bit-shifting—he [9] also gave an $O(n)$-size structure with $O(\log^2 n)$ query time. In the same paper, Chazellel [9] also gave several RAM structures with different space-query tradeoffs.

When the search region q is *2-sided*—namely, in the form $(-\infty, x] \times (-\infty, y]$—orthogonal range max can be reduced to the *point location* problem, which can be solved by a pointer machine structure of $O(n)$ size and $O(\log n)$ query time. Using range-tree ideas, this leads to a pointer machine structure of $O(n\log^2 n)$ size and $O(\log n)$ query time for general (i.e., *4-sided*) queries.

In external memory, Agarwal et al. [5] presented a structure of $O(n/B)$ space and $O(\log_B^2 n)$ query cost. When the query region is *3-sided*—in the form $[x_1, x_2] \times (-\infty, y]$—Sheng and Tao [20] obtained a structure of $O(n/B)$ space and $O(\log_B n)$ query cost. This gives rise to a structure of $O(\frac{n}{B}\log n)$ space that answers a 4-sided query in $O(\log_B n)$ I/Os.

1.2 Our Contributions

Results. Table 1 compares our main results to the relevant existing results. It is worth mentioning that our pointer machine structure for the orthogonal range max problem is the first achieving a space-query-time product of $o(n\log^2 n)$.

New Technique: Δ-AWT. A common approach [4, 21] to answer a top-k ORR query is to find a threshold σ, such that at least k but at most $O(k)$ points $p \in P$ (i) fall in the query region q, and (ii) have weights $w(p) \geq \sigma$. After this, one can

[2]All logarithms have base 2 by default.

retrieve all the points satisfying conditions (i) and (ii) using a 5-sided[3] orthogonal range query in the 3D space of x, y, and weight. The 5-sided range query can report only $O(k)$ points, and incurs $O(\log n)$ additional time by resorting to a structure of [3]. Finally, the top-k points can be found from those $O(k)$ points with k-selection.

In this work, we extend the above methodology, and propose to look at a new type of queries:

> Given an axis-parallel rectangle q and an integer k, a Δ-*approximate weight threshold* (Δ-AWT) query returns a value σ such that at least k but at most $O(k+\Delta)$ points $p \in P$ satisfy: $p \in q$, and $w(p) \geq \sigma$ (if $|P \cap q| < k$, then the query should return $-\infty$). Here, Δ is an integer in $[1, n]$ that is fixed for all queries.

Note that σ does *not* need to be the weight of any point in P.

To see the relevance of Δ-AWT to top-k ORR, fix $\Delta = \log n$. Given a top-k ORR query, we first answer a Δ-AWT query with the same q and k. Let σ be the Δ-AWT output. Proceed with a 5-sided query as described in the conventional approach. This 5-sided query returns at most $O(k+\Delta) = O(\log n + k)$ points, which has no impact on the overall running time $O(\log n + k)$.

Δ-AWT turns out to be an interesting problem in itself. When $\Delta = \omega(1)$, it may be possible to use structures of *sublinear* sizes to process Δ-AWT queries efficiently. In particular, when q is 2-sided, a Δ-AWT query can be answered in $O(\log(n/\Delta))$ time using a pointer machine structure of $O(n/\Delta)$ space—both the space and query costs are optimal, as we will prove in the next section. It is such space savings that allow us to obtain neat 3- and 4-sided Δ-AWT structures that pave the way to solving top-k ORR.

2. 2-SIDED Δ-AWT

This section serves as a proof for our first main result:

THEOREM 1. *The following statements are true:*

1. *Any structure correctly solving 2-sided Δ-AWT queries must store $\Omega(n/\Delta)$ words in the worst case. Restricting to the pointer machine model, any structure must incur $\Omega(\log(n/\Delta))$ query time in the worst case.*
2. *There is a pointer machine structure of $O(n/\Delta)$ size that answers a 2-sided Δ-AWT query in $O(\log(n/\Delta))$ time.*

Lower Bounds. To prove Statement 1 of the above theorem, we consider that a machine word has t bits, and that the weight domain is the integer range $[0, 2^t - 1]$. Let c be a constant such that, given a Δ-AWT query with parameters q and k, the structure returns a value σ ensuring at least k but at most $c(k+\Delta)$ points $p \in P$ satisfy $p \in q$ and $w(p) \geq \sigma$, provided that $|P \cap q| \geq k$. Define $\lambda = c(1+\Delta)+1$. We will assume that n is multiple of λ, that 2^t is a multiple of both n and n/λ, and that $2^t \geq n^2$.

It suffices to look at a set P of one-dimensional points, where the i-th ($i \in [1, n]$) point p_i has coordinate i. To decide the weights of these points, we divide P into n/λ *groups* of equal size: $P_1, ..., P_{n/\lambda}$, where P_j ($j \in [1, n/\lambda]$) consists of the points from coordinate $(j-1)\lambda + 1$ to $j\lambda$. Likewise, divide the weight domain into n/λ *chunks* of equal size $s = 2^t\lambda/n$, such that the j-th chunk ($j \in [1, n/\lambda]$) is the range $[(j-1)s, js-1]$.

The weights of the points in each group are generated independently. To describe the generation for group P_j (for any $j \in [1, n/\lambda]$), let $\pi_1, ..., \pi_\lambda$ be the points of P_j in ascending order, and $v_1, ..., v_s$ the values of chunk j in descending order. Then:

- The weight $w(\pi_1)$ of π_1 is picked from $\{v_{z\lambda} \mid z = 1, 2, ..., s/\lambda\}$ uniformly at random. Define $X_j = w(\pi_1)$ for the purpose of our analysis later.
- Then, for each $z' = 2, ..., \lambda$, the weight $w(\pi_{z'})$ of $\pi_{z'}$ is set to $w(\pi_1) - (z'-1)$.

The above construction endows P with two properties:

- For any $j_1 < j_2$, the points of group P_{j_1} have weights strictly smaller than those of group P_{j_2}.
- In group P_j (of any j), the points have weights $X_j, X_j - 1, ..., X_j - (\lambda - 1)$, respectively, with X_j being a multiple of λ.

Now consider a one-dimensional Δ-AWT query with $q = (-\infty, j\lambda]$ (note that this is a degenerated 2-sided rectangle) and $k = 1$, where j is an integer from 1 to n/λ. The structure must return a value $\sigma_j \in (X_j - (\lambda-1), X_j]$ due to the reasons below:

- P has a point (in group j) that is covered by q, and has weight X_j. Hence, no value higher than X_j can be returned.
- If $\sigma_j \leq X_j - (\lambda - 1)$, then all the points of P_j are covered by q, and have weights at least σ_j. This violates the requirement Δ-AWT because $|P_j| = \lambda > c(1+\Delta)$.

It thus follows that, the query algorithm can decide precisely the value of X_j as the nearest multiple of λ that is at least σ_j.

Therefore, the structure serves as an encoding of random variables $(X_1, X_2, ..., X_{n/\lambda})$. Note that each X_j ($j \in [1, n/\lambda]$) has an entropy of $\log_2(s/\lambda)$ bits. Due the independence of the n/λ variables, we know that the structure must contain $(n/\lambda)\log_2(s/\lambda) = (n/\lambda)\log_2(2^t/n)$ bits in the worst case. When $2^t \geq n^2$, this is at least $\Omega((n/\lambda)\log(2^t))$ bits, namely, $\Omega(n/\Delta)$ words.

The query lower bound in Statement 1 can be established by the following argument. First recall that any pointer machine structure can be modeled as a directed graph where each node stores $O(1)$ words. Consider running the n/λ queries as mentioned earlier, one for each $j \in [1, n/\lambda]$. Remove from the structure all nodes that are never touched by any of the queries. From the space lower bound, we know that at least $\Omega(n/\Delta)$ nodes remain. Any query algorithm must start from a unique node (called the *root*) of the structure, and at each node, can choose from any of its out-neighbors as the next hop. Since each node has $O(1)$ out-neighbors, we know that at least one node requires $\Omega(\log(n/\Delta))$ hops from the root.

Remark. Note that the $\Omega(n/\Delta)$ space lower bound is not restricted to pointer machine structures. It holds for any

[3]A 5-sided rectangle in 3D space has the form $[x_1, x_2] \times [y_1, y_2] \times [z, \infty)$.

RAM/EM structure. Furthermore, our information theoretic argument essentially has shown that randomization does not help: any structure must use $\Omega(n/\lambda)$ space in expectation.

Upper Bounds. Inspired by [4], we now describe a pointer machine structure matching the aforementioned lower bounds, assuming without loss of generality that n/Δ is a power of 2. First, convert each point $p = (x, y)$ in P to a 3D point (x, y, z) where $z = w(p)$. Let P' be the set of 3D points thus obtained. In general, given two 3D points $p_1 = (x_1, y_1, z_1)$ and $p_2 = (x_2, y_2, z_2)$, we say that p_1 *dominates* p_2 if $x_1 \le x_2$, $y_1 \le y_2$, $z_1 \le z_2$, and at least one equality does not hold. Also, we say that p_1 is *lower* than p_2 if $z_1 < z_2$.

Let S be a set of points in $\mathbb{R}^3$. We say that S is a *shallow τ-cutting* of P' if it has the following properties:

P1: S has $O(n/\tau)$ points.

P2: Each point in S dominates $O(\tau)$ points in P'.

P3: Any point p (which is not necessarily in P') dominating at most τ points in P' is dominated by at least a point in S.

As shown in [1], such an S exists for any $\tau \in [1, n]$.

Let $h = \log_2(n/\Delta)$. We obtain sets $S_1, ..., S_{h+1}$ where S_i ($i \in [1, h+1]$) is a shallow $(2^{i-1}\Delta)$-cutting of P'. On each S_i, we create a structure to answer *probing queries* of the form: given a 2D point $\pi = (x, y)$, find the lowest point $p \in S_i$ such that $x(p) \le x$ and $y(p) \le y$, where $x(p)$ and $y(p)$ are the x- and y-coordinates of p, respectively. Note that this is essentially a 2-sided orthogonal range max query, and thus, can be answered in logarithmic time by a linear size structure (see the literature review in Section 1). Since the sizes of $S_1, ..., S_{h+1}$ decrease geometrically, the total space of all these structures is dominated by the one on S_1, namely, $O(n/\Delta)$ by Property ***P1***.

To answer a Δ-AWT query with parameters $q = [x, \infty) \times [y, \infty)$ and k, we check whether $k < \Delta$. If so, we manually increase k to Δ before proceeding. When $k \ge \Delta$, we first determine the smallest i such that $2^i\Delta \ge k$. Then, perform a probing query on S_{i+1} with $\pi = (x, y)$. Let p be the point returned by this probing query. We return $z(p)$ (i.e., the z coordinate of p) as the answer for the Δ-AWT query.

Lemma 1. *The probing query definitely returns a point p. Furthermore, $z(p)$ is a correct answer for the Δ-AWT query.*

Proof. Let z be a value such that $p' = (x, y, z)$ dominates exactly k points in P'; if z does not exist, we set $z = -\infty$. In any case, p' dominates at most $k \le 2^i\Delta$ points in P'. Hence, by Property ***P3***, we know that S_{i+1} definitely contains a point dominating p', implying that the probing query cannot return an empty result. Furthermore, it also implies that p (that output of the probing query) definitely dominates p'.

Next, we prove the second part of the lemma focusing on the scenario where $k \ge \Delta$. First, if $z = -\infty$, it follows that $|P \cap q| < k$ and that $z(p)$ must be $-\infty$. Hence, our algorithm correctly returns $-\infty$.

The subsequent discussion considers $z \ne -\infty$ (and hence, p' dominates k points in P'). As $z(p) \le z$, point $(x, y, z(p))$ must dominate at least k points of P'. On the other hand,

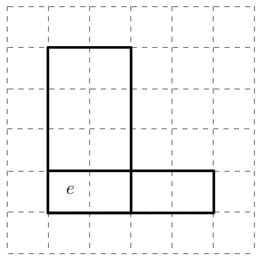


Figure 2: Two dyadic rectangles of e in solid edges

$(x, y, z(p))$ cannot dominate more points of P' than p does (because $x(p) \le x$ and $y(p) \le y$). Property ***P2*** guarantees that p dominates $O(2^i\Delta) = O(2k) = O(k + \Delta)$ points of P'. This completes the proof. $\square$

At first glance, the above query algorithm seems to require division to determine the smallest i such that $2^i\Delta \ge k$, contradicting our goal of designing a structure on an elementary pointer machine. This can be fixed by creating a binary search tree on the $h+1$ values Δ, 2Δ, $2^2\Delta$, ... $2^h\Delta$ so that at query time we can find i by finding the successor of k in this tree using $O(\log h)$ time. In this way, the query algorithm relies on only comparisons. The overall query cost is dominated by the time of a probing query, which is $O(\log(n/\Delta))$. We now conclude the proof of Theorem 1.

3. 4-SIDED Δ-AWT AND TOP-K ORR

In this section, we will present a structure for solving 4-sided Δ-AWT queries, and then, explain how the structure can be utilized to settle top-k ORR. Let us start with an easy lemma:

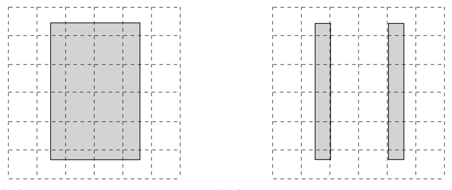
Lemma 2. *There is a pointer machine structure of $O(\frac{n}{\Delta}\log\frac{n}{\Delta})$ size that answers a 3-sided Δ-AWT query in $O(\log(n/\Delta))$ time. There is also a pointer machine structure of $O(\frac{n}{\Delta}\log^2\frac{n}{\Delta})$ size that answers a 4-sided Δ-AWT query in $O(\log(n/\Delta))$ time.*

Proof. By standard range-tree ideas. $\square$

The 4-sided structure in the above lemma is not powerful enough for obtaining our final result on top-k ORR. Next, we improve it by reducing its space to $O(\frac{n}{\Delta}\log^2\frac{n}{\Delta}/\log\log\frac{n}{\Delta})$. We achieve this by combining the standard approach of using a logarithmic-fanout range tree with new ideas based on dyadic rectangles.

Structure. Without loss of generality, we assume that n is a multiple of Δ. Divide the data space into n/Δ vertical slabs such that each slab covers exactly Δ points of P. Set $f = \log(n/\Delta)$. Create an f-ary tree T on these slabs, each of which corresponds to a leaf node in T. Given a node u in T, we will use *slab*(u) to denote the slab of u (if u is an internal node, *slab*(u) is the union of the slabs of its children). Define $P_u = P \cap slab(u)$, and $n_u = |P_u|$.

We associate each internal node u with several secondary structures. Let $m = \Delta\log^3(n/\Delta)$. Impose an $f \times \frac{n_u}{m}$ grid G_u on P_u, such that each column of G_u is the slab of a child of u, whereas each row covers $\Theta(m)$ points in P_u. For each column, create a *column structure* which is a 3-sided structure of Lemma 2 on the points of P_u in the column. For each row, create a *row structure* which which is a 4-sided structure of Lemma 2 on the points of P_u in the row.


(a) Original query (b) Vertical 3-sided queries

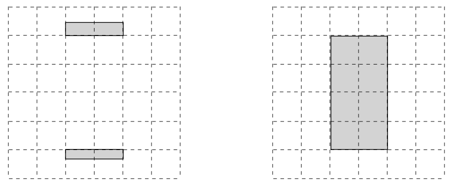

(c) Horizontal 3-sided queries (d) Grid query

Figure 3: Partitioning a query rectangle

Now consider a cell e in G_u. Let r be a rectangle whose edges are aligned with the column and row boundaries of G_u. In other words, r is tiled by a set of cells in G_u. Let a (b) be the number of columns (rows) of G_u that r covers. We say that r is a *dyadic* $a \times b$ *rectangle of* e if it satisfies two conditions:

- Both a and b are powers of 2.
- e is covered by r, and is located at one of the 4 corners of r.

See Figure 2 for an example. Note that e has $O(\log f \cdot \log(n_u/m)) = O(\log\log(n/\Delta) \cdot \log(n/\Delta))$ dyadic rectangles. For each such rectangle r, we store a *sketch*, which consists of the Δ-th largest, (2Δ)-th largest, $(2^2\Delta)$-th largest ... of the weights of the points in $r \cap P$. In other words, e is associated with $O(\log\log(n/\Delta) \cdot \log(n/\Delta))$ sketches, each of size $O(\log(n/\Delta))$.

Overall, the column structures of u occupy $O((n_u/\Delta)\log(n_u/\Delta))$ space, and its row structures occupy $O(\frac{n_u}{m}\frac{m}{\Delta}\log^2\frac{m}{\Delta}) = O(\frac{n_u}{\Delta}(\log\log\frac{n_u}{\Delta})^2)$ space. G_u has fn_u/m cells, each of which needs $O(\log^2\frac{n}{\Delta}\log\log\frac{n}{\Delta})$ space for its sketches. Hence, all the sketches of all cells in G_u occupy in total

$$O\left(\log\frac{n}{\Delta} \cdot \frac{n_u}{\Delta\log^3(n/\Delta)} \cdot \log^2\frac{n}{\Delta}\log\log\frac{n}{\Delta}\right)$$
$$= \quad O\left(\frac{n_u}{\Delta}\log\log\frac{n}{\Delta}\right)$$

space. In summary, all the secondary structures at each level of T use $O((n/\Delta)\log(n/\Delta))$ space. T has $O(\log\frac{n}{\Delta}/\log\log\frac{n}{\Delta})$ levels, and thus, needs $O(\frac{n}{\Delta}\log^2\frac{n}{\Delta}/\log\log\frac{n}{\Delta})$ space.

Query. We now explain how to answer a 4-sided AWT query with parameters q and $k \geq \Delta$ (if $k < \Delta$, simply increase k to Δ before proceeding). If q completely falls within the slab of a leaf node in T, we finish by returning $-\infty$. Otherwise, we use the following algorithm to answer q at the highest node u at which q intersects at least two columns in G_u. Note that u can be found in $O(\log(n/\Delta))$ time.

If q is completely contained in a row of G_u, we answer it using the corresponding row structure in $O(\log m) = O(\log\log(n/\Delta))$ time. Otherwise, q can be divided into two *vertical 3-sided* queries, two *horizontal 3-sided* queries, and a *grid query*, as illustrated in Figure 3. For each resulting query—let q' be its search rectangle—we compute the outcome of the Δ-AWT query with parameters q' and k on P. Specifically, let σ_{v1}, σ_{v2} be the outcomes of the two vertical 3-sided queries, σ_{h1}, σ_{h2} be the outcomes of the two horizontal 3-sided queries, and σ_g be the outcome of the grid query. We return $\sigma = \max\{\sigma_{v1}, \sigma_{v2}, \sigma_{h1}, \sigma_{h2}, \sigma_g\}$ as the final answer.

It is easy to see that σ_{v1}, σ_{v2} can be obtained using two column structures at u in $O(\log\frac{n_u/f}{\Delta}) = O(\log(n/\Delta))$ time, and σ_{h1}, σ_{h2} can be obtained using two row structures at u in $O(\log(m/\Delta)) = O(\log\log(n/\Delta)$ time. Next, we explain how to compute σ_g.

Denote by g the search rectangle of the grid query. Suppose that g spans a columns and b rows of G_u. Denote by a' and b' the largest powers of 2 at most a and b, respectively. Let e_1, e_2, e_3, and e_4 be the top-left, top-right, bottom-left, and bottom-right corners of g, respectively. Let r_1 be the dyadic $a' \times b'$ rectangle of e_1, having e_1 at its top-left corner. Similarly, let r_2, r_3, r_4 be the dyadic $a' \times b'$ rectangles having e_2, e_3, e_4 at their top-right, bottom-left, and bottom-right corners, respectively. Note that r_1, r_2, r_3 and r_4 may overlap, as is exemplified in Figure 4 using r_1 and r_4.

Let i be the lowest integer such that $2^i\Delta \geq k$. We set σ_{g1} to the i-th largest weight in the sketch of r_1, if the sketch has size at least i; otherwise, $\sigma_{g1} = -\infty$. Likewise, let σ_{g2}, σ_{g3}, and σ_{g4} be decided in the same manner from r_2, r_3, and r_4, respectively. Then, σ_g is determined as $\max\{\sigma_{g1}, \sigma_{g2}, \sigma_{g3}, \sigma_{g4}\}$.

To implement the above algorithm in $O(\log(n/\Delta))$ time on an (elementary) pointer machine, the only technicality worth mentioning is to derive a' and b' from a and b, respectively, on an elementary pointer machine. The crucial observation is that both a and b can distribute only inside the range $[1, n/\Delta]$. Hence, we can index with a binary search tree all the powers of 2 within that range. Then, a' (b') is simply the predecessor of a (b) in this tree.

LEMMA 3. *The answer σ thus computed is correct.*

PROOF. It suffices to show that σ_g is a correct answer for the Δ-AWT query with parameters g and k. First, if $\sigma_g = -\infty$, it means that $\sigma_{g1} = \sigma_{g2} = \sigma_{g3} = \sigma_{g4} = -\infty$. Thus, g can cover at most $4k$ points of P, in which case $\sigma_g = -\infty$ is correct.

Consider that $\sigma_g \neq -\infty$. Suppose without loss of generality that $\sigma_g = \sigma_{g1}$. By the definition of σ_{g1}, at least k points in $P \cap r_1$ have weights no less than σ_{g1}. Thus, at least k points in g have weights no less than σ_g.

Let c_i ($i = 1, 2, ..., 4$) be the number of points in $P \cap r_i$ having weights at least σ_{gi}. We know that $c_i = O(k)$. Let c be the number of points in $P \cap g$ having weights at least σ_g. If a point is not counted by any of $c_1, ..., c_4$, it cannot be counted by c either. Hence, $c \leq \sum_{i=1}^{4} c_i = O(k)$. □

The above discussion has established:

LEMMA 4. *There is a pointer machine structure of $O(\frac{n}{\Delta}\log^2\frac{n}{\Delta}/\log\log\frac{n}{\Delta})$ size that answers a 4-sided Δ-AWT query in $O(\log(n/\Delta))$ time.*

Top-k ORR. We now proceed to discuss top-k ORR queries. If queries are 2-sided, we can answer them in

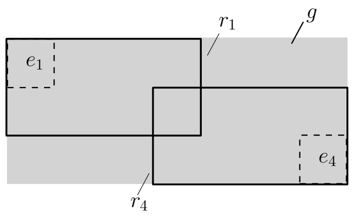


Figure 4: Covering a grid query with dyadic rectangles

$O(\log n+k)$ time using a structure of $O(n)$ size. For this purpose, we build a structure $\mathcal{T}$ of Theorem 1 with $\Delta = \log n$ on P. Given a top-k ORR query with a 2-sided rectangle q, we first perform Δ-AWT search on $\mathcal{T}$ to obtain a threshold σ. Then, all the points in $P \cap q$ with weights at least σ can be obtained in $O(\log n + \Delta + k) = O(\log n + k)$ time using a 3D dominance structure [1] which uses $O(n)$ space.

For 4-sided queries, a similar approach gives a structure of $O(n \log n/ \log\log n)$ space and $O(\log n + k)$ query time. Specifically, we create a structure $\mathcal{T}$ of Lemma 4 with $\Delta = \log n$. Note that $\mathcal{T}$ consumes $O(n \log n/ \log\log n)$ space. Given a top-k ORR query with parameters q and k, we perform Δ-AWT search on $\mathcal{T}$ to obtain a threshold σ. Then, all the points in $P \cap q$ with weights at least σ can be obtained in $O(\log n + k)$ time using a 3D 5-sided range reporting structure [3] which uses $O(n \log n/ \log\log n)$ space.

One may wonder what happens to 3-sided top-k ORR. Interestingly, there is no hope to improve upon the bounds obtained above for 4-sided queries. In general, for 3-sided top-k ORR, it is impossible to use $o(n \log n/ \log\log n)$ space if one wishes for a query time of $O(\log^c n + k)$ for any constant c. This can be proved with the following argument. Consider the so-called $Q(3,1)$ *range reporting* problem as defined in [2]: let P' be a set of n points in $\mathbb{R}^3$; we want to store P' in a structure such that, given a rectangle $q = [x_1, x_2] \times [y, \infty) \times [z, \infty)$, we can report all the points in $P' \cap q$ efficiently. It is known (from a lower bound in [10]) that every structure with $O(\log^c n + k')$ query time (where $k' = |P' \cap q|$) must use $\Omega(n \log n/ \log\log n)$ space. However, as we show below, any 3-sided top-k ORR structure with query time $O(\log^c n + k)$ can be deployed to answer a $Q(3,1)$ query in $O(\log^c n + k')$ time. This means that the 3-sided top-k ORR structure must use $\Omega(n \log n/ \log\log n)$ space.

Now we explain a reduction from $Q(3,1)$ to 3-sided top-k ORR. Let us regard each point $p' = (x, y, z)$ in P' as a 2D point $p = (x, y)$ with weight $w(p) = z$. Let P be the set of 2D points thus obtained. Store P in a 3-sided top-k ORR structure $\mathcal{T}$. Given a $Q(3,1)$ query $q = [x_1, x_2] \times [y, \infty) \times [z, \infty)$, we answer it with a series of 3-sided top-k ORR queries on $\mathcal{T}$ as follows. The search regions of all these queries are fixed to $[x_1, x_2] \times [y, \infty)$. The i-th ($i \geq 1$) query is issued with k set to $2^{i-1} \log^c n$. Let S_i be the set of points fetched by this query. Define z_i as the $(2^{i-1} \log^c n)$-th largest weight of the points in S_i if $|S_i| \geq 2^{i-1} \log^c n$, or set $z_i = -\infty$ otherwise. Report all the points in S_i whose weights are between $[\max\{z, z_i\}, z_{i-1}]$ (let $z_0 = \infty$ for the boundary case $i = 1$). The algorithm stops as soon as z_i falls below z.

It is clear that the above algorithm correctly reports all the points in $q \cap P'$. Next, we argue that the query cost is $O(\log n + k')$ where $k' = |q \cap P'|$. Define $k_i = 2^{i-1} \log^c n$ for $i \geq 1$, and $k_0 = 0$. Let ρ be the total number of 3-sided top-k ORR queries issued. For each $i \in [1, \rho - 1]$, the i-th query reports $k_i - k_{i-1} \geq k_i/2$ points, and incurs cost $O(\log^c n + k_i)$ which is $O(k_i)$ because all of k_1, ..., $k_{\rho-1}$ must be at least $\log^c n$. We can therefore amortize the time onto the $k_i/2$ points reported, so that each point gets charged only $O(1)$ cost. Regarding the ρ-th query, observe that k_ρ is at most $2k'$ if $\rho \geq 2$ (because $k_{\rho-1}$ is at most k'). If $\rho = 1$, then $k_\rho = \log^c n$. In any case, the time of the ρ-th query is bounded by $O(\log^c n + k_\rho) = O(\log^c n + k')$.

THEOREM 2. *The following statements are true:*

1. *There is a pointer machine structure of $O(n)$ size that answers a 2-sided top-k ORR query in $O(\log n + k)$ time.*
2. *For any constant c, a pointer machine structure that answers a 3-sided top-k ORR query in $O(\log^c n + k)$ time must use $\Omega(n \log n/ \log\log n)$ space.*
3. *There is a pointer machine structure of $O(n \log n/ \log\log n)$ size that answers a 4-sided top-k ORR query in $O(\log n + k)$ time.*

4. TOP-K ORR IN EXTERNAL MEMORY

In this section, we adapt the pointer machine results of the previous section to external memory. Our main technical novelty can be summarized as:

- Proof of an $\Omega(\log_B(n/\Delta))$ lower bound for Δ-AWT queries. Recall that the query lower bound in Theorem 1 applies only to pointer machines. However, an EM structure does not need to be pointer-based; instead, an algorithm can choose at its free will to access any block of the structure—a piece of luxury that a pointer machine structure does not have. This discrepancy prevents the pointer-machine's query cost argument from being applicable in EM.
- Working with Δ-AWT structures that store just as many words as the ones on pointer machines.
- Obtain a space-economical structure that can answer a 5-sided range reporting query in 3D space using $O(\log_B n + k/B)$ I/Os.

4.1 Query Lower Bound on 2-Sided Δ-AWT

The first statement of Theorem 1 implies that any EM structure solving the 2-sided Δ-AWT problem must occupy $\Omega(n/(B\Delta))$ blocks. In this subsection, we prove:

THEOREM 3. *For any $\Delta \in [1, n]$, an external memory structure of $O(n/(B\Delta))$ space must incur $\Omega(\log_B(n/\Delta))$ I/Os answering a 2-sided Δ-AWT query in the worst case.*

We will give a reduction from the *predecessor search* problem to 2-sided Δ-AWT with an interesting use of an upper bound result of [7] on *one-dimensional range reporting.*

Let t be the number of bits in a word. We consider that the input of predecessor search is a set S of n integers in $[0, \frac{2^t-1}{\Delta}]$; given an integer $x \in [0, \frac{2^t-1}{\Delta}]$, a query returns the predecessor of x in S, namely, $\max\{v \in S \mid v \leq x\}$. Without loss of generality, let us assume that 0 always belongs to S, so that the predecessor of x definitely exists. According to a result of Patrascu and Thorup [17], when $B \geq t$ and 2^t is sufficiently large compared to n, any structure using $O(n/B)$ space must incur $\Omega(\log_B n)$ I/Os answering a predecessor query in the worst case.

In 1D range reporting, the goal is to preprocess the set S aforementioned so that, given a range I inside $[0, 2^t - 1]$, a query can report all the elements in $S \cap I$ efficiently. In [7], Alstrup et al. described a structure of $O(n/B)$ space that answers such a query in $O(1+k'/B)$ I/Os, where $k' = |S \cap I|$.

At a high level, our reduction from predecessor search to 2-sided Δ-AWT works as follows. Given a predecessor query with parameter x, we will somehow perform a Δ-AWT query to obtain an interval $I = [x', x]$ with the guarantee that $|S \cap I| \in [1, O(1)]$. Then, we can retrieve all the elements of $S \cap I$ in $O(1)$ I/Os, after which the predecessor of x can be obtained from only these elements. Next, we describe the details.

Let $v_1, ..., v_n$ be the integers of S in ascending order. Define $S' = \{v_i\Delta \mid i \in [1, n]\}$. Note that each element of S' is in $[0, 2^t - 1]$. Then, we create a set P of $n\Delta$ points from S'. Specifically, for each $i \in [1, n]$, add to P the following Δ points: $(v_i\Delta, v_i\Delta + j)$ with weight $v_i\Delta + j$ for $j = 0, ..., \Delta - 1$—we say that these Δ points form the *group* of v_i.

We create a 2-sided Δ-AWT structure T of optimal space on P; in other words, T occupies $O(\frac{n\Delta}{\Delta B}) = O(n/B)$ blocks. Given a predecessor query with parameter x on S, we perform a 2-sided Δ-AWT query with parameters $q = (-\infty, x\Delta] \times (-\infty, \infty)$ and $k = \Delta$. Suppose that the Δ-AWT query returns a value σ. We set $x' = \lceil \sigma/\Delta \rceil$. We argue that $I = [x', x]$ is an interval we are looking for:

Lemma 5. *$|S \cap I|$ has at least 1 and at most $O(1)$ elements.*

Proof. Let v_i be the predecessor of x in S. Then, the points in the group of v_i have the Δ largest weights among all the points in $P \cap q$. Therefore, $\sigma \leq v_i\Delta$ (otherwise, less than $k = \Delta$ points in $P \cap q$ have weights at least σ). Hence, $x' \leq v_i$, meaning that $S \cap I$ has at least one element.

Let j be the largest integer such that $v_j \geq x'$. We know that all the $(i-j+1)\Delta$ points in the groups of $x_j, x_{j+1}, ..., x_i$ fall in $P \cap q$ and have weights at least σ. By definition of Δ-AWT, at most $O(k + \Delta) = O(\Delta)$ such points can exist. Hence, $i - j + 1 = O(1)$. □

It thus follows from the above definition that T must incur $\Omega(\log_B n)$ I/Os answering a 2-sided Δ-AWT query. Since T is created on a set P of size $n\Delta$, we thus conclude the proof of Theorem 3.

4.2 Δ-AWT Structures

This subsection is devoted to I/O-efficient Δ-AWT structures. First of all, the discussion in Section 2 directly leads to:

Lemma 6. *There is an external memory structure of $O(n/(B\Delta))$ space that answers a 2-sided Δ-AWT query in $O(\log_B(n/B))$ I/Os.*

Proof. It suffices to point out that there is a 3D dominance structure of $O(n/B)$ space that answers a query in $O(\log_B n + k'/B)$ I/Os, where k' is the output size [1]. □

The situation with 3- and 4-sided AWT, on the other hand, is more interesting. Take 3-sided as an example. Our pointer machine structure uses $O(\frac{n}{\Delta} \log \frac{n}{\Delta})$ words. By the standard wisdom behind the existing I/O-efficient techniques, one would probably expect an EM structure of $O(\frac{n}{B\Delta} \log_B \frac{n}{\Delta})$ blocks. However, the existence of such a structure implies an internal memory structure that uses only $O(\frac{n}{\Delta} \log_B \frac{n}{\Delta})$ words! Note that we have the luxury to increase B arbitrarily such that by using $B = (n/\Delta)^c$ for any $c \in (0, 1]$, we actually get an internal memory structure of $O(n/\Delta)$ size for 3-sided Δ-AWT—an ambitious goal whose achievement still remains elusive to us.

The hidden message behind the above discussion is that we should instead try to work with an EM structure that uses as many words as its internal-memory counterpart (e.g., for 3-sided, this means an EM structure of $O(\frac{n}{B\Delta} \log \frac{n}{\Delta})$ blocks—note that the logarithm has base 2). If one accepts this as the new goal, then it is not hard to derive:

Lemma 7. *There is an external memory structure of $O(\frac{n}{B\Delta} \log^2 \frac{n}{\Delta} / \log\log \frac{n}{\Delta})$ space that answers a 4-sided Δ-AWT query in $O(\log_B(n/\Delta))$ I/Os.*

Proof. Trivially applying the pointer machine structures already meets the space requirement, but the query cost is $O(\log(n/\Delta))$. The log base can be increased to B by the standard technique of grouping nodes of multiple levels of an f-ary tree into a block when $f \leq B$. □

The above lemma allows us to obtain optimal top-k ORR structures in external memory. The key is to set Δ to $B \log_B n$. Recall that by the reasoning described in Section 3, we will eventually rely on a 5-sided range reporting structure with query cost $O(\log_B n + k'/B)$, where k' is the number of points returned. Hence, by setting $\Delta = B \log_B n$, a top-k Δ-AWT query returns a threshold σ that will ensure $k' = O(k+\Delta) = O(k + B \log_B n)$, such that $O(\log_B n + k'/B)$ is just $O(\log_B n + k/B)$.

Lemma 8. *When $\Delta = B \log_B n$, the space complexity in Lemma 7 is $O(\frac{n}{B} \frac{\log n}{\log\log_B n})$.*

Proof. See the appendix. □

To close the deal, we need:

Lemma 9. *We can store n 3D points in a structure of $O(\frac{n}{B} \frac{\log n \cdot (\log\log B)^2}{\log\log_B n})$ space such that a 5-sided range reporting query can be answered in $O(\log_B n + k'/B)$ I/Os, where k' is the number of points reported.*

Proof. See Section 4.3. □

Now we claim:

Theorem 4. *The following statements are true in external memory:*

1. *There is a structure of $O(n/B)$ space that answers a 2-sided top-k ORR query in $O(\log_B n + k/B)$ I/Os.*
2. *For any constant c, a structure that answers a 3-sided top-k ORR query in $O(\log_B^c n + k/B)$ I/Os must use $\Omega(\frac{n}{B} \frac{\log n}{\log\log_B n})$ space.*
3. *There is a structure of $O(\frac{n}{B} \frac{\log n}{\log\log_B n})$ size that answers a 3-sided top-k ORR query in $O(\log_B n + k/B)$ I/Os.*

4. *There is a structure of $O(\frac{n}{B}\frac{\log n\cdot(\log\log B)^2}{\log\log_B n})$ size that answers a 4-sided top-k ORR query in $O(\log_B n+k/B)$ I/Os.*

PROOF. The proof follows the same argument for Theorem 2 but with a few changes. To prove Statement 1, the 3D dominance structure is the I/O-efficient one proposed in [1]. In the argument for statement 2, we should set k_i to $2^{i-1}B\log_B^c n$ instead. For Statement 3, we use the $Q(3,1)$-structure of [2] in external memory. For Statement 4, the 5-sided range reporting structure is the one in Lemma 9. $\square$

4.3 5-Sided Range Reporting

This subsection proves Lemma 9 by giving such a structure. Recall that the input is a set P' of n points in $\mathbb{R}^3$. Given a 5-sided rectangle $q=[x_1,x_2]\times[y_1,y_2]\times(-\infty,z]$, a query returns all the points in $P'\cap q$. In external memory, the best result we are aware of is a structure of $O(\frac{n}{B}(\frac{\log n}{\log\log_B n})^2)$ space and $O(\log_B n+k'/B)$ query cost due to Afshani et al. [2] (see [16] for an alternative result when the points fall on a 3D grid). Note that Lemma 9 strictly improves this result.

The next lemma is easy:

LEMMA 10. *For any parameter $\lambda\in[B,|P'|]$, there is an external memory structure of $O(\frac{|P'|}{B}\log^2(|P'|/\lambda))$ space that answers a 5-sided range reporting query in $O(\lambda/B+\log_B|P'|+k'/B)$ I/Os, where k' is the number of points reported.*

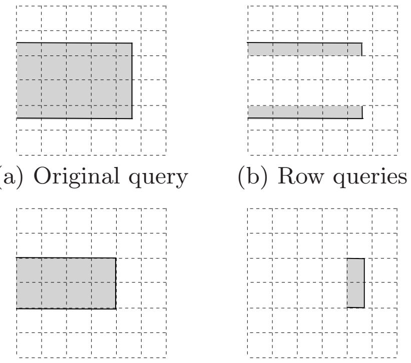
PROOF. Applying range-tree ideas to the 3D dominance structure of [1] yields a structure of $O(\frac{|P'|}{B}\log|P'|)$ space that supports 4-sided range reporting (i.e., the query region has the form $(-\infty,x]\times[y_1,y_2]\times(-\infty,z]$) in $O(\log_B|P'|)$ I/Os plus the minimum output cost. Note that the query cost has base B instead of base 2. The space can be brought down to $O(\frac{|P'|}{B}\log(|P'|/\lambda))$ by using fat leaves of size λ in the base tree. The query cost has an additive term of $O(\lambda/B)$ because we may need to scan the points in a leaf. Applying the same idea again on this structure gives the lemma. $\square$

Our structure of Lemma 9 leverages the above fact to bootstrap an adapted version of a pointer machine structure of [3]. Define $\mathcal{F}^{(1)}(n)=\sqrt{nB}\log n$, and $\mathcal{F}^{(i+1)}(n)=\mathcal{F}^{(1)}(\mathcal{F}^{(i)}(n))$ for $i\geq 1$. Let $\mathcal{F}^*(n)$ be the smallest integer i such that $\mathcal{F}^{(i)}(n)\leq B\log^{0.99}n\cdot(\log B+3\log\log n)^2$. We have:

LEMMA 11. *$\mathcal{F}^*(n)=\log\frac{\log n}{\log\log_B n}+O(1)$.*

PROOF. Note that $\mathcal{F}^{(i)}(n)\leq B\log^2 n\cdot n^{1/2^i}$. Hence, when $i=\lceil\log\frac{\log n}{\log\log_B n}\rceil$, $\mathcal{F}^{(i)}(n)<B\log^3 n$. One can then verify that $\mathcal{F}^{(2)}(B\log^3 n)\leq B\log^{0.99}n\cdot(\log B+3\log\log n)^2$ (see the appendix for details). $\square$

Structure. Set $\theta=\sqrt{n/(B\log^2 n)}$. We impose an orthogonal grid G in the xy-plane with θ rows and θ columns, such that each row (column) covers the xy-projections of no more than $\sqrt{nB}\log n$ points in P'. For each row (column) of G, we build a structure of [1] to answer 3D dominance queries on the points of P' whose xy-projections fall in that row (column).


(a) Original query (b) Row queries

(c) Grid query (d) Vertical query

Figure 5: Answering a 4-sided query

Given two points in $\mathbb{R}^3$, (same as in Section 2) we say that the former is *lower* if it has a smaller z-coordinate. Let e be a cell in G, and $P'(e)$ be the set of points in P' whose xy-projections fall in e. We store the points of $P'(e)$ in a linked list sorted by z-coordinate. The B-th lowest point in $P'(e)$ is called the *sentinel* of e; note that if $|P'(e)|<B$, no sentinel is defined for e. Let the *pilot set* of e be the lowest B points in $P'(e)$. Let Π be the union of the pilot sets of all the cells e in G. We create a structure of Lemma 10 on Π with $\lambda=B$, and refer to it as a *pilot structure*. Since $|\Pi|\leq B\theta^2=O(B\frac{n/B}{\log^2 n})=O(n/\log^2 n)$, the pilot structure uses $O(n/B)$ space.

We then recursively apply the above construction on the subproblem defined by each row and column, respectively (namely, on the set of points of P' whose xy-projections fall in that row or column). Note that, on each sub-problem, θ should be calculated by setting n to the size of that subproblem (i.e., θ decreases with the subproblem's size). Recursion completes when a sub-problem has at most $B\log^{0.99}n\cdot(\log B+3\log\log n)^2$ points, on which we build a structure of Lemma 10 with $\lambda=B\log_B n$. If we amortize the space of this structure onto those $B\log^{0.99}n\cdot(\log B+3\log\log n)^2$ points, the amount of space that each point accounts for is

$$O\left(\frac{1}{B}\log^2\frac{B\log^{0.99}n\cdot(\log B+3\log\log n)^2}{B\log_B n}\right)$$
$$=\quad O((\log\log B)^2/B).$$

Let h be the number of levels in the recursion. Standard analysis shows that the overall space consumption is $O(2^h\frac{|P'|}{B}(\log\log B)^2)$, which is $O(\frac{n}{B}\frac{\log n\cdot(\log\log B)^2}{\log\log_B n})$ by Lemma 11.

Query. We will first explain how to perform range reporting using 4-sided rectangles of the form $q=[-\infty,x]\times[y_1,y_2]\times(-\infty,z]$. If q falls completely within a row of G, we recursively answer the query on the subproblem defined by that row. At the end of recursion, we are looking at no more than $B\log^{0.99}n\cdot(\log B+3\log\log n)^2$ points; by Lemma 10, we answer the query in $O(\lambda/B+1+\log_B\log_B n)=O(\log_B n)$ I/Os plus the minimum output cost.

Now consider that q intersects at least two rows of G. The query can be partitioned into (i) two *row queries*, each being a 3D dominance query whose search region falls in a row on the xy-plane (see Figure 5a), (ii) a *grid-query*, a 4-sided query whose search region is aligned on the grid in the xy-plane (Figure 5b), and (iii) a *vertical query*, another 4-sided query whose search region falls in a column of G on the xy-plane (Figure 5c). We answer the row queries directly using the 3D dominance structures of the corresponding rows, and send the vertical query to the subproblem defined by that column. Next, we explain how to answer the grid query.

Let g be the (3D) search region of the grid query. We first search the pilot structure to report all the points stored there that fall in g. Furthermore, for each cell e whose sentinel has been reported, we jump to the linked list of $P'(e)$, and report the non-pilot points in $P'(e) \cap g$ in ascending order of z-coordinate.

If we denote $Q(m)$ as the query cost (only the output independent part) on a problem of size m. Then, it follows from the above discussion that $Q(m) = O(\log_B n)$ if $m \le B \log^{0.99} n \cdot (\log B + 3 \log\log n)^2$; otherwise:

$$Q(m) \;=\; Q(\mathcal{F}^{(1)}(m)) + O(\log_B m)$$

Setting $i = \mathcal{F}^*(m)$, we have:

$$\begin{aligned}
Q(m) &= O(\log_B n) + \sum_{j=1}^{i} O(\log_B(\mathcal{F}^{(j)}(m))) \\
&= O(\log_B n) + \sum_{j=1}^{i} O\left(\log_B(B \log^2 m \cdot m^{1/2^j})\right) \\
&= O(\log_B n) + \sum_{j=1}^{i} O\left(1 + \log_B \log m + \frac{1}{2^j} \log_B m\right) \\
&= O(\log_B n) + \mathcal{F}^*(m) \cdot O(1 + \log_B \log m)
\end{aligned}$$

where the second equality used the fact that $\mathcal{F}^{(j)}(m) \le B \log^2 m \cdot m^{1/2^j}$. Therefore:

$$\begin{aligned}
Q(n) &= O(\log_B n) + \mathcal{F}^*(n) \cdot O(\log_B \log n) \\
&= O(\log_B n) + O(\log\log n) \cdot O(\log_B \log n) \\
&= O(\log_B n).
\end{aligned}$$

Following the discussion of [3], a 5-sided query can be reduced to four 4-sided queries (which are can be processed as above) and a grid query whose search region is aligned on the grid G in the xy-plane. The grid query can be answered in $O(\log_B n)$ I/Os plus the linear output cost in the same way as we answered a query of Figure 5c. This completes the proof of Lemma 9.

4.4 Orthogonal Range Max

When k is fixed to 1, we can remove the $(\log\log B)^2$ factor in Statement 4 of Theorem 4. Furthermore, the structure can be made much simpler by utilizing the linear size structure of [20] for 3-sided range max—even without resorting to Δ-AWT.

THEOREM 5. *There is a structure of $O(\frac{n}{B} \frac{\log n}{\log\log_B n})$ space that answers an orthogonal range max query in $O(\log_B n)$ I/Os.*

PROOF. Let P be the input set of points in $\mathbb{R}^2$. Build a B-tree T on their x-coordinates with internal fanout $f = \log_B n$. Let u be an internal node. Naturally, u corresponds to a vertical slab in $\mathbb{R}^2$, which is divided into f smaller slabs by its children. Let P_u be the points of P in the slab of u, and let $n_u = |P_u|$. Create a structure on the points of P_u in each child slab to answer 3-sided orthogonal range max queries in $O(\log_B n)$ I/Os. Impose an $f \times \frac{n_u}{m}$ grid G_u with $m = \log^3 n$ in the same way as explained in Section 3. For each dyadic rectangle, store the maximum weight of the points of P_u in the rectangle. As there are $f \frac{n_u}{m} \log f \cdot \log n \le n_u$ dyadic rectangles, doing so requires $O(n_u/B)$ space. Finally, within each row of G_u, build the orthogonal range max structure of [5] (see Table 1) on the points of P_u in that row.

The overall space is $O(\frac{n}{B} \frac{\log n}{\log\log_B n})$. The query algorithm proceeds in essentially the same manner as described in Section 3. It is easy to see that the query cost is $O(\log_B n)$ (notice that within each row, the structure of [5] requires only $O((\log_B \log n)^2) = O(\log_B n)$ I/Os). $\square$

It is worth mentioning that the simplicity of our structure does *not* carry over to the pointer machine model, where it remains unknown whether there is an $O(n)$-size structure that can answer a 3-sided query in $O(\log n)$ time. In other words, we still need to go through Δ-AWT to obtain the claimed orthogonal range max structure in Table 1.

5. LINEAR SPACE TOP-K ORR

In this section, we give a top-k ORR structure in external memory that uses $O(n/B)$ space, and answers a query in $O(\sqrt{n/B} + k/B)$ I/Os. This tradeoff can be proven to be optimal among the class of so-called "non-replicate" structures as will be discussed later.

LEMMA 12. *We can store n 3D points in a structure of $O(n/B)$ space such that a 5-sided range reporting query can be answered in in $O(\sqrt{n/B} + k'/B)$ I/Os, where k' is the number of points reported.*

PROOF. This can be achieved by combining ideas of the kd-tree and the priority search tree. Build a kd-tree on the projections of the points of P onto the xy-plane. The tree has $O(n/B)$ leaves, each containing between $B/2$ and B points. In each node u, we store B *pilot points*, which are the B points with the highest z-coordinates that are not pilot points in any proper ancestor of u. Clearly, the space occupied is $O(n/B)$ blocks. Recall that each node u can be thought of as being associated with a 2D minimum bounding rectangle (MBR) of all the points in its subtree.

Consider a query with search region $q' = [x_1, x_2] \times [y_1, y_2] \times [z, \infty)$. Let q be the xy-projection of q'. We search the kd-tree with q in the standard way to obtain a canonical set S of $O(\sqrt{n/B})$ nodes whose MBRs together cover q. At each node which is an ancestor of at least one node in S, we scan all its pilot points to report those lying in q'. Let v be a child of a canonical node in S. Report the pilot points of v whose z-coordinates are at least z. If all the pilot points of v are reported, we recursively repeat this procedure at the two children of v. Standard analysis shows that the algorithm performs $O(\sqrt{n/B} + k'/B)$ I/Os. $\square$

THEOREM 6. *There is a structure of $O(n/B)$ space that answers a 4-sided top-k ORR query in $O(\sqrt{n/B} + k/B)$ I/Os.*

PROOF. We only need a Δ-AWT structure of Lemma 7 with $\Delta = \sqrt{n/B}$, and a 5-sided reporting structure of Lemma 12. $\square$

Remark on Non-Replication. Let P be the input set of n 2D points to the traditional orthogonal range reporting problem. Suppose that a query is required to report the ids of the points in the result. This means that a structure must also store point ids. A structure is said to be *non-replicate* [12, 22] if the id of each point is stored only once. In general, each point in P is associated an *information field* of L words, such that a query is required to report the information fields of all the result points. A structure is non-replicate [22] if it uses at most $O(n/B) + nL/B$ blocks (which implies that the information field of each point can be stored only once).

Our structure of Theorem 6 is a non-replicate one—in the structure of Lemma 12, we associate each node u in the kd-tree with $1 + xL/B$ blocks storing the information fields of the $x \in [1, B]$ pilot points of u. The space cost of the top-k ORR structure becomes $O(n/B) + nL/B$, whereas its query cost is $O(\sqrt{n/B} + kL/B)$. This is optimal following a lower bound of [22] and our reduction from (traditional) ORR to top-k ORR in Section 3.

ACKNOWLEDGEMENTS

This work was supported in part by Grants GRF 4168/13 and GRF 142072/14 from HKRGC.

6. REFERENCES

[1] Peyman Afshani. On dominance reporting in 3D. In *Proceedings of European Symposium on Algorithms (ESA)*, pages 41–51, 2008.

[2] Peyman Afshani, Lars Arge, and Kasper Dalgaard Larsen. Orthogonal range reporting in three and higher dimensions. In *Proceedings of Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 149–158, 2009.

[3] Peyman Afshani, Lars Arge, and Kasper Dalgaard Larsen. Orthogonal range reporting: query lower bounds, optimal structures in 3-d, and higher-dimensional improvements. In *Proceedings of Symposium on Computational Geometry (SoCG)*, pages 240–246, 2010.

[4] Peyman Afshani, Gerth Stolting Brodal, and Norbert Zeh. Ordered and unordered top-k range reporting in large data sets. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 390–400, 2011.

[5] Pankaj K. Agarwal, Lars Arge, Jun Yang, and Ke Yi. I/O-efficient structures for orthogonal range-max and stabbing-max queries. In *Proceedings of European Symposium on Algorithms (ESA)*, pages 7–18, 2003.

[6] Alok Aggarwal and Jeffrey Scott Vitter. The input/output complexity of sorting and related problems. *Communications of the ACM (CACM)*, 31(9):1116–1127, 1988.

[7] Stephen Alstrup, Gerth Stølting Brodal, and Theis Rauhe. Optimal static range reporting in one dimension. In *Proceedings of ACM Symposium on Theory of Computing (STOC)*, pages 476–482, 2001.

[8] Gerth Stolting Brodal, Rolf Fagerberg, Mark Greve, and Alejandro Lopez-Ortiz. Online sorted range reporting. In *International Symposium on Algorithms and Computation (ISAAC)*, pages 173–182, 2009.

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

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

[11] Greg N. Frederickson. An optimal algorithm for selection in a min-heap. *Information and Computation*, 104(2):197–214, 1993.

[12] Kothuri Venkata Ravi Kanth and Ambuj K. Singh. Optimal dynamic range searching in non-replicating index structures. In *Proceedings of International Conference on Database Theory (ICDT)*, pages 257–276, 1999.

[13] Marek Karpinski and Yakov Nekrich. Top-k color queries for document retrieval. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 401–411, 2011.

[14] Edward M. McCreight. Priority search trees. *SIAM Journal of Computing*, 14(2):257–276, 1985.

[15] Gonzalo Navarro and Yakov Nekrich. Top-k document retrieval in optimal time and linear space. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1066–1077, 2012.

[16] Yakov Nekrich. I/O-efficient point location in a set of rectangles. In *Latin American Symposium on Theoretical Informatics (LATIN)*, pages 687–698, 2008.

[17] Mihai Patrascu and Mikkel Thorup. Time-space trade-offs for predecessor search. In *Proceedings of ACM Symposium on Theory of Computing (STOC)*, pages 232–240, 2006.

[18] Saladi Rahul, Prosenjit Gupta, Ravi Janardan, and K. S. Rajan. Efficient top-k queries for orthogonal ranges. In *Proceedings of International Workshop on Algorithms and Computation (WALCOM)*, pages 110–121, 2011.

[19] Saladi Rahul and Ravi Janardan. A general technique for top-k geometric intersection query problems. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 26(12):2859–2871, 2014.

[20] Cheng Sheng and Yufei Tao. New results on two-dimensional orthogonal range aggregation in external memory. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 129–139, 2011.

[21] Cheng Sheng and Yufei Tao. Dynamic top-k range reporting in external memory. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, 2012.

[22] Yufei Tao. Indexability of 2d range search revisited: constant redundancy and weak indivisibility. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 131–142, 2012.

[23] Yufei Tao. A dynamic I/O-efficient structure for one-dimensional top-k range reporting. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 256–265, 2014.

APPENDIX

Proof of Lemma 8

When $\Delta = B \log_B n$, we have

$$\begin{aligned}
\frac{n}{B\Delta} \frac{\log^2 \frac{n}{\Delta}}{\log\log \frac{n}{\Delta}} &\leq \frac{n}{B^2 \log_B n} \cdot \frac{\log^2 n}{\log\log n} \\
&= \frac{n \log B \cdot \log n}{B^2 \log\log n} \\
&\leq \frac{n}{B} \frac{\log n}{B^{0.99} \log\log n} \\
&= O\left(\frac{n}{B} \frac{\log n}{\log\log_B n} \right).
\end{aligned}$$

Additional Details in the Proof of Lemma 11

Here are the details of showing $\mathcal{F}^{(2)}(B \log^3 n) \leq B \log^{0.99} n \cdot (\log B + 3 \log\log n)^2$. By definition:

$$\begin{aligned}
\mathcal{F}^{(1)}(B \log^3 n) &= \sqrt{B \log^3 nB} \cdot \log(B \log^3 n) \\
&= B \log^{1.5} n \cdot (\log B + 3 \log\log n).
\end{aligned}$$

Hence,

$$\begin{aligned}
& \mathcal{F}^{(2)}(B \log^3 n) \\
= \; & \mathcal{F}^{(1)}(B \log^{1.5} n \cdot (\log B + 3 \log\log n)) \\
= \; & \sqrt{B \log^{1.5} n \cdot (\log B + 3 \log\log n) \cdot B} \\
& \log(B \log^{1.5} n \cdot (\log B + 3 \log\log n)) \\
= \; & B \log^{0.75} n \cdot (\log B + 3 \log\log n)^{0.5} \\
& (\log B + 1.5 \log\log n + \log(\log B + 3 \log\log n)) \\
< \; & B \log^{0.75} n \cdot (\log B + 3 \log\log n)^{1.6}.
\end{aligned}$$