

Adaptive and Approximate Orthogonal Range Counting*

Timothy M. Chan[†]

Bryan T. Wilkinson[‡]

Abstract

We present three new results on one of the most basic problems in geometric data structures, *2-D orthogonal range counting*. All the results are in the w -bit word RAM model.

- It is well known that there are linear-space data structures for 2-D orthogonal range counting with worst-case optimal query time $O(\log_w n)$. We give an $O(n \log \log n)$ -space *adaptive* data structure that improves the query time to $O(\log \log n + \log_w k)$, where k is the output count. When $k = O(1)$, our bounds match the state of the art for the 2-D orthogonal range emptiness problem [Chan, Larsen, and Pătraşcu, SoCG 2011].
- We give an $O(n \log \log n)$ -space data structure for *approximate* 2-D orthogonal range counting that can compute a $(1 + \delta)$ -factor approximation to the count in $O(\log \log n)$ time for any fixed constant $\delta > 0$. Again, our bounds match the state of the art for the 2-D orthogonal range emptiness problem.
- Lastly, we consider the 1-D *range selection* problem, where a query in an array involves finding the k th least element in a given subarray. This problem is closely related to 2-D 3-sided orthogonal range counting. Recently, Jørgensen and Larsen [SODA 2011] presented a linear-space adaptive data structure with query time $O(\log \log n + \log_w k)$. We give a new linear-space structure that improves the query time to $O(1 + \log_w k)$, exactly matching the lower bound proved by Jørgensen and Larsen.

1 Introduction

Orthogonal range searching [4, 14, 32] is among the most extensively studied topics in algorithms and computational geometry and has numerous applications (for example, in databases and stringology). In this paper, we study one of the most basic versions of the problem: *static 2-D orthogonal range counting*. In this problem, we want to preprocess a set P of n points from \mathbb{R}^2 so that given a query rectangle $Q \subseteq \mathbb{R}^2$, we can efficiently compute $|P \cap Q|$. By contrast, in the *reporting* problem, we want to output all points in $P \cap Q$, and in the *emptiness* problem, we want to decide whether $P \cap Q = \emptyset$.

*This paper contains work published in the second author’s Master’s thesis. Work was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC).

[†]David R. Cheriton School of Computer Science, University of Waterloo, Canada, tmchan@uwaterloo.ca.

[‡]MADALGO, Aarhus University, Denmark, btw@cs.au.dk. MADALGO is a center of the Danish National Research Foundation.

The model we use in this paper is the standard w -bit word RAM [15]. In this model, the coordinates of the points of P and the vertices of Q are taken from a fixed-size universe $[U] = \{1, \dots, U\}$. Furthermore, operations on single words take constant time, and it is assumed that $w = \Omega(\log U)$ and $w = \Omega(\log n)$ so that any coordinate or index into the input array fits in a single word. Bounds on the space requirements of data structures are given in words, unless stated otherwise. We will assume that $U = O(n)$, since we can operate in *rank space* [16] (i.e., replace coordinates of points with their ranks), at the expense of performing an initial predecessor search during each query. For example, by using van Emde Boas trees for predecessor search [34, 36], we only need to add an $O(\log \log U)$ term to all the query bounds.

The classic “textbook” solution to orthogonal range searching is the *range tree* [7, 14, 32]. The space usage in 2-D is $O(n \log n)$, and the query time (assuming that we use a standard *layered* version of the range tree [37]) is $O(\log n)$ for counting or $O(\log n + k)$ for reporting, where k denotes the number of points in the query range. A long line of research [12, 29, 5, 22, 10] has been devoted to finding improved solutions. For example, for 2-D reporting, the current best upper bounds in the word RAM model were obtained recently by Chan, Larsen, and Pătraşcu [10], using a new form of compressed range tree, achieving the tradeoffs shown in the first three lines of Table 1. The current best bounds for 2-D emptiness are obtained by setting $k = 1$ in these query times. There is an $\Omega(\log \log n)$ lower bound on the query time for range emptiness, which holds for data structures using up to $n \log^{O(1)} n$ space [31].

For 2-D counting, Chazelle [12] described a compressed range tree which reduces space to $O(n)$ and keeps query time $O(\log n)$. JaJa, Mortensen, and Shi [22] used a w^ϵ -ary version of this compressed range tree to reduce the query time to $O(\log_w n)$. (Since $w = \Omega(\log n)$, the query bound is sometimes written as $O(\log n / \log \log n)$.) Pătraşcu [30] gave a matching $\Omega(\log_w n)$ lower bound on the query time for range counting, which holds for data structures using up to $n \log^{O(1)} n$ space.

Adaptive Range Counting. In this paper, we investigate the possibility of obtaining better query times

Reference	Problem	Space	Query Time
[10]	Reporting	$O(n)$	$O((1+k)\log^\epsilon n)$
[10]	Reporting	$O(n \log \log n)$	$O((1+k)\log \log n)$
[10]	Reporting	$O(n \log^\epsilon n)$	$O(\log \log n + k)$
[22]	Counting	$O(n)$	$O(\log_w n)$

Table 1: Known results for 2-D orthogonal range searching. (For the emptiness problem, set $k = 1$ in the first two lines.)

for 2-D orthogonal range counting when the output count k is small. We give an *adaptive* data structure whose query time is sensitive to k : with $O(n \log \log n)$ space, the query time is $O(\log \log n + \log_w k)$. This new query bound simultaneously encompasses the known $O(\log \log n)$ query time for emptiness when $k = 1$ (which requires $O(n \log \log n)$ space, as given in the second line of Table 1), and the known nonadaptive $O(\log_w n)$ query time for counting when $k = \Theta(n)$.

Previously, an adaptive data structure with $O(n)$ space and $O(\log \log n + \log_w k)$ query time was obtained by Jørgensen and Larsen [23] only for 2-D 3-sided orthogonal range counting (by way of a problem called range selection, defined below), but not for general 4-sided ranges.

Approximate Range Counting. Next, we consider the *approximate* 2-D orthogonal range counting problem, which involves computing an approximation k' such that $(1 - \delta)k \leq k' \leq (1 + \delta)k$ for a given fixed constant $\delta > 0$.

Much of the early research in the area of approximate range counting has focused on (nonorthogonal) halfspace range queries. Since approximate counting is at least as difficult as deciding emptiness, the ultimate goal is to get bounds matching those of emptiness. For example, for approximate 3-D halfspace range counting, Afshani et al. [2, 3] (improving earlier results [6, 24]) obtained linear-space data structures with $O(\log n)$ query time. These algorithms can be adapted to solve the related 3-D 3-sided orthogonal range counting problem (i.e., 3-D dominance counting), which includes 2-D 3-sided range counting as a special case. However, for orthogonal problems in the word RAM model, one should really aim for sublogarithmic query time. The techniques of Afshani et al. seem to inherently require suboptimal $\Omega((\log \log n)^2)$ query time, due to their use of recursion.

We give data structures for approximate 2-D orthogonal range counting with $O(n \log \log n)$ space and $O(\log \log n)$ query time, or alternatively with $O(n)$ space and $O(\log^\epsilon n)$ query time. These bounds completely match the state of the art for the 2-D range emptiness problem as stated in Table 1.

A previous work by Nekrich [27] described data structures for approximate orthogonal range counting using a more precise notion of approximation. His data structures compute an approximation k' such that $k - \delta k^p \leq k' \leq k + \delta k^p$, where $\delta > 0$ and $0 < p < 1$. His 2-D data structure requires $O(n \log^4 n)$ space and $O((1/p)\log \log n)$ query time, if p can be specified at query time. If p is fixed during preprocessing, the space requirement can be reduced to $O(n \log^2 n)$. In either case, these data structures require much more space than our new data structures.

Adaptive 1-D Range Selection. Given an array A of n elements from $[U]$, designing data structures to compute various aggregate functions over a query subarray $A[\ell : r]$ have been the subject of many papers in the literature. For example, the *range minimum query* problem (where we want to find the minimum of $A[\ell : r]$ for a given ℓ and r) has been studied since the 1980s [21]. Here, we examine a generalization that has gained much attention recently: in the *range selection query* problem, we want to find the k th least element in a given subarray, where k is specified at query time. The problem is also known as *range median* in the case when $k = \lceil (r - \ell + 1)/2 \rceil$. Range selection in an array is closely related to a special case of 2-D orthogonal range counting: deciding whether the k th smallest element in a subarray is less than a given value is equivalent to answering a range counting query for a *3-sided* 2-D orthogonal range, by treating array indices as x -coordinates and array values as y -coordinates. Improving a series of previous results [25, 19, 17, 11], Brodal and Jørgensen [8] eventually obtained a linear-space data structure that can answer range selection queries in $O(\log_w n)$ time.

At last year's SODA, Jørgensen and Larsen [23] investigated adaptive data structures for range selection that have query times sensitive to the parameter k . They proved a lower bound of $\Omega(\log_w k)$ on the query time, which holds for data structures using up to $n \log^{O(1)} n$ space. They also gave a data structure requiring $O(n)$ space and $O(\log \log n + \log_w k)$ query time. Note that for small k , there is still potential for improvement in the initial $\log \log n$ term; for example, the

classical results for range minimum queries ($k = 1$) [21] achieve *constant* query time.

In this paper, we obtain an $O(n)$ -space data structure that improves the query time to $O(1 + \log_w k)$. Thus, our bounds exactly match the the lower bound of Jørgensen and Larsen, and simultaneously extends the known result for range minimum and the known nonadaptive result for range median. (Note that for problems on 1-D arrays, the x -coordinates are already in rank space and we do not need to pay for an initial $O(\log \log U)$ predecessor search cost.)

Techniques. Our new solutions, as one might guess, use a combination of existing techniques. For example, in order to match the bounds for range emptiness, we need to adapt Chan, Larsen, and Pătraşcu’s compressed range tree and *ball inheritance* structure [10]. One of the main tools for approximate range counting, which was also used in Larsen and Jørgensen’s adaptive range selection data structure, is *shallow cuttings* [26]; we will make extensive use of such cuttings, but with additional space-saving ideas inspired by the realm of succinct data structures. Note that although the common ingredient is shallow cuttings, our range selection algorithm proceeds quite differently from Larsen and Jørgensen’s (which uses a recursion that inherently causes the extra $\log \log n$ term). Finding the right way to combine all these techniques is the main challenge, and the combination is not straightforward— as one can see from the outline in Figure 1.

2 K -Capped 3-Sided Range Counting

We begin with a problem we call *K -capped range counting*: we are given a fixed value K during preprocessing, and our goal is again to compute $|P \cap Q|$ given a query Q , but if $|P \cap Q| > K$, we are allowed to report failure. Query times are to be bounded in terms of the cap K rather than the actual count k . Our adaptive and approximate data structure will use K -capped range counting data structures as black boxes.

In this section, we first consider K -capped range counting for *3-sided* ranges, i.e., rectangles of the form $[\ell, r] \times [1, t]$. We begin with a simple solution using *shallow cuttings*.

Shallow cuttings were introduced by Matoušek [26] and were initially studied for arrangements of hyperplanes. Afshani [1] observed that shallow cuttings are applicable to 3-D 3-sided orthogonal range searching, i.e., 3-D *dominance* range searching. (Specifically in the context of 3-D dominance, a similar concept called “approximate boundaries” was also proposed by Vengroff and Vitter [35].) Assume queries are dominance regions of the form $[1, x] \times [1, y] \times [1, z]$. We use the following definition: a *K -shallow cutting* consists of $O(n/K)$ cells,

each of which is a subset of $O(K)$ points of P , with the property that for every query Q containing no more than K points, there exists a shallow cutting cell C such that $C \cap Q = P \cap Q$. It is known that a K -shallow cutting always exists with the stated size bounds. Furthermore, for every query Q , we can find its corresponding cell C by 2-D orthogonal point location.

Our interest is in 2-D 3-sided ranges, but these ranges are special cases of 3-D dominance regions, by simply mapping each point (a, b) in 2-D to $(a, b, -b)$ in 3-D. Shallow cuttings immediately imply an efficient data structure for K -capped 2-D 3-sided range counting:

LEMMA 2.1. *There exists a data structure for K -capped 2-D 3-sided orthogonal range counting requiring $O(n)$ space and $O(\log \log n + \log_w K)$ query time.*

Proof. We construct a K -shallow cutting of P . In each cell C of the shallow cutting we build the standard (nonadaptive) range counting data structure [22]. Each of the $O(n/K)$ cells thus requires $O(K)$ space, for a total of $O(n)$ space.

We can determine the cell that resolves a 3-sided query rectangle Q , if any, using a 2-D orthogonal point location query. We build the linear-space point location data structure of Chan [9] which answers queries in $O(\log \log n)$ time. If no shallow cutting cell is found, then $k > K$, so we report failure. If shallow cutting cell C is found, then we forward Q on to the standard range counting data structure stored for C . Since $|C| = O(K)$, the standard query takes $O(\log_w K)$ time. \square

Before we deal with general 4-sided queries, we need a 3-sided data structure that uses even less space than in the above lemma, i.e., a succinct data structure that uses a *sublinear* number of words of storage. For a sublinear space bound to be possible, we need to assume that the input point set is given in an implicit manner. Specifically, we assume an *oracle* is available that, given any index i , can return the point with x -rank i .

For example, Chan, Larsen, and Pătraşcu’s data structure [10] for general 4-sided emptiness relies on a succinct data structure for 3-sided emptiness queries, which is provided by the well-known *Cartesian tree* [16]. Cartesian trees are not sufficient for K -capped counting. Instead we need a new succinct representation of K -shallow cuttings, which we are able to obtain for the case of the 2-D 3-sided problem.

In the design of succinct data structures, a common technique involves finding ways to store pointers to input elements in fewer than $O(\log n)$ bits. An easy way to achieve this is to divide the input into blocks of size $\log^{O(1)} n$. Then, local pointers within a block require only $O(\log \log n)$ bits of space. We improve the

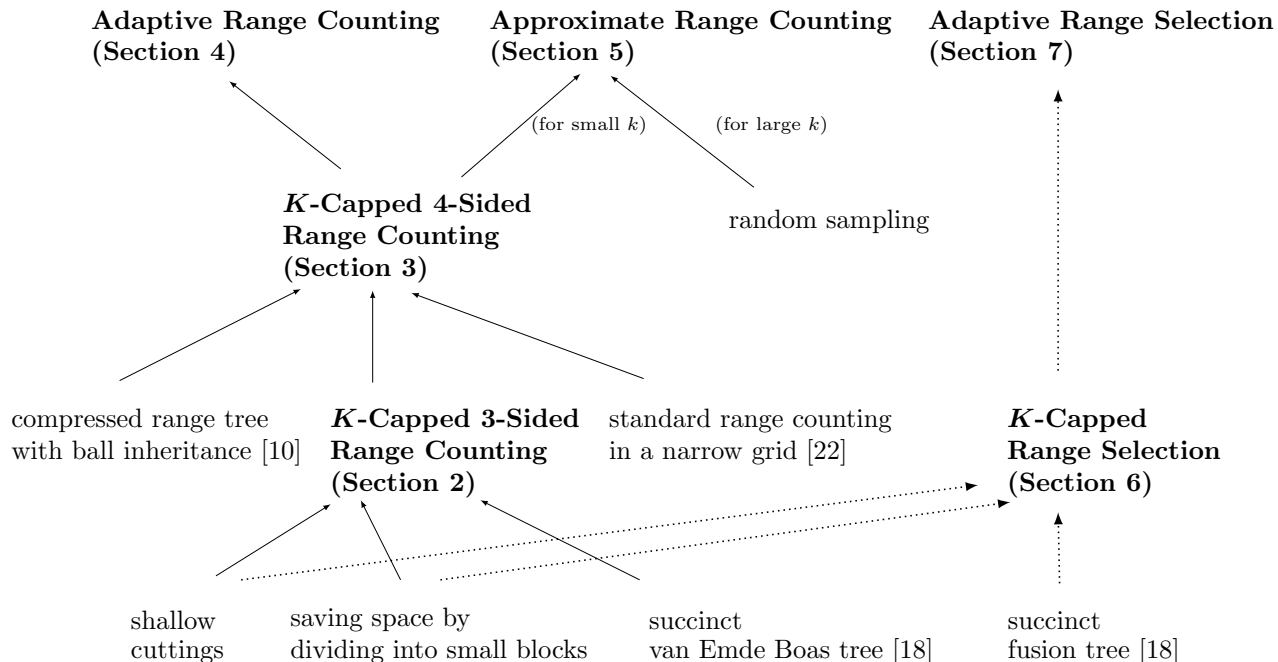


Figure 1: Outline of the techniques used in our data structures.

space bound of Lemma 2.1 with this idea in mind. We need the following succinct van Emde Boas structure for predecessor search.

LEMMA 2.2. (GROSSI ET AL. [18]) *There exists a data structure for predecessor search in a universe of size n requiring $O(n \log \log n)$ bits of space and $O(\log \log n + A(n))$ query time, where $A(n)$ is the time required for oracle access to an input element given its rank.*

THEOREM 2.1. *There exists a data structure for K -capped 2-D 3-sided orthogonal range counting requiring $O(n \log(K \log n))$ bits of space and $O(\log \log n + \log_w K + A(n))$ query time, where $A(n)$ is the time required for oracle access to a point given its x -rank in P .*

Proof. We partition P by x -coordinate into n/B vertical slabs $S_1, S_2, \dots, S_{n/B}$, each containing a block of $B = K \log n$ points. We decompose a 3-sided query of the form $[\ell, r] \times [1, t]$ into at most two 3-sided queries within slabs and at most one *slab-aligned* query: a 3-sided query whose vertical sides align with slab boundaries.

In each slab S_i we build the standard (nonadaptive) range counting data structure [22] for the points in S_i 's rank space, which requires $O(B \log B)$ bits of space. We also build the succinct predecessor search data structure of Lemma 2.2 for the points of S_i along both the x - and y -axes. These succinct data structures require only $O(B \log \log n)$ bits of space, but during a search they

require oracle access to the full $(\log n)$ -bit coordinates of $O(1)$ points, given their x - or y -ranks within S_i . For now, we simply convert oracle access given x - or y -rank within S_i to oracle access given x -rank within P . Given an x -rank j within S_i , the corresponding x -rank within P is $(i-1)B + j$, which we can easily compute in $O(1)$ time. Given a y -rank j within S_i , it is now sufficient to find the corresponding x -rank within S_i . We create an array requiring $O(B \log B)$ bits to map y -ranks to x -ranks within S_i in constant time. Across all slabs we require $O(n \log B) = O(n \log(K \log n))$ bits of space. Given a query Q within S_i , we first convert Q into a query Q' in S_i 's rank space via predecessor search in $O(\log \log n + A(n))$ time, where $A(n)$ is the time required for oracle access to a point given its x -rank in P . We then forward Q' onto the standard range counting data structure stored for S_i . Since $|S_i| = B = K \log n$, the standard query takes $O(\log_w |S_i|) = O(\log_w K)$ time.

It remains to handle slab-aligned queries. Assume we are given one such query Q . If $|Q \cap S_i| > K$, then we know that the $K+1$ points in S_i with the least y -coordinates lie in Q . The converse is also trivially true. Let S'_i be the set of $K+1$ points in S_i with the least y -coordinates. Then, for the purposes of K -capped range counting, computing $|Q \cap S'_i|$ instead of $|Q \cap S_i|$ is sufficient for a single slab S_i : if $|S_i \cap Q| \leq K$ then $|S'_i \cap Q| = |S_i \cap Q|$ and if $|S_i \cap Q| > K$ then $|S'_i \cap Q| > K$. To handle multiple slabs simultaneously,

we construct a set of points $P' = \bigcup_{i=1}^{n/B} S'_i$ and build the data structure of Lemma 2.1 over these points. Since $|P'| = (n/B)K = n/\log n$, this data structure requires only $O(n)$ bits of space. We compute $|P' \cap Q|$ in $O(\log \log n + \log_w K)$ time. If $|P' \cap Q| \leq K$ then $k = |P \cap Q| = |P' \cap Q|$. If $|P' \cap Q| > K$ then $k = |P \cap Q| > K$ and we report failure. \square

3 K -Capped 4-Sided Range Counting

We now solve the general K -capped 2-D orthogonal range counting problem. Our intention is to use a range tree in combination with Theorem 2.1 in order to support 4-sided queries.

We use the compressed range tree of Chan, Larsen, and Pătraşcu [10]. Their *ball inheritance* technique supports rank and select queries in the y dimension at every node of the range tree. Let P_v be the points stored at a node v of a range tree. Given a node v and the y -rank in P_v of a point $p \in P_v$, a *select* query returns the full $(\log n)$ -bit coordinates of p . Given a node v and a $(\log n)$ -bit y -coordinate y_0 , a *rank* query returns the y -rank in P_v of the point in P_v with the greatest y -coordinate less than or equal to y_0 . Let $S_0(n)$ and $Q_0(n)$ be the space and time required for such rank and select queries. The ball inheritance technique can achieve the following tradeoffs:

1. $S_0(n) = O(n)$ and $Q_0(n) = O(\log^\epsilon n)$, or
2. $S_0(n) = O(n \log \log n)$ and $Q_0(n) = O(\log \log n)$.

THEOREM 3.1. *There exists a data structure for K -capped 2-D orthogonal range counting requiring $O(n \log(K \log n) + S_0(n))$ space and $O(\log \log n + \log_w K + Q_0(n))$ query time.*

Proof. Assume for simplicity of presentation that n is a power of 2. Given a 4-sided query, we find the node v of the range tree that contains the two vertical sides of the query in different child nodes. Finding v can be done in constant time by simple arithmetic since the tree is perfectly balanced. We decompose the query into two 3-sided queries in child nodes of v .

At each node of the range tree, we build the data structure of Theorem 2.1 to handle 3-sided queries of the form $[1, r] \times [b, t]$ and $[\ell, \infty) \times [b, t]$. These data structures require oracle access to points given their y -ranks. The ball inheritance technique provides precisely such an oracle, with access cost $A(n) = Q_0(n)$. Excluding the ball inheritance structure, the 3-sided data structures require $O(n \log(K \log n))$ bits of space at each node of the range tree, for a total of $O(n \log(K \log n) \log n)$ bits, which are at most $O(n \log(K \log n))$ words. Given a 3-sided query Q at a child u of node v , we first convert

Q into a query Q' in u 's rank space in $Q_0(n)$ time. We then forward Q' to the 3-sided data structure stored for u . The query time is thus $O(\log \log n + \log_w K + Q_0(n))$. \square

The extra $\log(K \log n)$ factor in the space bound in Theorem 3.1 is a concern, especially for large K . In Theorem 3.2 below, we show that the factor can be completely eliminated without hurting the query time. For this, we need a known result about orthogonal range counting in a narrow $[D] \times [n]$ grid.

LEMMA 3.1. (JAJA ET AL. [22]) *There exists a data structure for 2-D orthogonal range counting on a $[D] \times [n]$ grid, for any $D \in \{1, \dots, n\}$, requiring $O(n \log D)$ bits of space and $O(\log_w D)$ query time.*

The above lemma was implicitly shown by JaJa, Mortensen, and Shi [22] when they derived their non-adaptive orthogonal range counting structure with $O(n)$ space and $O(\log_w n)$ query time. (They first solved the $[w^\epsilon] \times [n]$ grid case with $O(n \log w)$ bits of space and $O(1)$ query time, and then solved the general problem by building a w^ϵ -ary range tree. The height of the range tree would be $O(\log_w D)$ if the x values were in $[D]$. See also a paper by Nekrich [28].)

THEOREM 3.2. *There exists a data structure for K -capped 2-D orthogonal range counting requiring $O(n + S_0(n))$ space and $O(\log \log n + \log_w K + Q_0(n))$ query time.*

Proof. We follow the proof of Theorem 3.1 but use a D -ary range tree, for some parameter D which will be chosen later and is assumed to be a power of 2.

A given 4-sided query is now decomposed into at most two 3-sided queries in child nodes of v and at most one *slab-aligned* query in v : a 4-sided query whose vertical sides align with the boundaries of children of v . The two 3-sided queries are handled as before. For oracle access, we can still use the ball inheritance technique: since B is a power of 2, every node in our range tree corresponds to some node in a binary range tree. Because the height of the D -ary range tree is $O(\log_D n)$, the 3-sided structures now require a total of $O(n \log(K \log n) \log_D n)$ bits, which are at most $O(n \log(K \log n) / \log D)$ words.

It remains to handle slab-aligned queries. At each node we construct a point set P' by rounding the x -coordinates of all of the points of P to the boundaries of child nodes. Then, for any query Q whose boundaries align with those of child nodes, $|Q \cap P'| = |Q \cap P|$, so computing $|Q \cap P'|$ is sufficient. Since the points of P' lie on a narrow $[D] \times [n]$ grid, we construct the

data structure of Lemma 3.1 to handle these queries in $O(\log_w D)$ time. The space requirement at every node is $O(n \log D)$ bits, for a total of $O(n)$ words of space across the entire D -ary range tree.

To summarize, the space usage is $O(n \log(K \log n) / \log D + S_0(n))$ and the query time is $O(\log \log n + \log_w K + \log_w D + Q_0(n))$. Choosing $D = \Theta(K \log n)$ proves the theorem. \square

Substituting the known bounds on $S_0(n)$ and $Q_0(n)$, we obtain:

COROLLARY 3.1. *There exists a data structure for K -capped 2-D orthogonal range counting requiring $O(n \log \log n)$ space and $O(\log \log n + \log_w K)$ query time, or alternatively $O(n)$ space and $O(\log^\epsilon n + \log_w K)$ query time.*

4 Adaptive Range Counting

We can now get our adaptive data structure for 2-D orthogonal range counting, by building the K -capped data structures from Section 3 for different values of K .

THEOREM 4.1. *There exists a data structure for 2-D orthogonal range counting requiring $O(n \log \log n)$ space and $O(\log \log n + \log_w k)$ query time.*

Proof. We build the K -capped data structure of Theorem 3.2 for $K = 2^{2^i}$ for $i \in \{1, \dots, \lceil \log \log n \rceil\}$. Observe that the same compressed range tree and ball inheritance structure can be used across all $O(\log \log n)$ of these data structures. The total space requirement is thus $O(n \log \log n)$.

Given a query, we answer K -capped queries for $K = 2^{2^i}$ starting with $i = \lceil \log \log(w^{\log \log n}) \rceil$ and incrementing i until failure is no longer reported. If $k < w^{\log \log n}$, then the answer would be found in the first capped query in $O(\log \log n)$ time. Otherwise, the query time is upper-bounded by $O(\sum_{i=1}^{\lceil \log \log k \rceil} \log_w 2^{2^i}) = O(\log_w k)$. \square

5 Approximate Range Counting

We present an approximate data structure for 2-D orthogonal range counting, based on the following idea: if the count k is small, we can directly use the K -capped data structures from Section 3; if k is large, we can approximate k by *random sampling*, which reduces the input size and allows us to switch to a less space-efficient solution.

The idea of sampling has been used in previous approximate range counting papers [2, 20]. Consider a random sample $R \subseteq P$ where each point is chosen independently with probability r/n . For any fixed range Q with $|P \cap Q| = k$, the standard Chernoff bound

implies that for any $0 < \epsilon < 1$, $\Pr[|R \cap Q| - kr/n] > \epsilon kr/n] \leq e^{-\Omega(\epsilon^2 kr/n)}$. If $k > c(n/r) \log n$, we can set $\epsilon = \sqrt{cn/(kr) \log n}$ for a sufficiently large constant c to keep this probability smaller than $o(1/n^4)$. Since there are $O(n^4)$ combinatorially different orthogonal ranges, it follows that there exists a subset R of size $O(r)$ such that

$$(5.1) \quad \begin{aligned} |R \cap Q| - kr/n &\leq O(\sqrt{(kr/n) \log n}) \\ &\text{for all ranges } Q \text{ with} \\ |P \cap Q| = k &> c(n/r) \log n. \end{aligned}$$

THEOREM 5.1. *There exists a data structure for $(1 + 1/k^{1/2 - O(1/\log w)})$ -approximate 2-D orthogonal range counting requiring $O(n + S_0(n))$ space and $O(\log \log n + Q_0(n))$ query time.*

Proof. We build the K -capped data structure of Theorem 3.2, setting $K = w^{\log \log n}$. This data structure requires $O(n + S_0(n))$ space and computes exact counts for queries with $k \leq w^{\log \log n}$ in $O(\log \log n + \log_w K + Q_0(n)) = O(\log \log n + Q_0(n))$ time.

To handle queries such that $k > w^{\log \log n}$, we take a subset R satisfying (5.1) with size $r = n/\log^{\epsilon'} n$ for a constant ϵ' . For this subset R , we build a data structure for approximate 2-D orthogonal range counting with $O(|R| \log^{\epsilon'} |R|)$ space (which is $O(n)$) and $O(\log \log |R|)$ query time. In particular, we use Nekrich's $O(|R| \log^2 |R|)$ -space approximate data structure [27], which gives $O(\sqrt{k})$ additive error. (When we allow extra logarithmic factors in space, the problem becomes more straightforward: we can start with a 2-sided approximate counting structure and “add sides” by using the standard range tree.)

Given a query Q such that $k > w^{\log \log n}$, we first compute an approximate count k'' for $R \cap Q$ with additive error $O(\sqrt{|R \cap Q|}) = O(\sqrt{k})$. This count is found using Nekrich's data structure in $O(\log \log n)$ time. We then output $k' = (n/r)k''$. Then, k' is an approximation of $(n/r)|R \cap Q|$ with additive error $O((n/r)\sqrt{k})$. By (5.1), the difference between $(n/r)|R \cap Q|$ and k is at most $O(\sqrt{k(n/r) \log n})$. So, k' is an approximation of k with additive error $O(\sqrt{k} \log^{O(1)} n) = O(k^{1/2 + O(1/\log w)})$, since $k > w^{\log \log n}$. Thus, k' is a $(1 + 1/k^{1/2 - O(1/\log w)})$ -factor approximation of k . \square

COROLLARY 5.1. *There exists a data structure for $(1 + \delta)$ -approximate 2-D orthogonal range counting requiring $O(n \log \log n)$ space and $O(\log \log n)$ query time, or alternatively $O(n)$ space and $O(\log^\epsilon n)$ query time.*

6 K -Capped Range Selection

Given an array A of n elements from $[U]$, we consider in the next two sections the problem of preprocessing A

so that we can efficiently compute the k th least element in subarray $A[\ell : r]$, given $k, \ell, r \in \{1, \dots, n\}$ such that $\ell \leq r$ and $k \leq r - \ell + 1$. We may assume that $U = n$, via an array representation of the elements of A indexed by rank; no predecessor search cost is required to reduce to rank space.

The range selection query problem is very closely related to 2-D 3-sided range counting. Consider the 2-D point set $P = \{(i, A[i]) \mid i \in \{1, \dots, n\}\}$. Finding the k th least element in $A[\ell : r]$ is equivalent to finding the point $p_i = (i, A[i])$ with the k th lowest y -coordinate in $P \cap Q$, where $Q = [\ell, r] \times [n]$. Equivalently, p_i is the point for which $|P \cap Q'| = k$, where Q' is the 3-sided range $[\ell, r] \times [1, A[i]]$. Going forward, we work with this geometric interpretation of the range selection query problem.

We begin with the K -capped version of the range selection problem, where we are given a fixed value K during preprocessing, and all queries are known to have $k \leq K$. Query times are to be bounded in terms of K rather than k .

Our K -capped range selection data structure uses techniques similar to our K -capped 3-sided counting data structure from Section 2. However, we need new ideas to overcome two issues:

1. We use shallow cuttings but can no longer afford to spend $O(\log \log n)$ time to find the cell that resolves a query by 2-D point location, since the goal is now to get $O(1 + \log_w k)$ query time.
2. To get a succinct version of the K -capped data structure, we use the idea of dividing the input into slabs or blocks. The approach taken in the proof of Theorem 2.1 implicitly uses the fact that range counting is a *decomposable* problem—knowing the answers for two subsets, we can obtain the answer for the union of two subsets in constant time. However, range selection is not a decomposable problem.

Note that although we do not need to later extend the 3-sided structure to 4-sided and use Chan, Larsen, and Pătraşcu’s technique, we still require a succinct version of the K -capped data structure, since the final adaptive selection data structure is obtained by building multiple K -capped data structures for different values of K .

To address the first issue, we use a shallow cutting construction by Jørgensen and Larsen [23] that is specific to 2-D 3-sided ranges or 1-D range selection. In the context of the range selection query problem, a K -shallow cutting of P is again a set of $O(n/K)$ cells, each consisting of $O(K)$ points from P . If a query Q of the form $[\ell, r] \times [n]$ contains at least $k \leq K$ points, then there

must exist a cell C such that the point with the k th lowest y -coordinate in $C \cap Q$ is the point with the k th lowest y -coordinate in $P \cap Q$. We say shallow cutting cell C *resolves* query Q . The cells used in the construction are all 3-sided, i.e., subsets of the form $P \cap ([\ell, r] \times [1, t])$.

In the construction of the shallow cuttings of Jørgensen and Larsen [23], a horizontal sweep line passes from $y = 1$ to $y = n$. Throughout the sweep, we maintain a partition of the plane into vertical *slabs*, initially containing a single slab $(-\infty, \infty) \times [n]$. During the sweep, if any slab contains $2K$ points on or below the sweep line $y = y_0$, this slab is split in two at the median m of the x -coordinates of the $2K$ points. Let (m, y_0) be a *split point*. Throughout the sweep, we build a set $S = \{s_1, s_2, \dots, s_{|S|}\}$ of all split points sorted in order of x -coordinate.

Let $X = \{x_1, x_2, \dots, x_{|X|}\}$ be the x -coordinates of all split points immediately following the insertion of a new split point with x -coordinate x_i . Assume the sweep line is at $y = y_0$. After the insertion of the split point, we construct two shallow cutting cells. The first cell contains all points of P that lie in $[x_{i-2}, x_{i+1}] \times [1, y_0]$. The second cell contains all points of P that lie in $[x_{i-1}, x_{i+2}] \times [1, y_0]$. Each cell is assigned a *key*, which is a horizontal segment used to help determine in which cell a query range lies. The key for the cell defined by the range $[x_{i-2}, x_{i+1}] \times [1, y_0]$ is the segment with x -interval $[x_{i-1}, x_i]$ at height y_0 . The key for the cell defined by the range $[x_{i-1}, x_{i+2}] \times [1, y_0]$ is the segment with x -interval $[x_i, x_{i+1}]$ at height y_0 . For each pair of adjacent slabs in the final partition of the plane, we create an additional *keyless* shallow cutting cell containing all points in both slabs. Note that every cell in this shallow cutting is the intersection of some range of the form $[\ell, r] \times [1, t]$ with P .

Note that an invariant of the sweep is that all slabs contain at most $O(K)$ points on or below the sweep line. All cells (keyless or not) also contain $O(K)$ point as each cell overlaps a constant number of slabs (at the time of its creation) and only contains points on or below the sweep line. A new slab is created only when the number of points in an existing slab on or under the sweep line grows by at least K . Thus, the shallow cutting contains $O(n/K)$ cell.

Assume we are given a query $Q = [\ell, r] \times [n]$. If Q does not contain any key, then Q lies in one of the keyless shallow cutting cells. Otherwise, consider the lowest key contained in Q and let $X = \{x_1, x_2, \dots, x_{|X|}\}$ be the x -coordinates of all split points at the time of its creation. Assume without loss of generality that the key has x -interval $[x_i, x_{i+1}]$ and height y_0 . Assume that $\ell < x_{i-1}$. Then, there is a key in Q with x -interval $[x_{i-1}, x_{i+1}]$ and height less than y_0 : a contradiction.

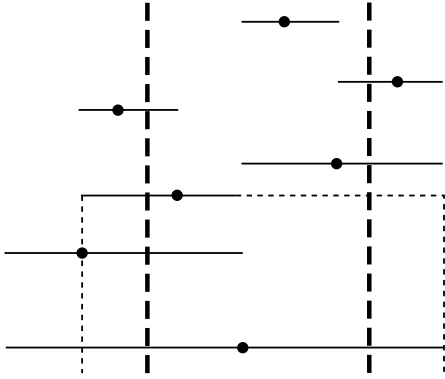


Figure 2: Split points and horizontal keys. The thin dashed rectangle is the shallow cutting cell that resolves the thick dashed query range.

Thus, $\ell \geq x_{i-1}$. Similarly, $r \leq x_{i+2}$. So, the x -interval of Q lies in the x -interval of the key's cell C . Additionally, since there are exactly K points on or below the key, C contains at least K points. For any $k \leq K$, the k th lowest point in Q must then lie in C . See Figure 2 for an example of a shallow cutting cell resolving a query range.

LEMMA 6.1. *There exists a data structure representation of a K -shallow cutting that requires $O(n)$ bits of space and that can find a shallow cutting cell that resolves a query in $O(1)$ time.*

Proof. Assume we are given a query $Q = [\ell, r] \times [n]$. In order to determine whether or not Q lies in a keyless cell, it is sufficient to count the number of split points in Q . If there are fewer than two, then Q lies in a keyless cell. We can count the number of split points in Q as well as determine the keyless cell containing Q via predecessor search in S . Predecessor search in a universe of size n can be solved by a rank operation on a binary string of length n . There exists a succinct data structure for the rank operation on a binary string that requires only $O(n)$ bits of space and constant query time [13].

If Q contains at least one key, it is sufficient to find the lowest key in Q . Consider first the lowest split point s_i in Q . There are two keys that share s_i as an endpoint. The other endpoint of each of these keys must either be directly above another split point or at negative or positive infinity along the x -axis. In the former case, since s_i is the lowest split point in Q , the key must extend to the x -coordinate of some split point that is outside of Q . In the latter case, the key is infinitely long. In either case, Q cannot contain the key.

Consider the second highest split point s_j in Q and assume without loss of generality that it lies to the left

of s_i . Then, there is a key whose left endpoint is s_j and whose right endpoint lies above s_i . This key is thus contained in Q . It is also the lowest key in Q since neither of the keys associated with s_i are in Q . Thus, we have reduced finding the lowest key in Q to finding the second lowest split point in Q .

We build a Cartesian tree over the points of S . We encode the tree succinctly (e.g., Sadakane and Navarro [33]), allowing constant-time LCA queries and requiring only $O(n/K)$ bits of space. Finding the second lowest point in $S \cap Q$ reduces to a constant number of predecessor searches in S and LCA queries in the Cartesian tree. Since we have succinct data structures for both of these problems with constant query time, we can find the second lowest point in constant time. \square

To address the second issue, in making shallow cuttings succinct, we modify the proof of Theorem 2.1 applying shallow cuttings *twice*: once to the original point set P and again to the subset P' .

LEMMA 6.2. *There exists a data structure representation of a K -shallow cutting that requires $O(n \log(K \log n) + (n/K) \log n)$ bits of space and that can access the full $(\log n)$ -bit coordinates of a point with a given x -rank in a given cell in $O(A(n))$ time, where $A(n)$ is the time required for oracle access to a point given its x -rank in P .*

Proof. We adapt the technique of Theorem 2.1. We partition P by x -coordinate into n/B vertical slabs $S_1, S_2, \dots, S_{n/B}$, each containing a block of $B = K \log n$ points. Since the size of each cell in our K -shallow cutting is $O(K)$, there exists some constant c such that each cell of our K -shallow cutting has size no greater than cK . Let S'_i be the set of cK points in S_i with the least y -coordinates. Let $P' = \bigcup_{i=1}^{n/B} S'_i$ so that $|P'| = O(n/\log n)$. We construct a cK -shallow cutting of P' . This shallow cutting has $O(n/(K \log n))$ cells, each containing $O(K)$ points.

Consider a cell C of our K -shallow cutting of P . Recall that in Jørgensen and Larsen's construction, the points of C are exactly $P \cap R$ for some 3-sided range R of the form $[\ell, r] \times [1, t]$. Let S_a be the slab that contains the left vertical side of R and let S_b be the slab that contains the right vertical side of R . Finally, consider the points $C' = C \setminus (S_a \cup S_b)$. Assuming C' is not empty, these points are exactly $P \cap R'$, where R' is a 3-sided rectangle whose vertical sides are aligned with the slab boundaries of S_a and S_b and whose top side is at the same height as R . Since $C' \subseteq C$, we have $|C'| \leq cK$. Assume towards contradiction that there is a point $p \in C'$ such that $p \notin P'$. Let S_i be the slab containing p . Since $p \notin P'$, it must also be

that $p \notin S'_i$. Since the vertical sides of R' are aligned with slab boundaries, C' must contain all points in S_i that are lower than p , including all points of S'_i . Since $|S'_i| = cK$, we have that $|C'| > cK$, a contradiction. Therefore, $C' \subseteq P'$ and $C' = P' \cap R'$. Since $|C'| \leq cK$, R' must lie in one of the cK -shallow cutting cells of P' . Let this cell be C^* . Each point $p \in C$ must either be in S_a , S_b , or C^* .

We store pointers to S_a , S_b , and C^* for C , which requires $O(\log n)$ bits. Across all cells of the K -shallow cutting of P , the space requirement is $O((n/K) \log n)$ bits. For each point $p \in C$, we store in a constant number of bits which of these three sets contains p . Let this set be X . We also store the x -rank of p within X . Since $|X| \leq \max\{|S_a|, |S_b|, |C^*|\} = O(K \log n)$, we require $O(\log(K \log n))$ bits per point. We store this information in an array T_C indexed by x -rank in C . Across all points of all cells of the K -shallow cutting of P , the space requirement is $O((n/K) \cdot K \cdot \log(K \log n)) = O(n \log(K \log n))$ bits. Given a point $p \in C$ and its x -rank i in C , we can then lookup $T_C[i]$, the x -rank of p in X , in constant time.

We store an array F_{C^*} containing the full $(\log n)$ -bit coordinates of all of the points of C^* indexed by x -rank in C^* . Across all points of all cells of the cK -shallow cutting of P' , the space requirement is $O((n/K \log n) \cdot K \cdot \log n) = O(n)$ bits. Assume we need access to the full $(\log n)$ -bit coordinates of the point $p \in C$ with x -rank i in C^* . We then lookup $F_{C^*}[i]$, the full $(\log n)$ -bit coordinates of p , in constant time.

Without loss of generality, assume we need access to a point $p \in C$ with x -rank i in S_a . We cannot afford to store the full $(\log n)$ -bit coordinates of all points of all slabs, as doing so would require $O(n \log n)$ bits of space. Instead we use an oracle that gives access to, in $A(n)$ time, the full $(\log n)$ -bit coordinates of a point given its x -rank in P . The x -rank of p within P is $(a-1)B + i$, which we can easily compute in $O(1)$ time. \square

In addition, we need a succinct predecessor search data structure for each cell, but we cannot afford the $O(\log \log n)$ cost of a van Emde Boas-type structure. Fortunately, we can use a succinct fusion tree structure (since a cell has $O(K)$ points and we can tolerate an $O(\log_w K)$ cost):

LEMMA 6.3. (GROSSI ET AL. [18]) *There exists a data structure for predecessor search requiring $O(n \log \log n)$ bits of space and $O(A(n) \log_w n)$ query time, where $A(n)$ is the time required for oracle access to an input element given its rank.*

THEOREM 6.1. *There exists a data structure for the K -capped range selection query problem that requires*

$O(n \log(K \log n) + (n/K) \log n)$ bits of space and $O(A(n)(1 + \log_w K))$ query time, where $A(n)$ is the time required for oracle access to a point given its x -rank in P .

Proof. We build a K -shallow cutting of P and represent it with the data structures of Lemmata 6.1 and 6.2. For each cell C of the shallow cutting, we build the succinct predecessor search data structure of Lemma 6.3 to search amongst the points of C along the x -axis. Each of the $O(n/K)$ cells requires $O(K \log \log n)$ bits of space for a total of $O(n \log \log n)$ bits. The succinct predecessor search data structure in each cell C requires oracle access to the full $(\log n)$ -bit coordinates of $O(\log_w K)$ points, given their x -ranks within C . We implement this oracle access via Lemma 6.2.

In each cell C , we also build the standard (non-adaptive) range selection data structure [8] in the rank space of C . This data structure requires $O(K \log K)$ bits. Across all $O(n/K)$ cells, the space requirement is $O(n \log K)$ bits.

Given a query range Q of the form $[\ell, r] \times [n]$ and a query rank $k \leq K$, we first find a cell C that resolves Q in $O(1)$ time via the data structure of Lemma 6.1. Next, we reduce Q to a query range Q' in the rank space of C in $O(A(n)(1 + \log_w K))$ time, via the succinct predecessor search data structure for C . We forward the query range Q' and the query rank k on to the standard range selection data structure for C , which requires $O(1 + \log_w K)$ time. The result is a point in C 's rank space, which we convert to a point in P in $O(A(n))$ time via Lemma 6.2. \square

7 Adaptive Range Selection

Finally we obtain an adaptive range selection data structure by using the K -capped data structure from Section 6:

THEOREM 7.1. *There exists a data structure for the range selection query problem that requires $O(n)$ space and $O(1 + \log_w k)$ query time.*

Proof. We build the K -capped data structure of Theorem 6.1 for $K = 2^{2^i}$ for $i \in \{1, \dots, \lceil \log \log n \rceil\}$. Each data structure requires the same oracle access to a point in P given its x -rank in P . We implement this oracle access simply by sorting the points of P in an array by x -rank. This implementation requires $O(n)$ space and access time $A(n) = O(1)$. The total space in bits required by all of our succinct capped data structures is $\sum_{i=1}^{\lceil \log \log n \rceil} O(n \log(2^{2^i} \log n) + (n/2^{2^i}) \log n) = O(n \log n)$.

Given a query rank k , we forward the query to the 2^{2^i} -capped data structure for $i = \lceil \log \log k \rceil$. The query

runs in $O(1 + \log_w 2^{2^i}) = O(1 + \log_w k)$ time. \square

8 Conclusion

We conclude with some open problems:

1. Our data structure for adaptive 2-D orthogonal range counting requires $O(n \log \log n)$ space, whereas JaJa et al.'s nonadaptive data structure [22] uses only $O(n)$ space. Does there exist a data structure for 2-D orthogonal range counting requiring linear space and $O(\log^\epsilon n + \log_w k)$ query time?

Note that in Corollary 3.1, we have already given a linear-space K -capped data structure for 2-D orthogonal range counting that requires $O(\log^\epsilon n + \log_w K)$ query time. However, we cannot follow the approach of building such a data structure for double exponentially increasing values of K to create an adaptive data structure, as the space requirement would increase by an $O(\log \log n)$ factor.

2. The best known data structure for the 3-D orthogonal range emptiness query problem requires $O(n \log^{1+\epsilon} n)$ space and $O(\log \log n)$ query time [10]. The best known data structure for 3-D orthogonal range counting requires $O(n \log_w n)$ space and $O((\log_w n)^2)$ query time [22]. Does there exist a data structure for 3-D orthogonal range counting requiring $O(n \log^{1+\epsilon} n)$ space and $O(\log \log n + (\log_w k)^2)$ query time? Does there exist a data structure for $(1 + \delta)$ -approximate 3-D orthogonal range counting requiring $O(n \log^{1+\epsilon} n)$ space and $O(\log \log n)$ query time?

Although K -shallow cuttings of size $O(n/K)$ exist for 3-D dominance regions, some of our techniques (e.g., dividing the input into blocks) do not seem to generalize well in 3-D.

References

- [1] P. Afshani. On dominance reporting in 3D. In *Proceedings of the 16th European Symposium on Algorithms (ESA'08)*, volume 5193 of *LNCS*, pages 41–51. Springer-Verlag, 2008.
- [2] P. Afshani and T. M. Chan. On approximate range counting and depth. *Discrete Comput. Geom.*, 42:3–21, 2009.
- [3] P. Afshani, C. Hamilton, and N. Zeh. A general approach for cache-oblivious range reporting and approximate range counting. *Comput. Geom. Theory Appl.*, 43(8):700–712, 2010.

- [4] P. K. Agarwal and J. Erickson. Geometric range searching and its relatives. In B. Chazelle, J. E. Goodman, and R. Pollack, editors, *Advances in Discrete and Computational Geometry*. AMS Press, 1999.
- [5] S. Alstrup, G. S. Brodal, and T. Rauhe. New data structures for orthogonal range searching. In *Proceedings of the 41st Symposium on Foundations of Computer Science (FOCS'00)*, pages 198–207. IEEE Computer Society, 2000.
- [6] B. Aronov and S. Har-Peled. On approximating the depth and related problems. *SIAM J. Comput.*, 38(3):899–921, 2008.
- [7] J. L. Bentley. Multidimensional divide-and-conquer. *Commun. ACM*, 23(4):214–229, 1980.
- [8] G. S. Brodal and A. G. Jørgensen. Data structures for range median queries. In *Proceedings of the 20th International Symposium on Algorithms and Computation (ISAAC'09)*, volume 5878 of *LNCS*, pages 822–831. Springer-Verlag, 2009.
- [9] T. M. Chan. Persistent predecessor search and orthogonal point location on the word RAM. In *Proceedings of the 22nd Symposium on Discrete Algorithms (SODA'11)*, pages 1131–1145. SIAM, 2011.
- [10] T. M. Chan, K. G. Larsen, and M. Pătraşcu. Orthogonal range searching on the RAM, revisited. In *Proceedings of the 27th Symposium on Computational Geometry (SoCG'11)*, pages 1–10. ACM, 2011.
- [11] T. M. Chan and M. Pătraşcu. Counting inversions, offline orthogonal range counting, and related problems. In *Proceedings of the 21st Symposium on Discrete Algorithms (SODA'10)*, pages 161–173. SIAM, 2010.
- [12] B. Chazelle. Functional approach to data structures and its use in multidimensional searching. *SIAM J. Comput.*, 17(3):427–462, 1988.
- [13] D. R. Clark and J. I. Munro. Efficient suffix trees on secondary storage. In *Proceedings of the 27th Symposium on Discrete Algorithms (SODA'96)*, pages 383–391. SIAM, 1996.
- [14] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, 3rd edition, 2008.
- [15] M. L. Fredman and D. E. Willard. Surpassing the information theoretic bound with fusion trees. *J. Comput. Syst. Sci.*, 47(3):424–436, 1993.
- [16] H. N. Gabow, J. L. Bentley, and R. E. Tarjan. Scaling and related techniques for geometry problems. In *Proceedings of the 16th Symposium on Theory of Computing (STOC'84)*, pages 135–143. ACM, 1984.
- [17] B. Gfeller and P. Sanders. Towards optimal range medians. In *Proceedings of the 36th International Colloquium on Automata, Languages and Programming (ICALP'09)*, volume 5555 of *LNCS*, pages 475–486. Springer-Verlag, 2009.
- [18] R. Grossi, A. Orlandi, R. Raman, and S. S. Rao. More haste, less waste: Lowering the redundancy in fully indexable dictionaries. In *Proceedings of the 26th Symposium on Theoretical Aspects of Computer Science (STACS'09)*, volume 3 of *Leibniz International*

- Proceedings in Informatics*, pages 517–528. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2009.
- [19] S. Har-Peled and S. Muthukrishnan. Range medians. In *Proceedings of the 16th European Symposium on Algorithms (ESA'08)*, volume 5193 of *LNCS*, pages 503–514. Springer-Verlag, 2008.
- [20] S. Har-Peled and M. Sharir. Relative (p, ε) -approximations in geometry. *Discrete Comput. Geom.*, 45(3):462–496, 2011.
- [21] D. Harel and R. E. Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM J. Comput.*, 13(2):338–355, 1984.
- [22] J. JaJa, C. W. Mortensen, and Q. Shi. Space-efficient and fast algorithms for multidimensional dominance reporting and counting. In *Proceedings of the 15th International Conference on Algorithms and Computation (ISAAC'04)*, volume 3341 of *LNCS*, pages 1755–1756. Springer-Verlag, 2005.
- [23] A. G. Jørgensen and K. G. Larsen. Range selection and median: Tight cell probe lower bounds and adaptive data structures. In *Proceedings of the 22nd Symposium on Discrete Algorithms (SODA'11)*, pages 805–813. SIAM, 2011.
- [24] H. Kaplan and M. Sharir. Randomized incremental constructions of three-dimensional convex hulls and planar Voronoi diagrams, and approximate range counting. In *Proceedings of the 17th Symposium on Discrete Algorithms (SODA'06)*, pages 484–493. ACM, 2006.
- [25] D. Krizanc, P. Morin, and M. Smid. Range mode and range median queries on lists and trees. *Nordic J. of Computing*, 12(1):1–17, 2005.
- [26] J. Matoušek. Reporting points in halfspaces. *Comput. Geom. Theory Appl.*, 2(3):169–186, 1992.
- [27] Y. Nekrich. Data structures for approximate orthogonal range counting. In *Proceedings of the 20th International Symposium on Algorithms and Computation (ISAAC'09)*, volume 5878 of *LNCS*, pages 183–192. Springer-Verlag, 2009.
- [28] Y. Nekrich. Orthogonal range searching in linear and almost-linear space. *Comput. Geom. Theory Appl.*, 42(4):342–351, 2009.
- [29] M. H. Overmars. Efficient data structures for range searching on a grid. *J. Algorithms*, 9(2):254–275, 1988.
- [30] M. Pătraşcu. Lower bounds for 2-dimensional range counting. In *Proceedings of the 39th Symposium on Theory of Computing (STOC'07)*, pages 40–46. ACM, 2007.
- [31] M. Pătraşcu and M. Thorup. Time-space trade-offs for predecessor search. In *Proceedings of the 38th Symposium on Theory of Computing (STOC'06)*, pages 232–240. ACM, 2006.
- [32] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, 1985.
- [33] K. Sadakane and G. Navarro. Fully-functional succinct trees. In *Proceedings of the 21st Symposium on Discrete Algorithms (SODA'10)*, pages 134–149. SIAM, 2010.
- [34] P. van Emde Boas, R. Kaas, and E. Zijlstra. Design and implementation of an efficient priority queue. *Theory of Comput. Syst.*, 10(1):99–127, 1976.
- [35] D. E. Vengroff and J. S. Vitter. Efficient 3-d range searching in external memory. In *Proceedings of the 28th Symposium on Theory of Computing (STOC'96)*, pages 192–201. ACM, 1996.
- [36] D. E. Willard. Log-logarithmic worst-case range queries are possible in space $\Theta(N)$. *Inf. Process. Lett.*, 17(2):81–84, 1983.
- [37] D. E. Willard. New data structures for orthogonal range queries. *SIAM J. Comput.*, 14(1):232–253, 1985.