Indexability of 2D Range Search with Constant Redundancy*

Yufei Tao

Department of Computer Science and Engineering
Chinese University of Hong Kong
Hong Kong
taoyf@cse.cuhk.edu.hk

August 25, 2014

Abstract

We consider the 2D *orthogonal range search problem* defined as follows. Let P be a set of 2D points where each point $p \in P$ carries an arbitrary *information field*. The goal is to store P in a data structure so that, given any axis-parallel query rectangle q, we can report the information fields of all the points in $P \cap q$ efficiently. For every constant integer $r \geq 1$, we prove a lower bound on the query cost of any external memory structure that stores an information field r times on average. Our lower bounds do not require that information fields be stored as atoms. Instead, a structure is free to store the bits of an information field in different blocks, and/or store some bits more often than others. We also describe structures whose query efficiency matches the lower bounds.

*A preliminary version of this paper appeared in PODS'12.

1 Introduction

We consider the 2D *orthogonal range search problem* defined as follows. The input is a set P of N points in $\mathbb{R}^2$, where each point $p \in P$ carries an arbitrary *information field*—denoted as *info*(p)—of length $L \geq 1$ words, where L is a parameter (not necessarily a constant). Given an axis-parallel rectangle q, a *range query* returns the information fields of all the points in $P \cap q$. The goal is to store P in a structure so that range queries can be answered efficiently in the worst case.

Our discussion will focus on the standard *external memory* (EM) model [1]. In this model, a computer is equipped with M words of memory and a disk that has been formatted into *blocks* of size B words. It holds that $M \geq 2B$. An *I/O* reads a block of data from the disk into memory, or conversely, writes B words from memory into a disk block. The *cost* of an algorithm is measured as the number of I/Os performed, whereas the *space* of a structure is measured as the number of disk blocks occupied. We will denote by w the number of bits in a machine word.

1.1 Weakly Indivisible Structures

A primary goal of this paper is to derive lower bounds on the best achievable query cost of a class of structures. In this section, we will give the definition of this class, and clarify several relevant concepts.

For lower bound analysis, we will be concerned only with the way that information fields are stored and retrieved by a structure. Let us refer to each bit in the *info*(p) of a point p as an *information bit*. We will regard a structure simply as a set of blocks, each of which contains up to wB information bits. Note that even though all the information bits may appear in the disk just as either 0 or 1, each of them is uniquely characterized by (i) the point p that the bit belongs to, and (ii) the position of the bit in *info*(p).

To answer a range query with rectangle q, the structure is required to do the following for every point $p \in P \cap q$ and every information bit of *info*(p): *access at least a block where the bit is stored*. In other words, if $K = |P \cap q|$, then in total wKL information bits must be read from the disk.

We say that a structure modeled as above is *weakly indivisible*. The modeling is less stringent than the conventional indivisibility assumption [9, 12] where every information field (usually, the coordinates of a point) is always stored as an atom within the same block—we call such a structure *strongly indivisible*. Notice that a weakly indivisible structure is allowed two functionalities beyond strong indivisibility: (i) chop up an information field, and store the resulting pieces in different blocks, and (ii) store some bits of an information field more often than the other bits.

It should be noted that our modeling of structures is nonetheless still captured by the indivisibility assumption (albeit in a weaker way), whose removal should allow the information bits to be *computed*. As mentioned earlier, we do not allow a structure to be this powerful: every information bit reported must be loaded from the disk. In EM, efforts towards genuinely removing the indivisibility assumption have been reported in [11, 19, 20] on, however, problems different from ours.

Given an integer $r \geq 1$, a structure is *r-redundant* if it stores at most $r \cdot N \cdot wL$ information bits. In particular, a 1-redundant structure is called *non-replicating* because apparently all information fields must be stored at least once.

1.2 Motivation

Redundancy vs. Query Efficiency. Most practical structures for orthogonal range search are non-replicating. Several well-known examples are the *kd-tree* [5, 14], the *R-tree* [4, 8], the *O-tree* [12],

and the *cross-tree* [7]. The main reason [12] behind their popularity is that, the input cardinality N in practice can be so huge that each information field should be stored only once in order to meet a tight space budget.

Today, while linear-size structures are still mandatory to cope with the vast data scale in many applications, constraining each information field to be stored *exactly once* appears excessively stringent. After all, the dollar-per-byte price of hard disks has been dropping continuously, making it realistic to replicate an information field a small number of times. In this paper, we are interested in r-redundant structures where r is a constant. It is well-known that duplicating information fields brings *polynomial* improvement to query efficiency. Consider, for example, $L = O(1)$. Any 1-redundant strongly-indivisible structure must entail $\Omega(\sqrt{N/B} + K/B)$ I/Os in answering a query reporting K information fields [9, 12]. However, when constant r is *sufficiently* large, it is possible to guarantee query cost $O((N/B)^\epsilon + K/B)$ for arbitrarily small constant $\epsilon > 0$ using a structure of $O(N/B)$ space [17].

What the previous research has not shown is precisely *how much* benefit can be gained by investing additional space. For example, what is the best query cost possible for $r = 2$? Conversely, if we aim at query cost, say, $O((N/B)^{1/10} + K/B)$ for $L = O(1)$, how many times must an information field be stored? At a higher level, if one looks at the query complexity as a function of r, the previous research has resolved the function at two extremes: $r = 1$, and r being an arbitrarily large constant. The tradeoff within the intermediate range (i.e., from $r = 2$ and onwards) still remains elusive.

Information Fields. No previous lower-bound study of orthogonal range search (surveyed in Section 2) explicitly considered information fields. Storing a point was implicitly understood as storing its *coordinates*, namely, $L = 2$. In such a case, the notion of "r-redundancy" with $r = O(1)$ appears unjustified when an arbitrarily large hidden constant is permitted in the space complexity $O(N/B)$. This is most obvious for non-replicating structures: why force $r = 1$ while allowing the structure to use cN/B blocks for some constant $c \gg 1$? One can harvest *polynomial* improvement in query cost by setting $r = 2$ and in the meantime, perhaps with some clever tricks, still maintain the space to be at most cN/B.

The introduction of information fields removes the above oddity by making certain r-redundant structures especially appealing: those consuming $O(N/B) + rNL/B$ blocks—note that the second term is outside the big-O. Intuitively, in addition to the storage of information fields, such a structure is permitted to use only $O(N/B)$ extra blocks, which prevents "cheating tricks" of keeping $O(NL/B)$ blocks of *search-guiding* data that have nothing to do with information fields. It should be clear that the separation of good r-redundant structures from cheating ones owes to the fact that L can be $\omega(1)$. Conventionally, with $L = O(1)$, $O(N/B) + rNL/B$ is hardly any more meaningful than simply $O(N/B)$.

As a note for practice, information fields do make sense in many applications. A user is seldom interested in the coordinates of a point p; usually, it is the details of the entity represented by p that have triggered the query in the first place. Let the entity be a hotel, for example; then $info(p)$ may be its concrete description (e.g., rating, prices, amenities, etc.). The size of $info(p)$ may not necessarily be treated as a constant.

Weak vs. Strong Indivisibility. As reviewed in Section 2, existing lower bounds on orthogonal range search are unanimously based on the strong indivisibility assumption. An intriguing question is whether a structure gains any extra power by being weakly indivisible.

1.3 Our Results

Our first main result is:

Theorem 1. *Consider $w = \Theta(\log n)$ and $L = B^\mu$, where μ is any constant satisfying $0 \leq \mu < 1$. Let $r \geq 2$ be a constant integer, and ϵ any value satisfying $0 < \epsilon \leq 1/r$. To ensure worst-case query cost*

$$O\left(\left(\frac{NL}{B}\right)^{\frac{1}{r}-\epsilon} + \frac{KL}{B}\right)$$

a weakly-indivisible structure must store at least $rNwL \cdot (1 - o(1))$ information bits.

Note that ϵ does not need to be a constant in the above statement. The theorem implies:

Corollary 1. *Consider $w = \Theta(\log n)$ and $L = B^\mu$, where μ is any constant satisfying $0 \leq \mu < 1$. For any constant integer $r \geq 1$, no r-redundant weakly-indivisible structure can guarantee worst-case query cost $O((NL/B)^{\frac{1}{r+1}-\epsilon} + KL/B)$, no matter how small $\epsilon > 0$ is.*

To understand why, notice by Theorem 1 that a structure must store $(r + 1) \cdot NwL \cdot (1 - o(1))$ information bits to achieve query cost $O((NL/B)^{\frac{1}{r+1}-\epsilon} + KL/B)$. However, an r-redundant structure stores no more than $rNwL$ such bits.

To obtain the above result, we applied the analytical framework behind the *redundancy theorem* in [9]. Several new ideas, however, are needed to overcome two obstacles. First, the original framework makes the strong indivisibility assumption, and hence, needs to be extended to work with weak indivisibility. Second, while the framework is good for establishing *asymptotic* lower bounds, it is not powerful enough to argue for *constant-revealed* lower bounds. More specifically, the techniques of [9] can be used to show that at least $\frac{r}{12}NL/B$ blocks are needed to achieve the query cost in Theorem 1, far less than our target of (nearly) rNL/B.

Our second main result is:

Theorem 2. *For any constant integer $r \geq 1$, there is an r-redundant strongly-indivisible structure that uses $rNL/B+O(N/B)$ space, and answers a query in $O((NL/B)^{\frac{1}{r+1}} + KL/B)$ I/Os. Furthermore, for $L = \Omega(B)$, there is a 1-redundant strongly-indivisible structure that uses $NL/B+O(N/B)$ space, and answers a query in $O((N/B)^\epsilon + KL/B)$ I/Os, where $\epsilon > 0$ can be an arbitrarily small constant.*

Note the terms outside big-O in the space complexities. Two interesting observations can be made by combining Theorems 1 and 2. First, weakly-indivisible structures carry *no extra power* beyond strongly-indivisible ones as far as orthogonal range search is concerned. Second, the length L of information bits has an important role in the problem's hardness: the problem becomes significantly harder as soon as L drops from B to B^μ for any positive constant $\mu < 1$.

2 Related Work

We now proceed with a review of existing results on orthogonal range search in the EM model. Focus will be placed on worst-case query efficient structures using linear space $O(NL/B)$, as they are the main subject of this paper.

Structures. 1-redundant (strongly-indivisible) structures have been well studied for $L = O(1)$. The best query cost possible is $O(\sqrt{N/B} + K/B)$ (see Section 1.3). This can be achieved by a

slightly modified version [14, 15] of Bentley's kd-tree [5], ensuring that each leaf node should contain $\Theta(B)$ information fields. The O-tree of Kanth and Singh [12] also guarantees the optimal query cost, and has the advantage of being fully dynamic: each insertion and deletion can be supported in $O(\log_B N)$ I/Os amortized. The same can also be achieved by the *cross-tree* of Grossi and Italiano [7].

We are not aware of any specific studies on r-redundant structures with $r > 1$. Somewhat related is a result mentioned in [9]. For $L = O(1)$, when the data points are aligned as a $B \times B$ grid (i.e., $N = B^2$), there is an r-redundant structure that solves a query in $O((N/B)^{\frac{1}{2r}} + K/B)$ I/Os. No structure was given for general inputs in [9]. Our Theorem 1 implies that a $B \times B$ grid is not the hardest input for this problem.

Lower Bounds under Strong Indivisibility. Various models of EM structures exist for proving lower bounds. The simplest one is perhaps the *comparison-based model*, where it is easy to show that the query cost must be $\Omega(\log_B N)$, regardless of the space consumption. This bound is excessively loose because we are aiming at query cost polynomial to N/B. Progress has been made in the past 15 years towards that goal. Our discussion below concentrates on $L = O(1)$ which was assumed in the derivation of all the bounds known.

The model of Kanth and Singh [12] can be thought of as the EM-equivalent of a *pointer machine* in internal memory. It represents a data structure as a tree such that, to visit a node of the tree, a query algorithm must first access all its ancestors, and follow their pointers leading to the node. Kanth and Singh showed that any 1-redundant structure in this model must incur $\Omega(\sqrt{N/B}+K/B)$ I/Os solving a query in the worst case. Due to the model's limitations, their proof does not work for structures that cannot be viewed as a tree, or query algorithms that can jump directly to a node without fetching its ancestors.

To date, the most general model of EM structures (for orthogonal range search) is due to Hellerstein et al. [9]. Their model imposes no constraint on how a query algorithm may choose the next disk block to access—it can be any block regardless of the I/Os that have already been performed. They developed the *redundancy theorem* which is a powerful tool for analyzing the tradeoff between space and query efficiency, and generalizes earlier results [10, 13, 16] under the same model. For 2D orthogonal range search, the theorem leads to the following fact for $N = B^2$: an r-redundant structure must satisfy

$$r \geq \frac{\log B}{12 \log A} \tag{1}$$

if the cost of processing a query with $K = B$ has to be $O(A)$ where A can be any positive value at most $\sqrt{B}/4$. The fact implies that any linear-size structure must incur query cost $\Omega((N/B)^c+K/B)$ in the worst case for some constant $c > 0$. To see this, notice that when $r = O(1)$, (1) indicates $\frac{\log B}{\log A} = O(1)$. This, in turn, means that $\log A \geq c \cdot \log B$, leading to $A \geq B^c$, where c is some positive constant. Given the choice of N, this translates to $A \geq (N/B)^c$.

The analysis of [9] relies on the strong indivisibility assumption. More specifically, the reliance is on the notion of *flake* (see Definition 5.2 in [9]), which is a subset of points whose information fields are stored in a common block. This notion is central to proving the redundancy theorem.

Others. Better query cost is possible if super-linear space can be afforded. For $L = O(1)$, the *external range search tree* [2] uses $O((N/B) \log N/ \log \log_B N)$ space, and answers a range query in $O(\log_B N + K/B)$ I/Os. The lower bounds of [2, 9, 18] show that this is already optimal in various models of EM structures. If, on the other hand, all query rectangles are *3-sided* (having

the form $(-\infty, x] \times [y_1, y_2]$), a query can be solved in $O(\log_B N + K/B)$ I/Os for constant L using the *external priority search tree* [2], which consumes only linear space.

3 Indexability Theorem for $r = 3$

In this section, we will prove Theorem 1 in the special case of $r = 3$. This allows us to explain the core of our techniques without the extra mathematical subtleties as are needed for general r (the general proof will be given in the next section).

Our discussion concentrates on a set P of points forming an $n \times n$ grid, namely, $N = n^2$. We choose n such that the query cost stated in the theorem becomes $O(nL/B)$ when $K = n$ information fields are reported. For this purpose, we solve n from:

$$\left(\frac{n^2 L}{B}\right)^{\frac{1}{3}-\epsilon} = \frac{nL}{B} \quad (2)$$

$$\Leftrightarrow n = \left(\frac{B}{L}\right)^{\frac{\epsilon+2/3}{2\epsilon+1/3}} = B^{\frac{(\epsilon+2/3)(1-\mu)}{2\epsilon+1/3}}. \quad (3)$$

Given $\epsilon \in (0, 1/3]$, it holds that $1 \leq \frac{\epsilon+2/3}{2\epsilon+1/3} < 2$. This, together with $\mu < 1$, indicates that n is always greater 1. For simplicity, let us assume that $\sqrt{n}$ is an integer (see the end of this section for the removal of the assumption).

From now on, we will consider only queries with output size $K = n$. Each such query must report nwL information bits. Given the choice of n in (3), the query is answered in

$$O\left(\left(\frac{n^2 L}{B}\right)^{\frac{1}{3}-\epsilon} + \frac{nL}{B}\right) = O(nL/B)$$

I/Os (note from (2) and $\epsilon \in (0, 1/3]$ that $nL/B \geq 1$). Hence, we can assume that its cost is at most $\alpha nL/B$ for some constant $\alpha > 0$.

Next, we define a notion called *bit-flake* to replace the concept of *flake* in [9].

Definition 1. *Consider a query with rectangle q. A* bit-flake of the query *is a non-empty set f of information bits satisfying two conditions:*

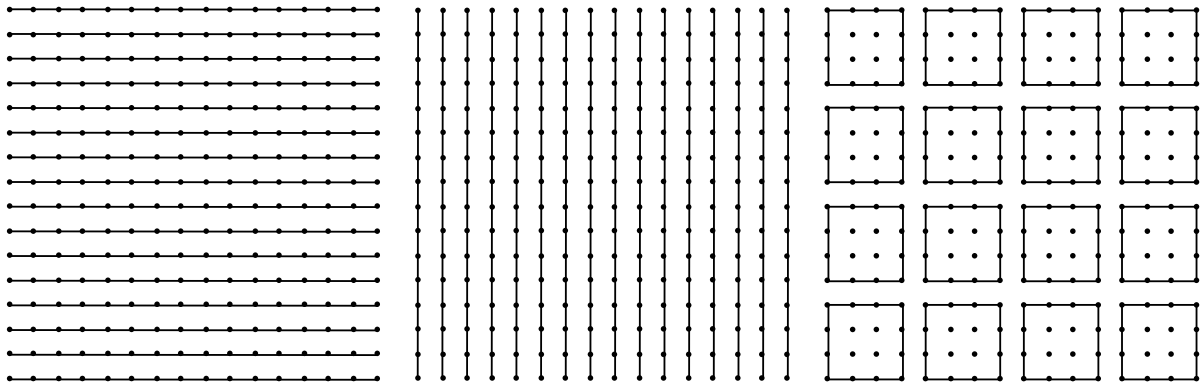
- *All the bits in f are stored in the same block accessed by the query.*
- *Each bit in f is in the info(p) of some point p covered by q.*

To illustrate the definition in another way, consider a block b accessed by the query. Let X be the set of all information bits in b that belong to points inside q. Then, any non-empty subset of X is a bit-flake.

Lemma 1. *A query with output size n has at least*

$$\frac{nwL}{s} - \frac{\alpha nL}{B}$$

pair-wise disjoint bit-flakes of size $s > 0$.



(a) Row queries (b) Column queries (c) Square queries

Figure 1: Hard input and queries on a 16×16 grid

Proof. We use the following algorithm to collect a number of bit-flakes needed to prove the lemma. Assume that the query accessed $z \leq \alpha nL/B$ blocks, denoted as $b_1, ..., b_z$, respectively (ordering does not matter). For each $i \in [1, z]$, let X_i be the set of information bits in b_i that (i) belong to points covered by q, and (ii) are absent from the preceding blocks $b_1, ..., b_{i-1}$. Clearly, $X_1, ..., X_z$ are pair-wise disjoint. Furthermore, $\sum_{i=1}^{z} |X_i| = nwL$ because each information bit of every point in q is in exactly one X_i.

From X_i, we form $\lfloor |X_i|/s \rfloor$ pair-wise disjoint bit-flakes, by dividing arbitrarily the bits of X_i into groups of size s, leaving out at most $s-1$ bits. The bit-flakes thus created from $X_1, ..., X_z$ are mutually disjoint. The number of those bit-flakes equals

$$\sum_{i=1}^{z} \left\lfloor \frac{|X_i|}{s} \right\rfloor > \sum_{i=1}^{z} \left(\frac{|X_i|}{s} - 1 \right) = \frac{nwL}{s} - z \geq \frac{nwL}{s} - \frac{\alpha nL}{B}$$

as claimed. □

We construct $3n$ queries as follows. Recall that the points of P form an $n \times n$ grid. Each row or column of the grid is taken as a query, referred to as a *row query* or *column query*, respectively. This has defined $2n$ queries. Each of the remaining n queries is a $\sqrt{n} \times \sqrt{n}$ square, and is therefore called a *square query*. Specifically, the square touches $\sqrt{n}$ consecutive rows and columns of the grid, respectively. All the n square queries are mutually disjoint, and together cover the entire P. Figure 1 illustrates these queries for $n = 16$.

By Lemma 1, the $3n$ queries define in total at least $3n(\frac{nwL}{s} - \frac{\alpha nL}{B})$ bit-flakes of size s (we will decide the value of s later), such that the bit-flakes from the same query are pair-wise disjoint. Refer to all these bit-flakes as *canonical bit-flakes*. Recall that the bits of a bit-flake f are in a common block b. We say that b *contains* f.

Lemma 2. *Let b be a block, and t the number of information bits in b that appear in at least one canonical bit-flake. Then, b can contain at most*

$$\frac{t}{s} + \frac{3L\sqrt{n} \cdot B^2 w^3}{s^3}$$

canonical bit-flakes.

Proof. Let F_{row} be the set of canonical bit-flakes that are contained in b, and are defined from row queries. Define F_{col} and F_{sqr} similarly with respect to column and square queries. The total number of information bits covered by at least one bit-flake in $F_{row} \cup F_{col} \cup F_{sqr}$ is t. Note that the bit-flakes in F_{row} are mutually disjoint because no two row queries retrieve a common point. The same is true for F_{col} and F_{sqr}, respectively. Hence, it holds from the set inclusion-exclusion principle that:

$$\begin{aligned}
&\sum_{f \in F_{row} \cup F_{col} \cup F_{sqr}} |f| \\
&- \sum_{f_{row} \in F_{row}, f_{col} \in F_{col}} |f_{row} \cap f_{col}| - \sum_{f_{row} \in F_{row}, f_{sqr} \in F_{sqr}} |f_{row} \cap f_{sqr}| - \sum_{f_{col} \in F_{col}, f_{sqr} \in F_{sqr}} |f_{col} \cap f_{sqr}| \\
&\leq \left| \bigcup_{f \in F_{row} \cup F_{col} \cup F_{sqr}} f \right| = t. \qquad (4)
\end{aligned}$$

Consider any bit-flakes f_{row}, f_{col} and f_{sqr} that are from F_{row}, F_{col} and F_{sqr}, respectively. There are at most wL bits in $f_{row} \cap f_{col}$ since a row query and a column query share exactly 1 point in their results (notice that the bits in $f_{row} \cap f_{col}$ must belong to the information field of that point). On the other hand, a row query and a square query share at most $\sqrt{n}$ points in their results. It follows that $|f_{row} \cap f_{sqr}| \leq wL\sqrt{n}$. Similarly, it holds that $|f_{col} \cap f_{sqr}| \leq wL\sqrt{n}$.

As the bit-flakes in F_{row} are pair-wise disjoint, we have $|F_{row}| \leq t/s$, which is at most Bw/s because a block has Bw bits. Likewise, the sizes of F_{col} and F_{sqr} are both at most Bw/s. Hence:

$$\begin{aligned}
\sum_{f_{row} \in F_{row}, f_{col} \in F_{col}} |f_{row} \cap f_{col}| &\leq wL \left(\frac{Bw}{s}\right)^2 \\
\sum_{f_{row} \in F_{row}, f_{sqr} \in F_{sqr}} |f_{row} \cap f_{sqr}| &\leq wL\sqrt{n} \left(\frac{Bw}{s}\right)^2 \\
\sum_{f_{col} \in F_{col}, f_{sqr} \in F_{sqr}} |f_{col} \cap f_{sqr}| &\leq wL\sqrt{n} \left(\frac{Bw}{s}\right)^2.
\end{aligned}$$

Plugging the above inequalities into (4) gives:

$$\begin{aligned}
\sum_{f \in F_{row} \cup F_{col} \cup F_{sqr}} |f| &\leq t + wL \left(\frac{Bw}{s}\right)^2 + 2wL\sqrt{n} \left(\frac{Bw}{s}\right)^2 \\
&\leq t + \frac{3L\sqrt{n} \cdot B^2 w^3}{s^2}
\end{aligned}$$

As $|f| = s$ for each f on the left hand side of the above inequality, we obtain:

$$|F_{row} \cup F_{col} \cup F_{sqr}| \leq \frac{t}{s} + \frac{3L\sqrt{n} \cdot B^2 w^3}{s^3}$$

thus completing the proof. □

We are now ready to prove a lower bound on the total number of information bits that must be stored. Denote by λ the number of blocks occupied by the underlying structure. Let us assume for now that $\lambda = O(NL/B)$—this assumption will be removed later with a simple trick. Hence:

$$\lambda \leq \beta NL/B = \beta n^2 L/B$$

for some constant $\beta > 0$. Denote by t_i ($1 \le i \le \lambda$) the number of information bits in the i-th block that appear in at least one canonical bit-flake. Combining Lemma 2 and the fact that there are at least $3n(\frac{nwL}{s} - \frac{\alpha nL}{B})$ canonical bit-flakes, it holds that:

$$\sum_{i=1}^{\lambda} \left(\frac{t_i}{s} + \frac{3L\sqrt{n} \cdot B^2 w^3}{s^3} \right) \quad \geq \quad 3n \left(\frac{nwL}{s} - \frac{\alpha nL}{B} \right)$$

Hence:

$$\begin{aligned} \sum_{i=1}^{\lambda} t_i &\geq 3ns \left(\frac{nwL}{s} - \frac{\alpha nL}{B} \right) - s \sum_{i=1}^{\lambda} \frac{3L\sqrt{n} \cdot B^2 w^3}{s^3} \\ &= 3ns \left(\frac{nwL}{s} - \frac{\alpha nL}{B} \right) - \lambda \cdot \frac{3L\sqrt{n} \cdot B^2 w^3}{s^2} \\ &\geq 3n^2 wL - \frac{3\alpha n^2 sL}{B} - \frac{\beta n^2 L}{B} \cdot \frac{3L\sqrt{n} \cdot B^2 w^3}{s^2} \\ &= 3n^2 wL \left(1 - \frac{\alpha s}{Bw} - \frac{\beta BLw^2\sqrt{n}}{s^2} \right). \end{aligned} \tag{5}$$

Lemma 3. *We can set $s = B^c$ for some $c > 0$ such that both $\alpha s/(Bw)$ and $\beta BLw^2\sqrt{n}/s^2$ are $o(1)$ when B is large enough.*

Proof. First, note that $w = \Theta(\log N) = \Theta(\log n) = \Theta(\log B)$, where the last equality used the fact that the term $\frac{(\epsilon+2/3)(1-\mu)}{2\epsilon+1/3}$ in (3) is $\Theta(1)$. Hence:

$$\frac{\alpha s}{Bw} = O\left(\frac{B^c}{B \log B} \right)$$

which is $o(1)$ when:

$$c \le 1. \tag{6}$$

Recall that $L = B^\mu$ where $0 \le \mu < 1$. This together with (3) gives:

$$\begin{aligned} \frac{\beta BLw^2\sqrt{n}}{s^2} &= O\left(\frac{B \cdot B^\mu \cdot \log^2 B \cdot B^{\frac{(\epsilon+2/3)(1-\mu)}{4\epsilon+2/3}}}{B^{2c}} \right) \\ &= O\left(\frac{B^{1+\mu+\frac{(\epsilon+2/3)(1-\mu)}{4\epsilon+2/3}} \log^2 B}{B^{2c}} \right) \end{aligned}$$

which is $o(1)$ when:

$$1 + \mu + \frac{(\epsilon + 2/3)(1-\mu)}{4\epsilon + 2/3} < 2c. \tag{7}$$

A value of c satisfying (6) and (7) exists when:

$$\begin{aligned} 1 + \mu + \frac{(\epsilon + 2/3)(1-\mu)}{4\epsilon + 2/3} &< 2 \\ \Leftrightarrow \frac{(\epsilon + 2/3)(1-\mu)}{4\epsilon + 2/3} &< 1 - \mu \\ \Leftrightarrow \epsilon + 2/3 &< 4\epsilon + 2/3 \end{aligned}$$

which is always true for $\epsilon > 0$. ∎

Therefore, by fixing s as stated in the lemma, we can rewrite (5) into

$$\sum_{i=1}^{\lambda} t_i \geq 3n^2 wL(1 - o(1)) = 3NwL(1 - o(1))$$

which is what we need for Theorem 1 at $r = 3$.

Removing the Assumption $\lambda = O(NL/B)$. We say that a block is *under-full* if it contains less than $wB/2$ information bits. It is easy to see that we only need to discuss structures with at most one under-full block—if there are two under-full blocks, we can always merge them into one block without increasing the cost of any queries (at all). Hence, we can remove the assumption $\lambda = O(NL/B)$ as follows. First, if the structure stores at least $3NwL$ information bits, then it already satisfies Theorem 1 with $r = 3$. Otherwise, we know that the number λ of blocks it occupies is at most $1 + \frac{3NwL}{wB/2} = O(NL/B)$.

Remarks. It is worth pointing out that the proof does not work for $\mu = 1$, because in this case (7) requires $c > 1$ which conflicts (6). This is consistent with Theorem 2 that better query cost is possible for $L = \Omega(B)$.

As mentioned in Section 2, [9] showed that, on a $B \times B$ grid, a 2-redundant structure can achieve query cost $O((N/B)^{1/4} + K/B)$ when $L = O(1)$—better than what is allowed by Theorem 1 for $r = 3$. By looking closely at our proof, one sees that our hard input is actually a $B^a \times B^a$ grid with a close to 2 (the exponent in (3) equals $\frac{\epsilon+2/3}{2\epsilon+1/3}$ for $\mu = 0$). In other words, the problem actually becomes significantly harder on a larger grid.

How are Our Techniques Different from [9]? Our proof was inspired by the analysis in [9]. In particular, we owe the method of flake counting to [9], which is the central ingredient for obtaining a tradeoff between space and query cost. Nevertheless, some new ideas were deployed, as summarized below.

The first one is to construct flakes at the bit level, which led to the introduction of bit-flakes (Definition 1). This proved to be a crucial step towards eliminating the strong indivisibility assumption. Naturally, it also demands re-designing several components in flaking counting, most notably (i) the approach described in the proof of Lemma 1 for collecting sufficiently many disjoint bit-flakes from a query, and (ii) in Lemma 2 bounding the number of canonical bit-flakes per block with respect to the number t of bits participating in at least one canonical bit-flake.

The second idea is to decide n in such a way that every query with output size n incurs in $O(nL/B)$ I/Os (see Equation 3). In retrospect, the idea sounds fairly reasonable. It forces the n points retrieved by each query to be stored in a compact manner. That is, they must be covered by asymptotically the minimum of blocks, noticing that nL/B blocks are compulsory for their storage. As a result, these blocks do not contain much information useful for answering other queries. Intuitively, the effect is that, the data structure must pack all the N points in NL/B blocks just to answer row queries, pack them again in another NL/B blocks for column queries, and yet again for square queries. Hence, the redundancy needs to be roughly 3. Of course, for the above idea to work, queries should have small overlaps in their results. It turned out that an overlap of no more than $\sqrt{n}$ points suffices.

The third idea was applied in Lemma 2, which replaced *Johnson's bound* in [9] (see Theorem 5.3 there). In fact, applying Johnson's bound in Lemma 2 would tell us that the number of canonical bit-flakes in b is at most $s/(wL\sqrt{n})$. This is quite different from what we have in Lemma 2, and does not seem to be tight enough for establishing our final result. At a higher level, the cause of

the ineffectiveness behind Johnson's bound here is that, in general, the bound can be loose when there are only a small number of canonical bit-flakes. This indeed happens in our proof, because the size s of a canonical bit-flake can be large (the value of c in Lemma 3 is close to 1 for small ϵ and μ).

Finally, Lemma 3 is what really turns the flake-counting method into a working argument. The way that s is decided is specific to our context, and does not have a counterpart in [9].

Non-Integer $\sqrt{n}$. In this case, set $n' = (\lceil\sqrt{n}\rceil)^2$. Clearly, $n' = \Theta(n)$. It is easy to adapt our proof to work instead with $N = (n')^2$ points forming an $n' \times n'$ grid. We will do so explicitly in the next section.

4 Indexability Theorem for General $r \geq 2$

This section serves as a complete proof of Theorem 1. Our argument is analogous to the one in the previous section, but includes extra details for handling general r. Remember that r is a constant integer at least 2.

As before, we consider a set P of points forming an $n \times n$ grid (i.e., $N = n^2$), where the value of n makes the query cost bounded by $O(nL/B)$ when the output size is $K = n$. Recall that the structure under our analysis has query cost $O((NL/B)^{\frac{1}{r}-\epsilon} + KL/B)$. We choose:

$$n = \left\lceil (n_0)^{1/(r-1)} \right\rceil^{r-1} \tag{8}$$

where

$$n_0 = B^{\frac{\epsilon+(r-1)/r}{2\epsilon+(r-2)/r}(1-\mu)}.$$

Note that, for $\epsilon \in (0, 1/r]$, it holds that $1 \leq \frac{\epsilon+(r-1)/r}{2\epsilon+(r-2)/r} < \frac{r-1}{r-2}$. Our choice ensures that $n^{1/(r-1)}$ is an integer.

The next lemma is rudimentary:

Lemma 4. *The following are true:*

$$\begin{aligned} n &= \Theta(n_0) \\ \left(\frac{n^2L}{B}\right)^{\frac{1}{r}-\epsilon} &= \Theta\left(\frac{nL}{B}\right) \\ nL/B &\geq 1. \end{aligned}$$

Proof. Set $x = \lceil (n_0)^{1/(r-1)} \rceil$. Hence, $n_0 > (x-1)^{r-1}$, and $n = x^{r-1}$. Consider sufficiently large B so that $x \geq 2$. In this case:

$$n \leq (2(x-1))^{r-1} < 2^{r-1} n_0 = O(n_0).$$

Then, $n = \Theta(n_0)$ follows from the obvious fact that $n \geq n_0$.

Since (as can be easily verified) $(\frac{(n_0)^2L}{B})^{(1/r)-\epsilon} = n_0 L/B$, we know:

$$\begin{aligned} \left(\frac{n^2L}{B}\right)^{\frac{1}{r}-\epsilon} &= \Theta\left(\left(\frac{(n_0)^2L}{B}\right)^{\frac{1}{r}-\epsilon}\right) \\ &= \Theta\left(\frac{n_0 L}{B}\right) = \Theta\left(\frac{nL}{B}\right). \end{aligned}$$

Finally, $nL/B \geq n_0 L/B = (\frac{(n_0)^2L}{B})^{(1/r)-\epsilon} \geq 1$ because $\epsilon \leq 1/r$. □

It follows that $\log n = \Theta(\log n_0) = \Theta(\log B)$—recall that r and μ are constants, while $\frac{\epsilon+(r-1)/r}{2\epsilon+(r-2)/r} = \Theta(1)$. The subsequent analysis concentrates on queries with output size n. Every such query can be answered in cost

$$O\left(\left(\frac{n^2L}{B}\right)^{\frac{1}{r}-\epsilon} + \frac{nL}{B}\right)$$

which is $O(nL/B)$ by Lemma 4. Hence, we can assume that its cost is no more than $\alpha nL/B$ for some constant $\alpha > 0$.

An axis-parallel rectangle is said to have *size* $l_x \times l_y$ if it covers an $l_x \times l_y$ subgrid of the grid underlying P. We consider r query sets, referred to as *set* 0, ..., *set* $r-1$ respectively, each of which consists of n queries. Specifically, the queries in set i ($0 \leq i \leq r-1$) have the same size

$$n^{\frac{i}{r-1}} \times n^{\frac{r-1-i}{r-1}}$$

are pair-wise disjoint, and together cover the entire grid. The total number of queries from all r sets is rn.

We still use Definition 1 to define *bit-flake*. Lemma 1 still holds in our current context. Furthermore, define a *canonical flake* in the same way as in Section 3. Hence, the nr queries constructed earlier give rise to at least

$$nr\left(\frac{nwL}{s} - \frac{\alpha nL}{B}\right)$$

canonical bit-flakes. Lemma 2, however, no longer holds, so we provide its counterpart:

Lemma 5. *Let b be a block, and t the number of information bits in b that appear in at least one canonical bit-flake. Then, b can contain at most*

$$\frac{t}{s} + \frac{n^{(r-2)/(r-1)} \cdot r(r-1) \cdot LB^2w^3}{2s^3}$$

canonical bit-flakes.

Proof. Recall that our queries are divided into set 0, ..., set $r-1$, each of which contains queries with the same size. Let F_i ($i \in [0, r-1]$) be the set of canonical bit-flakes that are contained in b, and defined from queries of set i. The bit-flakes in each F_i are mutually disjoint. The total number of information bits that appear in at least one bit-flake in $F_0 \cup ... \cup F_{r-1}$ is t. By the set exclusion-inclusion principle, we have:

$$\begin{aligned}\sum_{f \in (F_0 \cup ... \cup F_{r-1})} |f| - \sum_{i,j \text{ s.t. } i \neq j} \sum_{f_1 \in F_i, f_2 \in F_j} |f_1 \cap f_2| &\leq \left| \bigcup_{f \in (F_0 \cup ... \cup F_{r-1})} f \right| \\ &= t. \end{aligned} \tag{9}$$

We now show that

$$|f_1 \cap f_2| \leq wL \cdot n^{(r-2)/(r-1)} \tag{10}$$

for any $f_1 \in F_i$ and $f_2 \in F_j$ with $i \neq j$. Without loss of generality, suppose $i < j$. Let q_1 (q_2) be the query from which f_1 (f_2) was defined. In other words, q_1 and q_2 have sizes $n^{i/(r-1)} \times n^{(r-1-i)/(r-1)}$ and $n^{j/(r-1)} \times n^{(r-1-j)/(r-1)}$, respectively. Let $l_x \times l_y$ be the size of $q_1 \cap q_2$. It holds that

$$\begin{aligned} l_x &\leq n^{i/(r-1)} \\ l_y &\leq n^{(r-1-j)/(r-1)}. \end{aligned}$$

Therefore, $q_1 \cap q_2$ covers at most

$$l_x \cdot l_y \leq n^{(r-1-j+i)/(r-1)} \leq n^{(r-2)/(r-1)}$$

points of P. As $f_1 \cap f_2$ is a subset of the information bits belonging those points, (10) follows from the fact that a point's information field has wL bits.

Hence, (9) leads to:

$$\sum_{f \in (F_0 \cup ... \cup F_{r-1})} |f| - \sum_{i,j \text{ s.t. } i \neq j} \sum_{f_1 \in F_i, f_2 \in F_j} \left(wL \cdot n^{(r-2)/(r-1)} \right) \leq t.$$

The disjointness of the bit-flakes in F_i implies that $|F_i| \leq t/s \leq Bw/s$, with which the above inequality gives

$$\begin{aligned} & \sum_{f \in (F_0 \cup ... \cup F_{r-1})} |f| - \sum_{i,j \text{ s.t. } i \neq j} \left(wL \cdot n^{(r-2)/(r-1)} \cdot \frac{B^2 w^2}{s^2} \right) \leq t \\ \Rightarrow & \sum_{f \in (F_0 \cup ... \cup F_{r-1})} |f| - n^{(r-2)/(r-1)} \cdot \frac{r(r-1)}{2} \cdot \frac{LB^2 w^3}{s^2} \leq t. \end{aligned}$$

As $|f| = s$ for every $f \in F_0 \cup ... \cup F_{r-1}$, we arrive at:

$$|F_0 \cup ... \cup F_{r-1}| \leq \frac{t}{s} + n^{(r-2)/(r-1)} \cdot \frac{r(r-1)}{2} \cdot \frac{LB^2 w^3}{s^3}$$

completing the proof. □

By the trick explained in Section 3, it suffices to consider that the underlying structure uses $O(n^2 L/B)$ space, namely, $\lambda \leq \beta n^2 L/B$ blocks for some constant $\beta > 0$. Define t_i $(1 \leq i \leq \lambda)$ as the number of bits that are stored in the i-th block, and appear in at least one canonical bit-flake. The previous lemma, combined with the fact that there are at least $nr(\frac{nwL}{s} - \frac{\alpha nL}{B})$ canonical flakes, shows:

$$\sum_{i=1}^{\lambda} \left(\frac{t_i}{s} + \frac{n^{(r-2)/(r-1)} \cdot r(r-1) \cdot LB^2 w^3}{2s^3} \right) \geq nr \left(\frac{nwL}{s} - \frac{\alpha nL}{B} \right).$$

Hence:

$$\begin{aligned} \sum_{i=1}^{\lambda} t_i &\geq nsr \left(\frac{nwL}{s} - \frac{\alpha nL}{B} \right) - \lambda \cdot \frac{n^{(r-2)/(r-1)} \cdot r(r-1) \cdot LB^2 w^3}{2s^2} \\ &\geq n^2 rwL - \frac{\alpha n^2 rsL}{B} - \frac{\beta n^2 L}{B} \cdot \frac{n^{(r-2)/(r-1)} \cdot r(r-1) \cdot LB^2 w^3}{2s^2} \\ &= n^2 rwL \left(1 - \frac{\alpha s}{Bw} - \frac{(r-1)\beta \cdot BLw^2 \cdot n^{(r-2)/(r-1)}}{2s^2} \right). \end{aligned} \quad (11)$$

Lemma 6. *We can set $s = B^c$ for some $c > 0$ such that both $\frac{\alpha s}{Bw}$ and $\frac{(r-1)\beta \cdot BLw^2 \cdot n^{(r-2)/(r-1)}}{2s^2}$ are $o(1)$ when B is large enough.*

Proof. First, note that $w = \Theta(\log N) = \Theta(\log n) = \Theta(\log B)$. With $s = B^c$, we have

$$\frac{\alpha s}{Bw} = O\left(\frac{B^c}{B \log B}\right)$$

which is $o(1)$ when:

$$c \leq 1. \tag{12}$$

Applying $L = B^\mu$, (8) and Lemma 4, we know:

$$\frac{(r-1)\beta \cdot BLw^2 \cdot n^{(r-2)/(r-1)}}{2s^2} = O\left(\frac{B^{1+\mu+\frac{\epsilon+(r-1)/r}{2\epsilon+(r-2)/r}\frac{r-2}{r-1}(1-\mu)} \cdot \log^2 B}{B^{2c}}\right)$$

which is $o(1)$ when:

$$1 + \mu + \frac{\epsilon + (r-1)/r}{2\epsilon + (r-2)/r} \cdot \frac{r-2}{r-1} \cdot (1-\mu) < 2c. \tag{13}$$

A value of c satisfying (12) and (13) exists when:

$$\begin{aligned} 1 + \mu + \frac{\epsilon + (r-1)/r}{2\epsilon + (r-2)/r} \cdot \frac{r-2}{r-1} \cdot (1-\mu) &< 2 \\ \Leftrightarrow \frac{\epsilon r + (r-1)}{2\epsilon r + (r-2)} \cdot \frac{r-2}{r-1} &< 1 \\ \Leftrightarrow \epsilon r(r-2) &< 2\epsilon r(r-1) \\ \Leftrightarrow r - 2 &< 2r - 2 \end{aligned}$$

which is always true (recall that $r \geq 2$). □

Therefore, setting s as in the above lemma, (11) becomes

$$\sum_{i=1}^{\lambda} t_i \geq n^2 rwL(1 - o(1)) = rNwL(1 - o(1))$$

concluding the proof of Theorem 1.

5 *r*-Redundant Structures

Next, we present r-redundant structures achieving the performance in Theorem 2. For simplicity, we assume that the points in the input set P are in general position, such that no two points in P share the same x- or y-coordinate. This assumption can be removed by standard tie-breaking techniques.

Preliminary: External Interval Tree. This structure, due to Arge and Vitter [3], settles the following *stabbing problem*. The input set consists of N intervals in the real domain. Given a real value q, a *stabbing query* reports all the data intervals enclosing q. The external interval tree consumes $O(N/B)$ space, and answers a stabbing query in $O(\log_B N + K/B)$ I/Os, where K is the number of reported intervals.

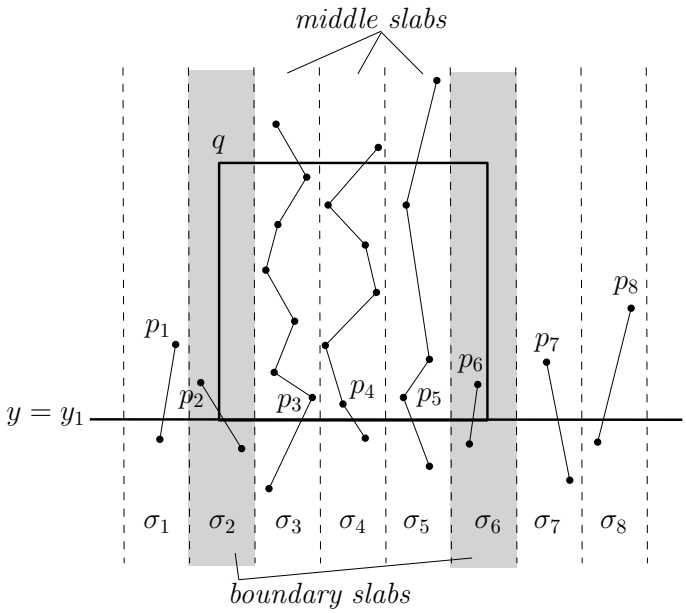


Figure 2: Answering a range query ($\rho = 8$)

The First 1-Redundant Structure. Let us start by giving a 1-redundant structure occupying $NL/B + \sqrt{NL/B} + O(N/B)$ space, and solving a query in $O(\sqrt{NL/B} + KL/B)$ I/Os. Note that the space cost is worse than required in Theorem 2. The structure in fact has been used as a component in the *range search tree* [6] and its external counterpart [2]. Our version here differs only in parameterization (as will be pointed out shortly). Nevertheless, we describe it in full because the details are useful in clarifying general r-redundant structures later.

We introduce a parameter $\rho = \sqrt{NL/B}$. Also, define a *slab* as the part of data space $\mathbb{R}^2$ between and including two vertical lines $x = c_1$ and $x = c_2$. Partition the x-axis into ρ segments such that P has $\lceil N/\rho \rceil$ points whose x-coordinates fall in each segment, except possibly the last segment. Denote by P_i ($1 \leq i \leq \rho$) the set of points in the i-th segment, counting from left to right. Denote by σ_i the slab corresponding to the i-segment.

For each P_i, sort its points in ascending order of y-coordinate. From now on, we will treat P_i as a sorted list, abusing the notation slightly. Naturally, the *j-th point of P_i* refers to the j-th point in the sorted list. If $2 \leq j \leq |P_i|$, the *predecessor* of the j-th point is the $(j-1)$-st point. As a special case, the predecessor of the first point in P_i is defined as a dummy point whose y-coordinate is $-\infty$. The x-coordinate of the dummy point is unimportant.

Define $L' = L + 2$. We store the coordinates and information fields of the points of P_i in an array A_i (respecting the ordering of those points). A_i thus occupies at most $1 + L'|P_i|/B$ blocks, i.e., all but the last block contains B words of data. Hence, arrays $A_1, ..., A_\rho$ require in total at most $\rho + NL'/B = \sqrt{NL/B} + NL/B + O(N/B)$ blocks. We are still allowed $O(N/B)$ extra space, which is sufficient to store the *coordinates* of each point constant times. As discussed below, we use that space to create an external interval tree $\mathcal{T}$.

$\mathcal{T}$ is built on N one-dimensional intervals obtained as follows. From each P_i, we generate a set I_i of $|P_i|$ intervals. Each point $p \in P_i$ *determines* an interval in I_i having the form $(y_{pred}, y_p]$, where y_p is the y-coordinate of p, and y_{pred} is that of its predecessor. We associate this interval with a pointer to the block of array A_i where $info(p)$ is stored, so that when the interval is fetched, we can jump to that block in one I/O. $\mathcal{T}$ indexes the union of $I_1, ..., I_\rho$. As $\mathcal{T}$ uses $O(N/B)$ blocks,

the overall space of our structure is $NL/B + \sqrt{NL/B} + O(N/B)$.

To answer a range query with search region $q = [x_1, x_2] \times [y_1, y_2]$, we first identify the (at most) two *boundary slabs* that contain the left and right edges of q, respectively (this takes $O(\log_B N)$ I/Os with another B-tree on the slabs' x-ranges). Denote them respectively as σ_{i_1} and σ_{i_2} with $i_1 \leq i_2$. Scan arrays A_{i_1} and A_{i_2} completely to report the information fields of the qualifying points there. Since each array has at most $\lceil N/\rho \rceil$ points, the cost of the scan is

$$O\left(\frac{NL'}{\rho B}\right) = O(\sqrt{NL/B}).$$

In the example of Figure 2 (where $\rho = 8$), the boundary slabs are σ_2 and σ_6, in which all the points are examined.

The other qualifying points can lie only in slabs σ_i where i ranges from $i_1 + 1$ to $i_2 - 1$. Refer to those slabs as the *middle slabs*. For each such slab σ_i, we find the lowest point p_i in σ_i on or above the horizontal line $y = y_1$. After this, jump to the block in A_i where $info(p_i)$ is stored, and start scanning A_i from p_i to retrieve the information fields of the other points above the line $y = y_1$. We do so in ascending order of those points' y-coordinates, so that the scan can terminate as soon as encountering a point falling out of q. All the points already scanned prior to this moment are the only points in σ_i satisfying q. If the number of them is K_i, the scan performs at most $O(1+K_iL'/B)$ I/Os. Hence, carrying out the scan in all slabs σ_i ($i \in [i_1 + 1, i_2 - 1]$) takes $O(\rho + K_{mid}L'/B)$ I/Os, where K_{mid} is the number of qualifying points from the middle slabs. In Figure 2, the middle slabs are σ_3, σ_4 and σ_5. The scan in σ_3, for instance, starts from p_3, and ends at the lowest point in σ_3 above the query rectangle.

It remains to explain how to find all the p_i for each $i \in [i_1 + 1, i_2 - 1]$. This can be settled with the external interval tree $\mathcal{T}$. Recall that p_i determines an interval in I_i. The definition of p_i makes that interval the only one from I_i that contains the value y_1. Hence, it can be found by a stabbing query on $\mathcal{T}$ with y_1 as the search value. Note that this stabbing query may retrieve an interval for every slab, including those that are not a middle slab. In Figure 2, for example, the stabbing query returns (the intervals determined by) 8 points: $p_1, ..., p_8$. Nevertheless, as there are only ρ slabs, the stabbing query finishes in $O(\log_B N + \rho/B)$ I/Os, after which we can keep only the points in the middle slabs and discard the others. Therefore, overall the cost of reporting the qualifying points in the middle slabs is bounded by

$$O(K_{mid}L'/B + \rho + \log_B N + \rho/B) \quad = \quad O(\sqrt{NL/B} + K_{mid}L/B). \tag{14}$$

As analyzed earlier, the qualifying points from the boundary slabs can be found in $O(\sqrt{NL/B})$ I/Os. Hence, the total query cost is $O(\sqrt{NL/B} + KL/B)$.

As mentioned, this structure has been used in [2, 6]. The only nuance in our scenario is the choice of ρ (which was logarithmic in [2, 6]). The techniques in the rest of the section, on the other hand, are newly developed in this paper.

Reducing the Space. The previous structure incurs more space than our target in Theorem 2 by an additive term of $\rho = \sqrt{NL/B}$. This term can be eliminated with a trick we call *tail collection*, as explained next.

A close look at our earlier description reveals that the extra term ρ exists because each A_i ($1 \leq i \leq \rho$) may have an *under-full* block, which does not have B words of data, and thus, wastes space. This under-full block, if present, must be the last block in A_i. We remove it from A_i, after which all the blocks in A_i are fully utilized. We concatenate the data of all the non-full blocks (from different arrays) into a separate *tail file* G. All blocks in G store B words of data, except

possibly one. Therefore, the total space used by G and the arrays is now at most $1 + NL/B$, i.e., ρ blocks less than before.

Note that G itself has no more than ρ blocks. Therefore, we can afford to scan it completely in answering a query, which will add only ρ I/Os to the query cost, and hence, does not change the query complexity $O(\sqrt{NL/B} + KL/B)$. The query algorithm is still the same as before, except that in scanning an array, if we have come to its end and see that some information has been moved to G, we should continue scanning the relevant portion in G.

***r*-Redundant Structure.** Assuming that there is an $(r-1)$-redundant structure with space $(r-1)NL/B+O(N/B)$ and query cost $O((NL/B)^{1/r}+KL/B)$, next we will obtain an r-redundant structure with space $rNL/B + O(N/B)$ and query cost $O((NL/B)^{1/(r+1)} + KL/B)$. This can be done by modifying the earlier 1-redundant structure, as shown below.

The first change is the value of ρ, which is now set to $(NL/B)^{1/(r+1)}$. Then, in the same manner as in the 1-redundant case, we divide P into $P_1, ..., P_\rho$, and create arrays $A_1, ..., A_\rho$, the tail file G, and the external interval tree $\mathcal{T}$. Currently, the information field of each point has been stored once, such that the space consumption is $NL/B + O(N/B)$. On each P_i $(1 \leq i \leq \rho)$, we build an $(r-1)$-redundant structure T_i, which occupies $(r-1)|P_i|L/B + O(|P_i|/B)$ space. Hence, $T_1, ..., T_\rho$ together use

$$\sum_{i=1}^{\rho} \frac{(r-1)|P_i|L}{B} + O\left(\frac{|P_i|}{B}\right) = \frac{(r-1)NL}{B} + O(N/B)$$

space. This explains why the overall space is $rNL/B + O(N/B)$. Note that the final structure is r-redundant.

To answer a range query $q = [x_1, x_2] \times [y_1, y_2]$, as in the 1-redundant case, we start by identifying the boundary slabs σ_{i_1} and σ_{i_2} $(i_1 \leq i_2)$. Recall that they define middle slabs σ_i for each $i \in [i_1 + 1, i_2 - 1]$. The information fields of the qualifying points in the middle slabs are retrieved in the same way as in the 1-redundant case, i.e., utilizing $\mathcal{T}$, the arrays of the middle slabs, and perhaps also the tail file. If K_{mid} points from the middle slabs satisfy the query, as shown in (14), they can be extracted in $O(\rho + K_{mid}L/B) = O((NL/B)^{1/(r+1)} + K_{mid}L/B)$ I/Os.

Finally, to report the qualifying points in the boundary slabs, we query the $(r-1)$-redundant structures T_{i_1} and T_{i_2}. Notice that each of these structures indexes at most $\lceil N/\rho \rceil$ points. Hence, if K_1 and K_2 points are found from T_{i_1} and T_{i_2} respectively, searching the two structures entails

$$\begin{aligned} O\left(\left(\frac{NL}{\rho B}\right)^{1/r} + (K_1+K_2)L/B\right) &= O\left(\left(\frac{NL}{B}\right)^{\frac{r}{r+1}\cdot\frac{1}{r}} + \frac{(K_1+K_2)L}{B}\right) \\ &= O\left(\left(\frac{NL}{B}\right)^{1/(r+1)} + \frac{(K_1+K_2)L}{B}\right) \end{aligned}$$

I/Os. As $K_{mid} + K_1 + K_2 = K$, the overall query cost is $O((NL/B)^{1/(r+1)} + KL/B)$.

Combining the above inductive construction with our earlier 1-redundant structure, we have established the first part of Theorem 2.

A Better Structure for $L = \Omega(B)$. The peculiarity at $L = \Omega(B)$ arises from the fact that, the cost of reporting $K \geq 1$ points is $\Omega(KL/B) = \Omega(K)$, namely, the query algorithm can afford to spend one I/O on each qualifying point. Imagine that we store the information fields of all points in an array, indexed by point ids. The array uses $NL/B + O(1)$ blocks. Then, create on P an $O(N/B)$-space structure designed for orthogonal range search with $L = 1$, treating each id

as an information field. This structure allows us to report the ids of the points satisfying a query in $O((N/B)^\epsilon + K/B)$ I/Os, after which their information fields can be extracted from the array in $O(K + KL/B) = O(KL/B)$ I/Os. We thus have obtained a 1-redundant structure with space $NL/B + O(N/B)$ and query time $O((N/B)^\epsilon + KL/B)$.

We now conclude the whole proof of Theorem 2.

6 Conclusions

This paper revisited linear-space data structures for the 2D orthogonal range search problem in external memory. The primary objective is to understand the best achievable query efficiency when the information of each point can be stored at most $r \geq 1$ times, when r is a given constant. Interestingly, we showed that the length L (measured in words) of the information field carried by each point plays a crucial role in this problem. Specifically, for $L = O(B^\mu)$ where μ is a positive constant smaller than 1, the best query cost is $O((NL/B)^{\frac{1}{r+1}})$ plus the linear cost of outputting the information fields, whereas for $L = \Omega(B)$, even with $r = 1$ it is possible to achieve query cost $O((N/B)^\epsilon)$ (plus the linear output cost) for arbitrarily small constant $\epsilon > 0$. Our argument also showed that a structure gains no extra power by dividing the information fields and storing the bits therein separately, as long as those bits must be retrieved from the disk for reporting, instead of being computed.

Acknowledgements

This research was supported in part by GRF grants 4164/12, 4168/13, and 142072/14.

References

[1] A. Aggarwal and J. S. Vitter. The input/output complexity of sorting and related problems. *Communications of the ACM (CACM)*, 31(9):1116–1127, 1988.

[2] L. Arge, V. Samoladas, and J. S. Vitter. On two-dimensional indexability and optimal range search indexing. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 346–357, 1999.

[3] L. Arge and J. S. Vitter. Optimal external memory interval management. *SIAM Journal of Computing*, 32(6):1488–1508, 2003.

[4] N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An efficient and robust access method for points and rectangles. In *Proceedings of ACM Management of Data (SIGMOD)*, pages 322–331, 1990.

[5] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM (CACM)*, 18(9):509–517, 1975.

[6] B. Chazelle. Filtering search: A new approach to query-answering. *SIAM Journal of Computing*, 15(3):703–724, 1986.

[7] R. Grossi and G. F. Italiano. Efficient splitting and merging algorithms for order decomposable problems. *Information and Computation*, 154(1):1–33, 1999.

[8] A. Guttman. R-trees: a dynamic index structure for spatial searching. In *Proceedings of ACM Management of Data (SIGMOD)*, pages 47–57, 1984.

[9] J. M. Hellerstein, E. Koutsoupias, D. P. Miranker, C. H. Papadimitriou, and V. Samoladas. On a model of indexability and its bounds for range queries. *Journal of the ACM (JACM)*, 49(1):35–55, 2002.

[10] J. M. Hellerstein, E. Koutsoupias, and C. H. Papadimitriou. On the analysis of indexing schemes. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 249–256, 1997.

[11] J. Iacono and M. Patrascu. Using hashing to solve the dictionary problem (in external memory). To appear in *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2012.

[12] K. V. R. Kanth and A. 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] E. Koutsoupias and D. S. Taylor. Tight bounds for 2-dimensional indexing schemes. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 52–58, 1998.

[14] O. Procopiuc, P. K. Agarwal, L. Arge, and J. S. Vitter. Bkd-tree: A dynamic scalable kd-tree. In *Proceedings of Symposium on Advances in Spatial and Temporal Databases (SSTD)*, pages 46–65, 2003.

[15] J. T. Robinson. The K-D-B-tree: A search structure for large multidimensional dynamic indexes. In *Proceedings of ACM Management of Data (SIGMOD)*, pages 10–18, 1981.

[16] V. Samoladas and D. P. Miranker. A lower bound theorem for indexing schemes and its application to multidimensional range queries. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 44–51, 1998.

[17] M. Streppel and K. Yi. Approximate range searching in external memory. *Algorithmica*, 59(2):115–128, 2011.

[18] S. Subramanian and S. Ramaswamy. The p-range tree: A new data structure for range searching in secondary memory. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 378–387, 1995.

[19] E. Verbin and Q. Zhang. The limits of buffering: a tight lower bound for dynamic membership in the external memory model. In *Proceedings of ACM Symposium on Theory of Computing (STOC)*, pages 447–456, 2010.

[20] K. Yi and Q. Zhang. On the cell probe complexity of dynamic membership. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 123–133, 2010.