Optimal External Memory Planar Point Enclosure

Lars Arge[1⋆], Vasilis Samoladas[2], and Ke Yi[1⋆]

[1] Department of Computer Science, Duke University, Durham, NC 27708, USA.
{large,yike}@cs.duke.edu
[2] Technical University of Crete, Greece. vsam@softnet.tuc.gr

Abstract. In this paper we study the external memory planar point enclosure problem: Given N axis-parallel rectangles in the plane, construct a data structure on disk (an index) such that all K rectangles containing a query point can be reported I/O-efficiently. This problem has important applications in e.g. spatial and temporal databases, and is dual to the important and well-studied orthogonal range searching problem. Surprisingly, we show that one cannot construct a linear sized external memory point enclosure data structure that can be used to answer a query in $O(\log_B N + K/B)$ I/Os, where B is the disk block size. To obtain this bound, $\Omega(N/B^{1-\epsilon})$ disk blocks are needed for some constant $\epsilon > 0$. With linear space, the best obtainable query bound is $O(\log_2 N + K/B)$. To show this we prove a general lower bound on the tradeoff between the size of the data structure and its query cost. We also develop a family of structures with matching space and query bounds.

1 Introduction

In this paper we study the external memory planar *point enclosure* problem: Given N axis-parallel rectangles in the plane, construct a data structure on disk (an index) such that all K rectangles containing a query point q can be reported I/O-efficiently. This problem is the dual of the orthogonal range searching problem, that is, the problem of storing a set of points in the plane such that the points in a query rectangle can be reported I/O-efficiently. The point enclosure problem has important applications in temporal databases; for example, if we have a database where each object is associated with a time span and a key range, retrieving all objects with key ranges containing a specific key value at a certain time corresponds to a point enclosure query. Also, in spatial databases, irregular planar objects are often represented in a data structure by their minimal bounding boxes; retrieving all bounding boxes that contain a query point, i.e. a point enclosure query, is then the first step in retrieving all objects that contain the query point. Point enclosure can also be used as part of an algorithm for finding all bounding boxes intersecting a query rectangle, since such a query can be decomposed into a point enclosure query, a range query, and

⋆ Supported in part by the National Science Foundation through RI grant EIA–9972879, CAREER grant CCR–9984099, ITR grant EIA–0112849, and U.S.–Germany Cooperative Research Program grant INT–0129182.

two segment intersection queries. While a lot of work has been done on developing worst-case efficient external memory data structures for range searching problems (see e.g. [3, 17] for surveys), the point enclosure problem has only been considered in some very restricted models of computation [1, 14]. In this paper we prove a general lower bound on the tradeoff between the size of an external memory point enclosure structure and its query cost, and show that this tradeoff is asymptotically tight by developing a family of structures with matching space and query bounds. Surprisingly, our results show that the point enclosure problem is harder in external memory than in internal memory.

1.1 Previous work

The B-tree [13] is the most fundamental external memory data structure. It uses $O(N/B)$ disk blocks of size B to store N elements and can be used to answer a one-dimensional range query in $O(\log_B N + K/B)$ I/Os; an I/O is the movement of one disk block between disk and main memory. These bounds are optimal in comparison-based models of computation, and they are the bounds we would like to obtain for more complicated external memory data structure problems. In the study of such problems, a number of different lower bound models have been developed in recent years: the *non-replicating index* model [21], the *external memory pointer machine* model [25], the *bounding-volume hierarchy* model [1], and the *indexability* model [20, 19]. The most general of these models is the indexability model of Hellerstein, Koutsoupias, and Papadimitriou [20, 19]. To our knowledge, it captures all known external data structures for reporting problems (that is, problems where the output is a subset of the objects in the structure). In this model, the focus is on bounding the number of disk blocks containing the answers to a query q, given a bound on the number of blocks required by the data structure. The cost of computing what blocks to access to answer the query (the *search cost*) is ignored. More formally, an instance of a problem is described by a workload W, which is a simple hypergraph $(\mathcal{O}, \mathcal{Q})$, where $\mathcal{O}$ is the set of N objects, and $\mathcal{Q}$ is a set of subsets of $\mathcal{O}$. The elements of $\mathcal{Q} = \{q_1, \ldots, q_m\}$ are called queries. An *indexing scheme* $\mathcal{S}$ for a given workload $W = (\mathcal{O}, \mathcal{Q})$ is a B-regular hypergraph $(\mathcal{O}, \mathcal{B})$, where each element of $\mathcal{B}$ is a B-subset of $\mathcal{O}$ (called a block), and where $\mathcal{O} = \cup\mathcal{B}$. An indexing scheme $\mathcal{S}$ can be thought of as a placement of the objects of $\mathcal{O}$ on disk pages, possibly with redundancy. The cost of answering a query q is $\min\{|\mathcal{B}'| \mid \mathcal{B}' \subseteq \mathcal{B}, q \subseteq \bigcup \mathcal{B}'\}$, i.e., the minimum number of blocks whose union contains all objects in the query. The efficiency of an indexing scheme is measured using two parameters: its *redundancy* r and its *access overhead* A. The redundancy r is defined to be the average number of copies of an object stored in the indexing scheme, i.e. $r = B|\mathcal{B}|/N$, and the access overhead A is defined to be the worst-case ratio between the cost of a query and the ideal cost $\lceil |q|/B \rceil$ (where $|q|$ denotes the query output size). In other words, an indexing scheme with redundancy r and access overhead A occupies $r\lceil N/B \rceil$ disk blocks and any query q is covered by at most $A\lceil |q|/B \rceil$ disk blocks.

Following the introduction of the indexability model, a sequence of results by Koutsoupias and Taylor [22], Samoladas and Miranker [24] and Arge, Samoladas, and Vitter [6] proved a tradeoff of $r = \Omega(\log(N/B)/\log A)$ for the (two-dimensional) orthogonal range searching problem in the model. Using this tradeoff, Arge, Samoladas and Vitter [6] proved that $\Omega(\frac{N}{B}\frac{\log(N/B)}{\log\log_B N})$ space is needed to design an indexing scheme that answers queries in $O(\log_B N + K/B)$ I/Os, i.e., that two-dimensional range searching is harder than the one-dimensional case. They also designed a structure with matching bounds; note that this structure works in the classical I/O-model [2] where the search cost *is* considered. A similar bound was proven in the external memory pointer machine model by Subramanian and Ramaswamy [25], and the result resembles the internal memory pointer machine case, where $\Theta(N \log N/\log\log N)$ space is needed to obtain $O(\log N+K)$ query cost [11]. If only $O(N/B)$ space can be used, $\Theta((N/B)^\epsilon)$ I/Os are needed to answer a query [6]. If exactly $\lceil N/B \rceil$ space is allowed, $\Theta(\sqrt{N/B})$ I/Os are needed [21].

Compared to orthogonal range searching, relatively little is known about the dual point enclosure problem. Arge and Vitter [7] designed a linear space external interval tree structure for the one-dimensional version of the problem where the data is intervals. This structure answers point enclosure queries, also called *stabbing queries*, in $O(\log_B N + K/B)$ I/Os. The two-dimensional version of the problem we consider in this paper can be solved using the linear space R-tree [18] or its variants (see e.g. [17] for a survey); very recently Arge et al. [5] designed a variant that answers a query in worst-case $O(\sqrt{N/B} + K/B)$ I/Os. This is optimal in the very restrictive bounding-volume hierarchy model [1]. However, in the internal memory pointer machine model a linear size data structure that can answer a query in the optimal $O(\log N + K)$ time has been developed [10]. Thus, based on the correspondence between internal and external memory results for the range searching problem, as well as for the one-dimensional enclosure problem, one would expect to be able to develop a linear space and $O(\log_B N + K/B)$ I/O query structure. No such structure is known.

1.2 Our results

Surprisingly, in this paper we show that in the indexability model it is *not* possible to design a linear sized point enclosure indexing scheme that can answer a query in $O(\log_B N + K/B)$ I/Os. More precisely, we show that to obtain an $O(\log_B N + K/B)$ query bound, $\Omega(N/B^{1-\epsilon})$ disk blocks are needed for some constant $\epsilon > 0$; with linear space the best obtainable query bound is $\Omega(\log_2 N + K/B)$. Thus in some sense this problem is harder in external memory than in internal memory. An interesting corollary to this result is that, unlike in internal memory, an external interval tree cannot be made partially persistent without increasing the asymptotic space or query bound. Refer to Table 1 for a summary of tight complexity results for the orthogonal range search and point enclosure problems when we are restricted to linear space or a logarithmic (base 2 or B) query bound.

In section 2 we prove our lower bounds by proving a general tradeoff between the size of a point enclosure indexing scheme and its query cost. To do so, we

Range searching	Internal memory		External memory	
Query with linear space	N^ϵ	[9]	$(N/B)^\epsilon$	[6]
Space with log (base 2 or B) query	$N\frac{\log N}{\log\log N}$	[10, 11]	$\frac{N}{B}\frac{\log(N/B)}{\log\log_B N}$	[6]

Point enclosure	Internal memory		External memory	
Query with linear space	$\log_2 N$	[10]	$\log_2(N/B)$	New
Space with log (base 2 or B) query	N	[10]	$N/B^{1-\epsilon}$	New

Table 1. Summary of asymptotically tight complexity results for the orthogonal range searching and point enclosure problems in the internal memory pointer machine model and the external memory indexability model.

refine the indexability model in a way similar to [6]: We introduce parameters A_0 and A_1 instead of A, and require that any query q can be covered by at most $A_0 + A_1\lceil|q|/B\rceil$ blocks rather than $A\lceil|q|/B\rceil$ blocks. With this refinement, we prove the lower bound tradeoff $A_0A_1^2 = \Omega(\log(N/B)/\log r)$, which leads to the results mentioned above. Interestingly, if the known tradeoff for range search indexing schemes is expressed in the refined indexability model, it becomes $r = \Omega(\log(N/B)/\log(A_0 \cdot A_1))$. Thus, the tradeoffs of the dual range search and point enclosure indexing problems are symmetric with respect to r and A_0, A_1.

In Section 3 we show that our lower bound for the tradeoff is asymptotically tight in the most interesting case $A_1 = O(1)$ by developing a family of external memory data structures (where the search cost is considered) with optimal space and query cost. More precisely, we describe a structure that, for any $2 \le r \le B$, uses $O(rN/B)$ disk blocks and answers queries in $O(\log_r(N/B) + K/B)$ I/Os. The structure can be constructed in $O(r\frac{N}{B}\log_{M/B}\frac{N}{B})$ I/Os, where M is the size of main memory.

2 Lower Bounds

2.1 Refined Redundancy Theorem

The Redundancy Theorem of [24, 19] is the main tool in most indexability model lower bound results. We develop a version of the theorem for the refined indexability model, that is, the model where the access overhead A has been replaced by two parameters A_0 and A_1, such that a query is required to be covered by at most $A_0 + A_1\lceil|q|/B\rceil$ blocks.

Theorem 1 (Redundancy Theorem [24, 19]). *For a workload $W = (\mathcal{O}, \mathcal{Q})$, where $\mathcal{Q} = \{q_1, q_2, \ldots, q_m\}$, let $\mathcal{S}$ be an indexing scheme with access overhead $A \le \sqrt{B}/4$ such that for any $1 \le i, j \le m, i \ne j$:*

$$|q_i| \ge B/2 \qquad \textit{and} \qquad |q_i \cap q_j| \le \frac{B}{16A^2},$$

then the redundancy of $\mathcal{S}$ is bounded by

$$r \ge \frac{1}{12N}\sum_{i=1}^{m}|q_i|.$$

We extend this theorem to the refined indexability model as follows.

Theorem 2 (Refined Redundancy Theorem). *For a workload $W = (\mathcal{O}, \mathcal{Q})$, where $\mathcal{Q} = \{q_1, q_2, \ldots, q_m\}$, let $\mathcal{S}$ be an indexing scheme with access overhead (A_0, A_1) with $A_1 \leq \sqrt{B}/8$ such that for any $1 \leq i, j \leq m, i \neq j$:*

$$|q_i| \geq BA_0 \quad \textit{and} \qquad |q_i \cap q_j| \leq \frac{B}{64A_1^2},$$

then the redundancy of $\mathcal{S}$ is bounded by

$$r \geq \frac{1}{12N} \sum_{i=1}^{m} |q_i|.$$

Proof. Since for any i, $|q_i| \geq BA_0$, q_i is required to be covered by at most $A_0 + A_1\lceil |q_i|/B \rceil \leq 2A_1\lceil |q_i|/B \rceil$ blocks. If we set $A = 2A_1$, $\mathcal{S}$ is an indexing scheme that covers each query of $\mathcal{Q}$ by $A\lceil |q|/B \rceil$ blocks. Then applying Theorem 1 leads to the desired result.

2.2 Lower bound for point enclosure

In order to apply Theorem 2 to the point enclosure problem we need to design a workload $W = (\mathcal{O}, \mathcal{Q})$ such that each query is sufficiently large but the intersection of any two is relatively small. To do so we use the point set called a *Fibonacci lattice*, which was also used in previous results on orthogonal range searching [22, 6, 19].

Definition 1 ([23]). *The Fibonacci lattice F_m is a set of two-dimensional points defined by $F_m = \{(i, if_{k-1} \bmod m) | i = 0, 1, \ldots, m-1)\}$, where $m = f_k$ is the kth Fibonacci number.*

The following property of a Fibonacci lattice will be crucial in our lower bound tradeoff proof.

Lemma 1 ([16]). *For the Fibonacci lattice F_m and for $\alpha \geq 0$, any rectangle with area αm contains between $\lfloor \alpha/c_1 \rfloor$ and $\lceil \alpha/c_2 \rceil$ points, where $c_1 \approx 1.9$ and $c_2 \approx 0.45$.*

Let $m = \lambda \frac{\alpha N}{BA_0}$, where α is a parameter to be determined later, and $1 \leq \lambda < 2$ is chosen such that m is a Fibonacci number. The idea is to use the points in F_m as queries to form $\mathcal{Q}$ and construct approximately N rectangles as objects to form $\mathcal{O}$; in the previous range searching tradeoff the roles of the points and rectangles were reversed [6]. We may not construct exactly N rectangles but the number will be between λN and $4\lambda N$, so this will not affect our asymptotic result.

We construct rectangles of dimensions $\alpha t^i \times m/t^i$, where $t = (m/\alpha)^{1/(BA_0)}$ and $i = 1, \ldots, BA_0$. For each choice of i, the rectangles are constructed in a tiling fashion on F_m until they cross the boundary. In this way, between $\frac{m^2}{\alpha m} = \frac{\lambda N}{BA_0}$

and $4\frac{\lambda N}{BA_0}$ rectangles are constructed on each layer (each i), for a total of $\Theta(N)$ rectangles.

Since each point is covered by BA_0 rectangles, one from each layer, the first condition of Theorem 2 is satisfied. For the second one, consider any two different query points q_1 and $q_2 \in F_m$, and let x and y be the differences between their x and y-coordinates, respectively. Since the rectangle with q_1 and q_2 as corners contains at least two points, namely q_1 and q_2 themselves, by Lemma 1, we have $xy \geq c_2 m$. We now look at how many rectangles can possibly cover both q_1 and q_2. For a rectangle with dimension $\alpha t^i \times m/t^i$ to cover both points, we must have $\alpha t^i \geq x$ and $m/t^i \geq y$, or equivalently, $x/\alpha \leq t^i \leq m/y$. rectangle for each i that can cover both points, thus, $|q_1 \cap q_2|$ is at most

$$\left\lceil \log_t \frac{\alpha m}{xy} \right\rceil \leq \left\lceil \frac{\log(\alpha/c_2)}{\log t} \right\rceil = \left\lceil BA_0 \frac{\log(\alpha/c_2)}{\log(m/\alpha)} \right\rceil \leq 1 + BA_0 \frac{\log(\alpha/c_2)}{\log(\lambda N/(BA_0))}.$$

The second condition holds as long as

$$\frac{1}{B} + A_0 \frac{\log(\alpha/c_2)}{\log(\lambda N/(BA_0))} \leq \frac{1}{64A_1^2}. \tag{1}$$

On the other hand, when (1) is satisfied, by Theorem 2, we have

$$r \geq \frac{1}{12} \frac{mBA_0}{4\lambda N} = \frac{\alpha}{48}.$$

Therefore, any combination of α, A_0 and A_1 that satisfy (1) constitutes a lower bound on the tradeoff of r, A_0 and A_1. When A_0 and A_1 are small enough, namely $A_0 < (N/B)^{1-\delta}$ where $0 < \delta < 1$ is some constant, and $A_1 < \sqrt{B/128}$, we can simplify (1) to obtain the following:

Theorem 3. *Let $\mathcal{S}$ be a point enclosure indexing scheme on the Fibonacci workload. If $A_0 \leq \left(\frac{N}{B}\right)^{1-\delta}$ for any fixed $0 < \delta < 1$, and $A_1 \leq \sqrt{B/128}$, then its redundancy r and access overhead (A_0, A_1) must satisfy*

$$A_0 A_1^2 \geq \frac{\delta}{128} \frac{\log(N/B)}{7 + \log r} = \Omega\left(\frac{\log(N/B)}{\log r}\right).$$

The Fibonacci lattice is only one of many low-discrepancy point sets [23] we could have used, but none of them would improve the result of Theorem 3 by more than a constant factor. We note that the above proof technique could also have been used to obtain a tradeoff in the original indexability model. However, in that case we would obtain a significantly less informative tradeoff of $A = \Omega(\sqrt{\log(N/B)/\log r})$.

2.3 Tradeoff implications

Query cost $\log_B N$. In indexing schemes with an $O(\log_B N + K/B)$ query cost we have $A_0 = O(\log_B N)$ and $A_1 = O(1)$. Assuming that $A_0 \leq c_0 \log_B \frac{N}{B}$, we have

$$c_0 \frac{\log(N/B)}{\log B} A_1^2 \geq \frac{\delta}{128} \frac{\log(N/B)}{7 + \log r},$$

which is

$$\log r \geq \frac{\delta}{128 c_0 A_1^2} \log B \ - 7, \text{ or } \ r \geq \frac{1}{128} B^{\delta/(128 c_0 A_1^2)}.$$

Thus any such indexing scheme must have redundancy at least $\Omega(B^\epsilon)$, for some small constant ϵ.

Corollary 1. *Any external memory data structure for the point enclosure problem that can answer a query in $O(\log_B N + K/B)$ I/Os in the worst case must use $\Omega(N/B^{1-\epsilon})$ disk blocks for some constant $\epsilon > 0$.*

Linear space. In linear space indexing schemes r is a constant. If we also want A_1 to be a constant, A_0 has to be $\Omega(\log \frac{N}{B})$ by Theorem 3. As mentioned, this is a rather surprising result, since linear space structures with the optimal $O(\log N + K)$ query time exist in internal memory [10].

Corollary 2. *Any linear sized external memory data structure for the point enclosure problem that answers a query in $f(N) + O(K/B)$ I/Os in the worst case must have $f(N) = \Omega(\log_2 \frac{N}{B})$.*

Note however that our lower bound does *not* rule out linear size indexes with a query cost of for example $O(\log_B N + \sqrt{\log B}\ \frac{K}{B})$.

Persistent interval tree. An interesting consequence of the above linear space result is that, unlike in internal memory, we cannot make the external interval tree [7] partially persistent [15] without either increasing the asymptotic space or query bound. Details will appear in the full paper.

3 Upper Bounds

In this section, we develop a family of external memory data structures with space and query bounds that match the lower bound tradeoff of Theorem 3. In Section 3.1 we first develop a structure that uses $O(rN/B)$ disk blocks and answers queries in $O(\log_B N \cdot \log_r(N/B) + K/B)$ I/Os for any $2 \leq r \leq B$. In Section 3.2 we then discuss how to improve the query bound to obtain the $O(\log_r(N/B) + K/B)$ bound matching Theorem 3. These bounds are measured in the classical I/O model [2], where the search cost is considered, and where space used to store auxiliary information (such as e.g. "directories" or "internal nodes") is also counted when bounding the sizes of the structures.

3.1 Basic point enclosure structure

Base tree T. Let S be a set of N axis-parallel rectangles in the plane. Our structure for answering point enclosure queries on S is similar to an internal memory point enclosure structure due to Chazelle [10]. Intuitively, the idea is to build an r-ary base tree[3] T by first dividing the plane into $r \leq B$ horizontal

[3] The fanout of the root of the base tree may be less than r to make sure that we have $O(N/B)$ nodes in the tree.

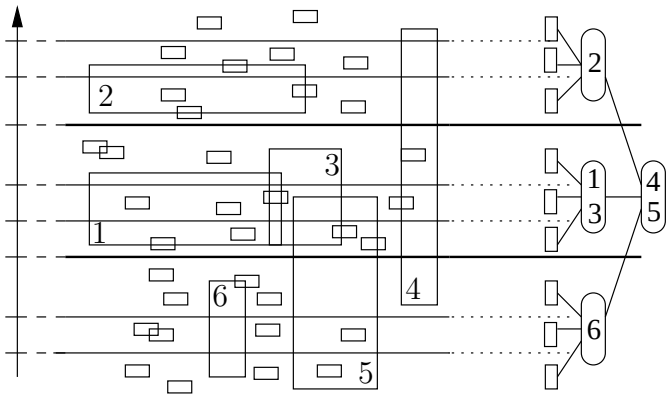

Fig. 1. Partitioning of rectangles among nodes in a 3-ary base tree.

slabs with approximately the same number of rectangle corners in each slab, and then recursively construct a tree on the rectangles completely contained in each of the slabs. The rectangles that cross one or more slab boundaries are stored in a secondary structure associated with the root of T. The recursion ends when the slabs contain at most $4B$ rectangle corners each. Refer to Figure 1.

The base tree T can relatively easily be constructed and the rectangles distributed to the internal nodes of T in $O(\frac{N}{B}\log_{M/B}\frac{N}{B})$ I/Os (the number of I/Os needed to sort N elements): We first sort the rectangle corners by y-coordinate. Then we construct the top $\Theta(\log_r\frac{M}{B})$ levels of the tree, that is, $O(M/B)$ nodes, in $O(N/B)$ I/Os. Finally, we distribute the remaining (sorted) rectangles to the leaves of the constructed tree and recursively construct the corresponding subtrees. Details will appear in the full version of this paper.

Now let N_v be the number of rectangles associated with an internal node v in T. Below we describe how the secondary structure associated with v uses $O(rN_v/B + r)$ blocks and can be constructed in $O(\frac{rN_v}{B}\log_{M/B}\frac{N_v}{B})$ I/Os. Since there are $O(N/(Br))$ internal nodes, and since each rectangle is stored in exactly one leaf or internal node, this means that after distributing the rectangles to nodes of T, all the secondary structures use $O(rN/B)$ blocks and they can be constructed in $O(\frac{rN}{B}\log_{M/B}\frac{N}{B})$ I/Os.

Answering a query on T. To answer a query $q = (x_q, y_q)$ on T we simply query the secondary structure of the root v of T, and then recursively query the tree rooted in the node corresponding to the horizontal slab containing q. When we reach a leaf, we simply load the at most B rectangles in the leaf and report all rectangles that contain q. Below we describe how a query on the secondary structure of v can be performed in $O(\log_B N + K_v/B)$ I/Os, where K_v is the number of reported rectangles. Thus a query is answered in $O(\log_B N \cdot \log_r\frac{N}{B} + K/B)$ I/Os overall.

Secondary structures. All that remains is to describe the secondary structure used to store the N_v rectangles associated with an internal node v of T. The

structure actually consists of $2r$ small structures, namely two structures for each of the r horizontal slabs associated with v. The first structure Ψ_t^i for slab i contains the top (horizontal) sides of all rectangles with top side in slab i, as well as all rectangles that completely span slab i; the second structure Ψ_b^i contains the bottom (horizontal) sides of all rectangles with bottom side in slab i. The structure Ψ_t^i supports *upward ray-shooting queries* from a point $q = (x_q, y_q)$, that is, it can report all segments intersected by the vertical half-line from (x_q, y_q) to $(x_q, +\infty)$. Refer to Figure 2. Similarly, Ψ_b^i supports *downward ray-shooting queries*. Thus, it is easy to see that to answer a point enclosure query $q = (x_q, y_q)$ in v, all we need to do is to query the Ψ_t^i and Ψ_b^i structures of the slab i containing point (x_q, y_q). Refer to Figure 3.

All Ψ_t and Ψ_b structures are implemented in the same basic way, using a (partially) persistent B-tree [8, 4]. A persistent B-tree uses $O(N/B)$ blocks, supports updates in $O(\log_B N)$ I/Os in the current version of the structure as normal B-trees, and answers range queries in $O(\log_B N + K/B)$ I/Os in all the previous versions of the structure. In these bounds, N is the number of updates performed on it. To build Ψ_t we simply imagine sweeping the plane with a vertical line from $-\infty$ to $+\infty$, while inserting the y-coordinate y_1 of a horizontal segment (x_1, y_1, x_2) when its left endpoint x_1 is reached, and deleting it again when its right endpoint x_2 is reached. An upward ray-shooting query $q = (x_q, y_q)$ can then be answered in $O(\log_B N + K_v/B)$ I/Os as required, simply by performing a range query $(y_q, +\infty)$ on the structure we had when the sweep-line was at x_q. Ψ_b is constructed in a symmetric manner.

The top horizontal segment of each of the N_v rectangles associated with v can be stored in r Ψ_t structures, while the bottom segments are stored in exactly one Ψ_b. Therefore, since a Ψ_t or Ψ_b structure on L segments uses $O(L/B)$ blocks and can be constructed in $O(\frac{L}{B} \log_{M/B} \frac{L}{B})$ I/Os [26], the $2r$ structures in v use $O(rN_v/B + r)$ blocks and can be constructed in $O(\frac{rN_v}{B} \log_{M/B} \frac{N_v}{B})$ I/Os overall.

Theorem 4. *A set of N axis-parallel rectangles in the plane can be stored in an external memory data structure that uses $O(rN/B)$ blocks, such that a point enclosure query can be answered in $O(\log_B N \cdot \log_r \frac{N}{B} + K/B)$ I/Os, for any $2 \le r \le B$. The structure can be constructed using $O(r\frac{N}{B} \log_{M/B} \frac{N}{B})$ I/Os.*

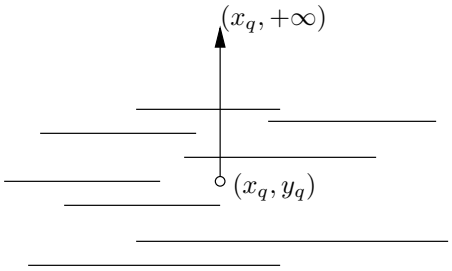

Fig. 2. An upward ray-shooting query.

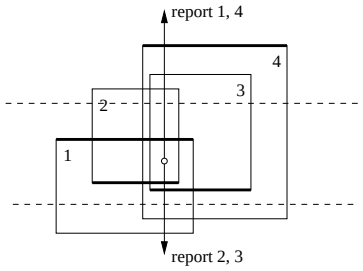

Fig. 3. Answering a point enclosure query.

3.2 Improved point enclosure structure

In this section we sketch how to improve the $O(\log_B N \cdot \log_r(N/B)+K/B)$ query bound of the structure described in the previous section to $O(\log_r(N/B)+K/B)$. First note that the extra $\log_B N$-term was a result of the query procedure using $O(\log_B N)$ I/Os to search a secondary structure (a Ψ_t and a Ψ_b structure) on each level of the base tree T. Since all of these structures are queried with the same query point q, it is natural to use fractional cascading [12] to reduce the overall search cost to $O(\log_B N + \log_r(N/B) + K/B)) = O(\log_r(N/B) + K/B))$. However, in order to do so we need to modify the Ψ_t and Ψ_b structures slightly. Below we first discuss this modification and then sketch how fractional cascading can be applied to our structure. We will only consider the Ψ_t structure since the Ψ_b structure can be handled in a similar way.

Modified secondary structure. The modification of the Ψ_t structure is inspired by the so-called *hive-graph* technique of Chazelle [10]. In order to describe it, we need the following property of a Ψ_t structure (that is, of a persistent B-tree), which follows easily from the discussions in [8].

Lemma 2. *Each of the $O(L/B)$ leaves of a persistent B-tree Ψ_t on L segments defines a rectangular region in the plane and stores the (at most) B segments intersecting this region. The regions of all the leaves define a subdivision of the minimal bounding box of the segments in Ψ_t and no two regions overlap. An upward ray-shooting query on Ψ_t visits $O(1 + K/B)$ leaves.*

By Lemma 2 an upward ray shooting query $q = (x_q, y_q)$ on a Ψ_t in node v will visit $O(1+K_v/B)$ regions (leaves) in the subdivision induced by the leaves. Now suppose we have links from any region (leaf) to its downward neighbors. Given the highest region that contains x_q we could then follow these links downward to answer the query. To find the relevant highest regions, we introduce a *head list* consisting of the left x-coordinates of the $O(L/B)$ highest regions in sorted order, where each entry also has a pointer to the corresponding region/leaf. The top region can then be found simply by searching the head list. Refer to Figure 4.

There is one problem with the above approach, namely that a region may have more than B downward neighbors (so that following the right link may require more than one I/O). We therefore modify the subdivision slightly: We imagine sweeping a horizontal line from $-\infty$ to ∞ and every time we meet the bottom edge of a region with more than B downward links, we split it into smaller regions with $\Theta(B)$ downward links each. We construct a new leaf for each of these regions, all containing the relevant segments from the original region (the segments intersecting the new region). Refer to Figure 5. Chazelle showed that the number of new regions (leaves) introduced during the sweep is $O(L/B)$ [10], so the modified subdivision still consists of $O(L/B)$ regions. It can also be constructed in $O(\frac{L}{B}\log_{M/B}\frac{L}{B})$ I/Os. Due to lack of space, we omit the details. They will appear in the full paper.

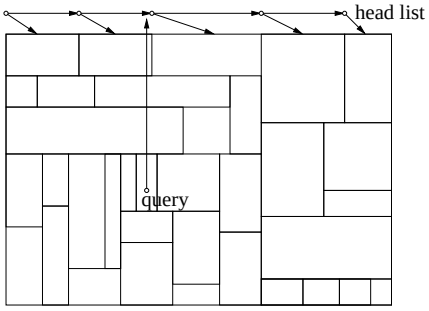

Fig. 4. The leaves of a persistent B-tree defines a rectangular subdivision.

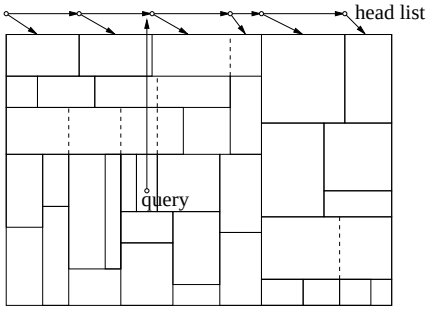

Fig. 5. Reorganizing the leaves in the persistent B-tree ($B = 2$)

Lemma 3. *A modified Ψ_t structure on L segments consists of $O(L/B)$ blocks and a head list of $O(L/B)$ x-coordinates, such that after locating x_q in the head list, an upward ray-shooting query $q = (x_q, y_q)$ can be answered in $O(1 + K/B)$ I/Os. The structure can be constructed in $O(\frac{L}{B} \log_{M/B} \frac{L}{B})$ I/Os.*

Fractional cascading. We are now left with the task of locating x_q in the head list of each Ψ_t structure that we visit along a root-to-leaf path in the base tree T. This repetitive searching problem can be solved efficiently by adapting the fractional cascading technique [12] to the external memory setting, such that after locating x_q in the first head list, finding the position of x_q in each of the subsequent head lists only incurs constant cost. Again, due to space limitations we omit the details. In the full version of this paper we show how the search cost of our structure can be reduced to $O(\log_r \frac{N}{B} + K/B)$ I/Os while maintaining the space and construction bounds; this leads to our main result.

Theorem 5. *A set of N axis-parallel rectangles in the plane can be stored in an external memory data structure that uses $O(rN/B)$ blocks such that a point enclosure query can be answered in $\Theta(\log_r \frac{N}{B} + K/B)$ I/Os, for any $2 \leq r \leq B$. The structure can be constructed with $O(r\frac{N}{B} \log_{M/B} \frac{N}{B})$ I/Os.*

References

1. P. K. Agarwal, M. de Berg, J. Gudmundsson, M. Hammer, and H. J. Haverkort. Box-trees and R-trees with near-optimal query time. In *Proc. ACM Symposium on Computational Geometry*, pages 124–133, 2001.
2. A. Aggarwal and J. S. Vitter. The Input/Output complexity of sorting and related problems. *Communications of the ACM*, 31(9):1116–1127, 1988.
3. L. Arge. External memory data structures. In J. Abello, P. M. Pardalos, and M. G. C. Resende, editors, *Handbook of Massive Data Sets*, pages 313–358. Kluwer Academic Publishers, 2002.
4. L. Arge, A. Danner, and S.-H. Teh. I/O-efficient point location using persistent B-trees. In *Proc. Workshop on Algorithm Engineering and Experimentation*, 2003.

5. L. Arge, M. de Berg, H. J. Haverkort, and K. Yi. The priority R-tree: A practically efficient and worst-case optimal R-tree. In *Proc. SIGMOD International Conference on Management of Data*, 2004.
6. L. Arge, V. Samoladas, and J. S. Vitter. On two-dimensional indexability and optimal range search indexing. In *Proc. ACM Symposium on Principles of Database Systems*, pages 346–357, 1999.
7. L. Arge and J. S. Vitter. Optimal external memory interval management. *SIAM Journal on Computing*, 32(6):1488–1508, 2003.
8. B. Becker, S. Gschwind, T. Ohler, B. Seeger, and P. Widmayer. An asymptotically optimal multiversion B-tree. *VLDB Journal*, 5(4):264–275, 1996.
9. J. L. Bentley. Multidimensional divide and conquer. *Communications of the ACM*, 23(6):214–229, 1980.
10. B. Chazelle. Filtering search: a new approach to query-answering. *SIAM J. Comput.*, 15(3):703–724, 1986.
11. B. Chazelle. Lower bounds for orthogonal range searching: I. the reporting case. *Journal of the ACM*, 37(2):200–212, Apr. 1990.
12. B. Chazelle and L. J. Guibas. Fractional cascading: I. A data structuring technique. *Algorithmica*, 1:133–162, 1986.
13. D. Comer. The ubiquitous B-tree. *ACM Computing Surveys*, 11(2):121–137, 1979.
14. M. de Berg, J. Gudmundsson, M. Hammar, and M. Overmars. On R-trees with low stabbing number. In *Proc. European Symposium on Algorithms*, pages 167–178, 2000.
15. J. R. Driscoll, N. Sarnak, D. D. Sleator, and R. Tarjan. Making data structures persistent. *Journal of Computer and System Sciences*, 38:86–124, 1989.
16. A. Fiat and A. Shamir. How to find a battleship. *Networks*, 19:361–371, 1989.
17. V. Gaede and O. Günther. Multidimensional access methods. *ACM Computing Surveys*, 30(2):170–231, 1998.
18. A. Guttman. R-trees: A dynamic index structure for spatial searching. In *Proc. SIGMOD International Conference on Management of Data*, pages 47–57, 1984.
19. J. Hellerstein, E. Koutsoupias, D. Miranker, C. Papadimitriou, and V. Samoladas. On a model of indexability and its bounds for range queries. *Journal of ACM*, 49(1), 2002.
20. J. M. Hellerstein, E. Koutsoupias, and C. H. Papadimitriou. On the analysis of indexing schemes. In *Proc. ACM Symposium on Principles of Database Systems*, pages 249–256, 1997.
21. K. V. R. Kanth and A. K. Singh. Optimal dynamic range searching in non-replicating index structures. In *Proc. International Conference on Database Theory, LNCS 1540*, pages 257–276, 1999.
22. E. Koutsoupias and D. S. Taylor. Tight bounds for 2-dimensional indexing schemes. In *Proc. ACM Symposium on Principles of Database Systems*, pages 52–58, 1998.
23. J. Matoušek. *Geometric Discrepancy*. Springer, 1999.
24. V. Samoladas and D. Miranker. A lower bound theorem for indexing schemes and its application to multidimensional range queries. In *Proc. ACM Symposium on Principles of Database Systems*, pages 44–51, 1998.
25. S. Subramanian and S. Ramaswamy. The P-range tree: A new data structure for range searching in secondary memory. In *Proc. ACM-SIAM Symposium on Discrete Algorithms*, pages 378–387, 1995.
26. J. van den Bercken, B. Seeger, and P. Widmayer. A generic approach to bulk loading multidimensional index structures. In *Proc. International Conference on Very Large Databases*, pages 406–415, 1997.