Orthogonal Range Searching on the RAM, Revisited

Timothy M. Chan* Kasper Green Larsen† Mihai Pătraşcu‡

March 28, 2011

Abstract

We present a number of new results on one of the most extensively studied topics in computational geometry, orthogonal range searching. All our results are in the standard word RAM model:

1. We present two data structures for 2-d orthogonal range emptiness. The first achieves $O(n \lg \lg n)$ space and $O(\lg \lg n)$ query time, assuming that the n given points are in rank space. This improves the previous results by Alstrup, Brodal, and Rauhe (FOCS'00), with $O(n \lg^{\varepsilon} n)$ space and $O(\lg \lg n)$ query time, or with $O(n \lg \lg n)$ space and $O(\lg^2 \lg n)$ query time. Our second data structure uses $O(n)$ space and answers queries in $O(\lg^{\varepsilon} n)$ time. The best previous $O(n)$-space data structure, due to Nekrich (WADS'07), answers queries in $O(\lg n / \lg \lg n)$ time.
2. We give a data structure for 3-d orthogonal range reporting with $O(n \lg^{1+\varepsilon} n)$ space and $O(\lg \lg n + k)$ query time for points in rank space, for any constant $\varepsilon > 0$. This improves the previous results by Afshani (ESA'08), Karpinski and Nekrich (COCOON'09), and Chan (SODA'11), with $O(n \lg^3 n)$ space and $O(\lg \lg n + k)$ query time, or with $O(n \lg^{1+\varepsilon} n)$ space and $O(\lg^2 \lg n + k)$ query time. Consequently, we obtain improved upper bounds for orthogonal range reporting in all constant dimensions above 3.

   Our approach also leads to a new data structure for 2-d orthogonal range minimum queries with $O(n \lg^{\varepsilon} n)$ space and $O(\lg \lg n)$ query time for points in rank space.
3. We give a randomized algorithm for 4-d *offline* dominance range reporting/emptiness with running time $O(n \lg n)$ plus the output size. This resolves two open problems (both appeared in Preparata and Shamos' seminal book):
   (a) given a set of n axis-aligned rectangles in the plane, we can report all k enclosure pairs (i.e., pairs (r_1, r_2) where rectangle r_1 completely encloses rectangle r_2) in $O(n \lg n + k)$ expected time;
   (b) given a set of n points in 4-d, we can find all maximal points (points not dominated by any other points) in $O(n \lg n)$ expected time.

   The most recent previous development on (a) was reported back in SoCG'95 by Gupta, Janardan, Smid, and Dasgupta, whose main result was an $O([n \lg n + k] \lg \lg n)$ algorithm. The best previous result on (b) was an $O(n \lg n \lg \lg n)$ algorithm due to Gabow, Bentley, and Tarjan—from STOC'84! As a consequence, we also obtain the current-record time bound for the maxima problem in all constant dimensions above 4.

*This author's work was supported by an NSERC grant. School of Computer Science, University of Waterloo, tmchan@uwaterloo.ca

†MADALGO, Aarhus University, larsen@cs.au.dk. This author's work was supported in part by MADALGO—Center for Massive Data Algorithmics, a Center of the Danish National Research Foundation—and in part by a Google Europe Fellowship in Search and Information Retrieval.

‡AT&T Labs, mip@alum.mit.edu

1 Introduction

We revisit one of the most fundamental and well-studied classes of problems in computational geometry, orthogonal range searching. The goal of these problems is to preprocess a set of n input points in d-dimensional space such that one can efficiently aggregate information about the points contained in an axis-aligned query rectangle or box. The most typical types of information computed include counting the number of points, computing their semigroup or group sum, determining emptiness, and reporting the points in the query range. These problems have been studied extensively for more than three decades, yet many questions have remained unresolved. See e.g. [9, 38, 57, 1, 60, 8, 49, 10, 19, 18, 24, 20, 25, 21, 23, 46, 50, 47, 39, 6, 5, 31, 61, 54, 40] for just a fraction of the vast amount of publications on orthogonal range searching.

Recent papers [2, 3] have made progress on the pointer machine model and I/O model. In this paper, we study orthogonal range searching in the standard word RAM model, which is arguably the most natural and realistic model of computation to consider in internal memory. We obtain the best RAM upper bounds known to date for a number of problems, including: 2-d orthogonal range emptiness, 3-d orthogonal range reporting, and offline 4-d dominance range reporting.

1.1 Range Searching Data Structures

In what follows, when stating data structure results, we assume that all input point sets are in *rank space*, i.e., they have coordinates on the integer grid $[n]^d = \{0, \ldots, n-1\}^d$. This assumption is for convenience only: in a w-bit word RAM when all coordinates are in $[U]^d$ with $U = 2^w$, we can always reduce to the rank-space ($U = n$) case by adding to the query time bound a term proportional to the cost of predecessor search [55], which is e.g. $O(\lg\lg U)$ by van Emde Boas trees [59] or $O(\lg_w n)$ by fusion trees [30]. After rank space reduction, all the algorithms mentioned use only RAM operations on integers of $O(\lg n)$ bits. (The predecessor lower bound holds even for range emptiness in 2-d, so the additive predecessor cost in the upper bound is optimal.)

Range reporting in 2-d. The most basic version of orthogonal range searching is perhaps range reporting in 2-d (finding all points inside a query range). Textbook description of range trees [56] implies a solution with $O(n \lg n)$ space and $O(\lg n + k)$ query time, where k denotes the output size of the query (i.e., the number of points reported). Surprisingly, the best space–query bound for this basic problem is still open. Chazelle [19] gave an $O(n)$-space data structure with $O(\lg n + k \lg^{\varepsilon} n)$ query time, which has been reduced slightly by Nekrich [51] to $O(\lg n / \lg\lg n + k \lg^{\varepsilon} n)$. (Throughout the paper, $\varepsilon > 0$ denotes an arbitrarily small constant.) Overmars [52] gave a method with $O(n \lg n)$ space and $O(\lg\lg n + k)$ query time. This query time is optimal for $O(n \lg^{O(1)} n)$-space structures in the cell probe model (even for range emptiness in the rank-space case), by reduction from colored predecessor search [55]. Alstrup, Brodal and Rauhe [8] presented two solutions, one achieving $O(n \lg^{\varepsilon} n)$ space and optimal $O(\lg\lg n + k)$ query time, and one with $O(n \lg\lg n)$ space and $O(\lg^2 \lg n + k \lg\lg n)$ query time, both improving Chazelle's earlier data structures [19] with the corresponding space bounds. In Section 2, we present two new solutions. Our first solution achieves $O(n \lg\lg n)$ space and $O((1 + k) \lg\lg n)$ query time, thus strictly improving Alstrup et al.'s second structure. Secondly, we present an $O(n)$-space data structure with query time $O((1 + k) \lg^{\varepsilon} n)$, significantly improving the first term of Nekrich's result.

We can also solve range emptiness in 2-d (testing whether a query rectangle contains any input point) by setting $k = 0$. Here, our results are the most attractive, improving on all previous results. For example, our method with $O(n \lg\lg n)$ space has optimal $O(\lg\lg n)$ query time, and simultaneously improves both of Alstrup et al.'s solutions ($O(n \lg^{\varepsilon} n)$ space and $O(\lg\lg n)$ time, or $O(n \lg\lg n)$ space and $O(\lg^2 \lg n)$ time).

Range reporting in 3-d. By a standard reduction, Alstrup et al.'s first 2-d result directly implies a data structure for 3-d orthogonal range reporting with space $O(n \lg^{1+\varepsilon} n)$ and query time $O(\lg n + k)$; this improved an already long chain of previous work. Nekrich [50] was the first to achieve sublogarithmic query time for 3-d orthogonal range reporting: his data structure has $O(n \lg^4 n)$ space and $O(\lg^2 \lg n + k)$ query time. Afshani [1] subsequently improved the space to $O(n \lg^3 n)$ while maintaining the same $O(\lg^2 \lg n + k)$ query time. Karpinski and Nekrich [40] later reduced the space to $O(n \lg^{1+\varepsilon} n)$ at the cost of increasing the query time to $O(\lg^3 \lg n + k)$, by borrowing ideas of Alstrup et al. [8]. In these methods by Afshani [1] and Karpinski and Nekrich [40], two of the $\lg \lg n$ factors come from orthogonal planar point location. By using the most recent result on orthogonal point location by Chan [14], one of the $\lg \lg n$ factors can automatically be eliminated in all of these time bounds. This still leaves the query time of Karpinski and Nekrich's structure at $O(\lg^2 \lg n + k)$, however. In Section 3, we present a new method with $O(n \lg^{1+\varepsilon} n)$ space and optimal $O(\lg \lg n + k)$ query time, simultaneously improving all previous methods that have linear dependence in k.[1]

Range reporting in higher dimensions. By a standard reduction, the previous 3-d results [1, 40, 14] imply data structures for d-dimensional orthogonal range reporting with $O(n \lg^d n)$ space and $O(\lg^{d-3} n \lg \lg n + k)$ query time, or $O(n \lg^{d-2+\varepsilon} n)$ space and $O((\lg n / \lg \lg n)^{d-3} \lg^2 \lg n + k)$ query time for $d \geq 4$. Our result implies a d-dimensional data structure with $O(n \lg^{d-2+\varepsilon} n)$ space and $O((\lg n / \lg \lg n)^{d-3} \lg \lg n + k)$ query time. This query bound is the best known among all data structures with $O(n \lg^{O(1)} n)$ space; our space bound is the best known among all data structures with $O(\lg^{O(1)} n + k)$ query time.

The 4-d case is especially nice, as we get $O(n \lg^{2+\varepsilon} n)$ space and $O(\lg n + k)$ query time. This query time almost matches Pătraşcu's $\Omega(\lg n / \lg \lg n)$ lower bound [54] for $O(n \lg^{O(1)} n)$-space structures in the cell probe model for 4-d emptiness.

Range minimum in 2-d. Our 3-d range reporting method can also be modified to give a new result for the 2-d range minimum query problem (see the appendix), with $O(n \lg^{\varepsilon} n)$ space and $O(\lg \lg n)$ query time.

1.2 Offline Range Searching

Finally, in Section 4, we turn to *offline* (or *batched*) versions of orthogonal range searching where all queries are given in advance; the goal is to minimize the total time needed to answer all queries, including preprocessing. Offline problems are important, as efficient algorithms are often obtained through the use of efficient data structures in offline settings. Offline problems also raise new challenges, beyond simply the issue that preprocessing times sometimes get ignored in analysis of data structures in the literature. Interestingly, the complexity of offline problems may be fundamentally different from their online counterparts: examples include predecessor search (where the offline problem is related to integer sorting and can be solved in $O(\sqrt{\lg \lg n})$ expected time per query [37]), orthogonal 2-d range counting (where recently Chan and Pătraşcu [16] have obtained an offline $O(\sqrt{\lg n})$ bound per query, better than the online $O(\lg n / \lg \lg n)$ bound), and nonorthogonal 2-d point location (where Chan and Pătraşcu [17] have obtained an offline $2^{O(\sqrt{\lg \lg n})}$ bound, better than the current online $O(\lg n / \lg \lg n)$ or $O(\sqrt{\lg U / \lg \lg U})$ bound [15]).

Offline dominance reporting in 4-d and the rectangle enclosure problem in 2-d. Our main result on offline range searching is a new algorithm for the offline 4-d dominance reporting problem: given n input points and n query points, report for each query point q all input points that are dominated by q. Here, $p = (x_1, \ldots, x_d)$ is *dominated* by $q = (a_1, \ldots, a_d)$ iff $x_i \leq a_i$ for every i, i.e., p lies inside the d-sided range $(-\infty, a_1] \times \cdots \times (-\infty, a_d]$ (an *orthant*). In other words, given n red points and n blue points, we want to report all pairs (p, q) where the red point p is dominated by the blue point q. We give a randomized algorithm that solves this problem in $O(n \lg n + k)$ expected time in 4-d, where k denotes the total output size.

(Note that the best known online data structure for 4-d dominance reporting with $O(\lg n + k)$ query time requires $O(n \lg^{1+\varepsilon} n)$ space and preprocessing time at least as big, and thus is not applicable here.)

[1] As Karpinski and Nekrich [40] observed, space can be slightly reduced to $O(n \lg n \lg^{O(1)} \lg n)$ if one is willing to give up linear dependence in k, with query time $O(\lg^2 \lg n + k \lg \lg n)$.

In the literature, offline 4-d dominance reporting is studied under the guise of the 2-d *rectangle enclosure* problem: given n rectangles in 2-d, report all pairs (r_1, r_2) where rectangle r_1 completely encloses rectangle r_2. By mapping each rectangle to a point in 4-d, it is easy to see that the problem reduces to offline 4-d dominance reporting (in fact, it is equivalent to dominance reporting in the "monochromatic" case, where we equate the query point set with the input point set). This classical problem has the distinction of being the last problem covered in Preparata and Shamos' standard textbook [56]. In the early 1980s, Vaishnavi and Wood [58] and Lee and Preparata [45] both gave $O(n \lg^2 n + k)$-time algorithms. The main result of a SoCG'95 paper by Gupta et al. [36] was an $O([n \lg n + k] \lg \lg n)$-time algorithm. An alternative algorithm by Lagogiannis et al. [44] obtained the same time bound. A number of researchers (the earliest seems to be Bentley and Wood [12]) raised the question of finding an $O(n \lg n + k)$-time algorithm. Particularly frustrating is the fact that obtaining $O(n \lg n + k)$ time is easy for the similar-sounding *rectangle intersection* problem (reporting all pairs of intersecting rectangles). Our new randomized algorithm shows that the 2-d rectangle enclosure problem can be solved in $O(n \lg n + k)$ time, finally resolving a 3-decades-old question.

By a standard reduction, our result implies a randomized $O(n \lg^{d-3} n + k)$-time algorithm for offline dominance reporting for any constant dimension $d \geq 4$.

Offline dominance emptiness and the maxima problem. Our algorithm can also solve the offline dominance emptiness problem in $O(n \lg^{d-3} n)$ expected time for any constant $d \geq 4$: here, given n input points and n query points, we want to decide for each query point q whether some input point is dominated by q.

A notable application is the *maxima* problem: given n points, identify all maximal points, i.e., points that are dominated by no other point. Like its cousin, the convex hull problem, this problem plays a fundamental role in computational geometry and is often used as examples to illustrate basic algorithmic techniques. It has many applications and is related to concepts from other fields (e.g., skyline queries in databases and Pareto optimality in economics). The earliest result for dimensions $d \geq 3$ was Kung, Luccio, and Preparata's $O(n \lg^{d-2} n)$-time algorithm [43, 56] from 1975. While progress has been made on probabilistic results for random point sets [11, 27, 34], output-sensitive results [27, 41], and even instance-optimal results [4], the best worst-case result for the maxima problem has remained the one from Gabow, Bentley, and Tarjan's classic STOC'84 paper [33]. (That paper is well remembered for introducing Cartesian trees.) Gabow et al.'s time bound is $O(n \lg^{d-3} n \lg \lg n)$ for $d \geq 4$. Our (randomized) result implies the first improvement in two and a half decades: $O(n \lg^{d-3} n)$. In particular, we obtain the first $O(n \lg n)$ algorithm for the 4-d maxima problem.

Other applications. Our offline dominance result also leads to the current best results for other standard problems (see the appendix), such as bichromatic L_∞-closest pair and L_∞-minimum spanning tree for $d \geq 4$.

Organization. In the three subsequent sections, we describe our new methods for 2-d orthogonal range reporting, 3-d orthogonal range reporting, and offline 4-d dominance range reporting. These sections are independent of each other and can be read separately. Interestingly, our techniques for 2-d range reporting are not based on Alstrup et al.'s previous grid-based approach [8] but draw on new ideas related to succinct data structures. Our 3-d range reporting structure is based on Alstrup et al.'s approach, but with new twists. Finally, our 4-d offline algorithm involves an unusual (and highly nonobvious) mixture of bit-packing techniques [16] and classical computational geometric tools (Clarkson–Shor-style random sampling).

2 Range Reporting in 2-d

The goal of this section is to prove:

Theorem 2.1. *For any $2 \leq B \leq \lg^\varepsilon n$, we can solve 2-d orthogonal range reporting in rank space with: $O(n \cdot B \lg \lg n)$ space and $O(\lg \lg n + k \cdot \lg_B \lg n)$ query time; or $O(n \cdot \lg_B \lg n)$ space and $(1+k) \cdot O(B \lg \lg n)$ query time.*

At the inflection point of the two trade-offs, we get a data structure with space $O(n \lg \lg n)$ and query time $(1+k) \cdot O(\lg \lg n)$. At one extreme point, we have space $O(n)$ and query time $(1+k) \cdot O(\lg^{\varepsilon} n)$. At the other extreme, we have space $O(n \lg^{\varepsilon} n)$ and query time $O(\lg \lg n + k)$, thus matching the bounds of Alstrup et al. Note that all these tradeoffs also apply to emptiness with k set to 0. This can be seen through a black-box reduction: Assume a reporting data structure with query time $t_1 + t_2 k$ is available. Given an emptiness query, run the query algorithm on the reporting data structure using the same query. If the query algorithm terminates within t_1 computation steps, we immediately get the answer, otherwise we terminate after $t_1 + 1$ operations, at which point we know $k > 0$ and thus we know the range is nonempty.

We will describe a linear-space reduction from 2-d orthogonal range reporting to the following "ball inheritance" problem, a pointer-chasing-type problem reminiscent of fractional cascading [24]. Consider a perfect binary tree with n leaves. Also consider n labelled balls, which appear in an ordered list at the root of the tree. We imagine distributing the balls from the root down to the leaves, in $\lg n$ steps. In the i-th step, a node on level $\lg n - i$ contains a subset of the balls in an ordered list, where the order is the same as the original order at the root. Each ball chooses one of the two children of the node and is "inherited" by that child. The number of balls in each node across a level is the same. That is, on level i, each node contains exactly 2^i balls, and each leaf contains exactly one ball.

Given the inheritance data described above, the goal is to build a data structure that answers the following type of query: given a node and an index into its list of balls, what leaf does the indicated ball eventually reach? We may imagine each ball as having $\lg n$ copies, one at each node on its root-to-leaf path. Conceptually, each ball stores a pointer to its copy on the level below. The identity of a ball on level i consists of a node at level i, and the index of the ball in the node's list. The goal is, given (the identity of) a ball on some level, to traverse the pointers down to the tail of the list, and report the leaf's identity.

In Section 2.1, we give space/time trade-offs for the problem with results parallel to Theorem 2.1. These trade-offs come out naturally given the definition of ball inheritance: our data structure mimicks skip lists on n independent lists with the copies of each ball. In Section 2.2, we give a reduction from 2-dimensional range reporting to this abstract ball-inheritance problem.

2.1 Solving the Ball-Inheritance Problem

This subsection will prove:

Lemma 2.2. *For any $2 \le B \le \lg^{\varepsilon} n$, we can solve the ball-inheritance problem with: (1) space $O(nB \lg \lg n)$ and query time $O(\lg_B \lg n)$; or (2) space $O(n \lg_B \lg n)$ and query time $O(B \lg \lg n)$.*

Using standard techniques, one can represent the pointers on each level of the tree with $O(n)$ bits such that we can traverse a pointer in constant time. This uses the rank problem from succinct data structures: represent a bit vector $A[1\,..\,n]$ using $O(n)$ bits, to answer $\mathrm{rank}(k) = \sum_{i \le k} A[i]$. In fact solutions with very close to n bits of space and constant query time are known [53]. For every node, we store a solution to the rank problem among its balls, where "0" denotes a ball going to the left child, and "1" a ball going to the right child. The index of a ball in the right child is $\mathrm{rank}(i)$ evaluated at the parent. The index of ball i in the left child is $i - \mathrm{rank}(i)$.

This trivial data structure uses $O(n \lg n)$ bits in total, or $O(n)$ words, but has $O(\lg n)$ query time. For faster queries, the query will need to skip many levels at once. We use an easy generalization of rank queries, which we prove in the appendix:

Lemma 2.3. *Consider an array $A[1\,..\,n]$ with elements from some alphabet Σ. We can construct a data structure of $O(n \lg \Sigma)$ bits which answers in constant time $\mathrm{rank}(k)$ = the number of elements in $A[1\,..\,k]$ equal to $A[k]$.*

In our context, the lemma implies that we can store all pointers from balls at level i to balls at level $i+\Delta$ using $O(n\Delta)$ bits of space. Indeed, each ball can be inherited by 2^Δ descendants of its current node (the alphabet Σ will denote this choice of the descendant Δ levels below). To compute the index of a ball in the list of its node at level $i+\Delta$, we simply have to count how many balls before it at level i go to the same descendant (a rank query).

For intuition of how to use this building block, consider an abstract problem. We need to augment a linked list of m nodes in a manner similar to a skip list. Any node is allowed to store a pointer Δ nodes ahead, but this has a cost of Δ. The goal is to reach the tail of the list from anywhere in a minimal number of hops, subject to a bound on the total cost of the skip pointers. For any $2 \le B \le m$, we can solve this problem as follows:

- Traversal time $O(\lg_B m)$ with pointer cost $O(m \cdot B \lg_B m)$. Define the level of a node to be the number of trailing zeros when writing the node's position in base B. Each node on level i stores a pointer to the next node on level $i+1$, or to the tail if no such node exists. The cost of a node at level i is $O(B^{i+1})$, which is $O(\frac{m}{B^i}B^{i+1}) = O(mB)$ across a level. The traversal needs to look at $O(\lg_B m)$ pointers (one per level) before reaching the tail.
- Traversal time $O(B \lg_B m)$ with pointer cost $O(m \cdot \lg_B m)$. Each node on level i stores a pointer that skips B^i nodes, or to the tail if no such node exists. In other words, each node on level i stores a pointer to the next node on level i or higher (whichever comes first). The cost of a node on level i is $O(B^i)$, so all nodes on level i cost $O(\frac{m}{B^i}B^i) = O(m)$. The total cost is thus $O(m \lg_B m)$. We can reach the tail from anywhere with $O(B \lg_B m)$ pointer traversals, since we need at most B nodes on each level, before reaching a node on a higher level.

Returning to the ball-inheritance problem, we will implement the above strategies on the n lists of copies of each ball, using Lemma 2.3 to store pointers. We use the first strategy in the regime of fast query time, but higher space (tradeoff (1) in Lemma 2.2). Nodes on levels of the tree that are a multiple of B^i store pointers to the next level multiple of B^{i+1}. This costs $O(B^{i+1})$ bits per ball, so the total cost is $\sum_i \frac{\lg n}{B^i} \cdot O(B^{i+1}) = O(\lg n \cdot B \cdot \lg_B \lg n)$ bits per ball. This is $O(nB \lg_B \lg n)$ words of space. The query time is $O(\lg_B \lg n)$, since in each step, we jump from a level multiple of B^i to a multiple of B^{i+1}. Since the bound is insensitive to polynomial changes in B, the trade-off can be rewritten as: space $O(nB \lg\lg n)$ and query time $O(\lg_B \lg n)$.

The second strategy gives low space, but slower query, i.e. tradeoff (2) in Lemma 2.2. Nodes on levels that are a multiple of B^i store pointers to B^i levels below (or to the leaves, if no such level exists). The cost of such a level is $O(B^i)$ bits per ball, so the total cost is $\sum_i \frac{\lg n}{B^i} \cdot O(B^i) = O(\lg n \cdot \lg_B \lg n)$ bits per ball. This is space $O(n \lg_B \lg n)$ words. The query time is $O(B \lg_B \lg n)$, since we need to traverse at most B levels that are multiples of B^i before reaching a level multiple of B^{i+1}. Thus, we obtain query time $O(B \lg\lg n)$ with space $O(n \lg_B \lg n)$.

2.2 Solving Range Reporting

This subsection will show:

Lemma 2.4. *If the ball inheritance problem can be solved with space S and query time τ, 2-d range reporting can be solved with space $O(S+n)$ and query time $O\big(\lg\lg n + (1+k)\cdot\tau\big)$.*

Consider n points in 2-d rank space; we may assume n to be a power of two. We build a perfect binary tree over the x-axis. Each ball will represent a point, and the leaf where the ball ends up corresponds to its x coordinate. The order of balls at the root is the sorted order by y coordinate. We store a structure for this ball inheritance problem. The true identity of the points (their x and y coordinates) are only stored at the leaves, taking linear space. We will now describe additional data structures that allow us to answer range reporting with query time $O(\lg\lg n)$, plus $(O(1)+k)$ ball inheritance queries.

Our first ingredient is a succinct data structure for the range minimum problem (RMQ). Consider an array A of n keys (which can be compared). The query is, given an interval $[i, j]$, report the index of the minimum key among $A[i], \ldots, A[j]$. Note that a data structure for this problem does not need to remember the keys. Information theoretically, the answer is determined if we know the Cartesian tree [33] of the input; a tree takes just $2n$ bits to describe. Effective data structures matching this optimal space bound are known. See, for example, [29], which describes a data structure with $2n + o(n)$ bits of space and $O(1)$ query time.

In each node that is the right child of its parent, we build a succinct RMQ data structure on the points stored in the subtree rooted at that node. In this structure, we use the y rank of the points as indices in the array, and their x coordinates as keys. In each node that is a left child of its parent, we build a range-maximum data structure (equivalently, an RMQ data structure on the mirrored input). Since each data structure takes a number of bits linear in the size of the node, they occupy a total of $O(n \lg n)$ bits, i.e. $O(n)$ words of space.

To report points in the range $[x_1, x_2] \times [y_1, y_2]$, we proceed as follows:

1. Compute the lowest common ancestor $\mathrm{LCA}(x_1, x_2)$ in the perfect binary tree. This is a constant time operation based on the xor of x_1 and x_2: the number of zero bits at the end indicates the height of the node, and the rest of the bits indicate the nodes identity. (For instance, we can use an array of n entries to map the $x_1 \oplus x_2$ to the right node.)
2. We convert $[y_1, y_2]$ into the rank space of points inside the left and right child of $\mathrm{LCA}(x_1, x_2)$. This entails finding the successor $\hat{y}_1$ of y_1 among the y values of the points under the two nodes, and the predecessor $\hat{y}_2$ of y_2. See below for how this is done.
3. We descend to the right child of $\mathrm{LCA}(x_1, x_2)$. Using the RMQ data structure, we obtain the index m (the y rank) of the x-minimum point in the range $[\hat{y}_1, \hat{y}_2]$. We use the ball-inheritance structure to find the leaf of this point. We retrieve the x coordinate of the point from the leaf, and compare it to x_2. If greater, there is no output point in the right node. If smaller, we report this point and recurse in $[\hat{y}_1, m-1]$ and $[m+1, \hat{y}_2]$ to report more points. We finally apply the symmetric algorithm in the left child of $\mathrm{LCA}(x_1, x_2)$, using the range maxima until the points go below x_1.

The cost of step 3 is dominated by the queries to the ball-inheritance problem. The number of queries is two if the range is empty, and otherwise at most twice the number of points reported in each child of the LCA.

We now describe how to support step 2 in $O(\tau + \lg\lg n)$ time, with just $O(n)$ space in total. We will use a succinct index for predecessor search. Consider supporting predecessor queries in a sorted array $A[1\,..\,n]$ of w-bit integers. If we allow the query to access entries of the array through an oracle, it is possible to obtain a data structure of sublinear size. More precisely, one can build a data structure of $O(n \lg w)$ bits, which supports queries in $O(\lg w)$ time plus oracle access to $O(1)$ entries; see [35, Lemma 3.3]. (This idea is implicit in fusion trees [32], and dates further back to [7].)

We build such a data structure on the list of y coordinates at each node. The oracle access to the original y coordinates is precisely what the ball-inheritance problem supports. Since our points have coordinates of $O(\lg n)$ bits, each data structure uses $O(\lg\lg n)$ bits per point, so the total space is $O(n \lg\lg n)$ words. To reduce the space to linear, we store these predecessor structures only at levels that are a multiple of $\lg\lg n$. From $\mathrm{LCA}(x_1, x_2)$, we go up to the closest ancestor with a predecessor structure. We run predecessor queries for y_1 and y_2 at that node, which take $O(\lg\lg n)$ time, plus $O(1)$ queries to the ball-inheritance problem. Then, we translate these predecessors into the rank space of the left and right child of $\mathrm{LCA}(x_1, x_2)$, by walking down at most $\lg\lg n$ levels in the ball-inheritance problem (with constant time per level). This concludes the proof of Lemma 2.4, which combined with Lemma 2.2 proves Theorem 2.1.

3 Range Reporting in 3-d

In this section, we present a new data structure for 3-d orthogonal range reporting. We find it more convenient now to ignore the default assumption that points are given in rank space. The special case of dominance (i.e., 3-sided) reporting can already be solved with $O(n)$ space and $O(\lg\lg U + k)$ time by known methods [1], using the latest result on orthogonal planar point location [14]. We will show how to "add sides" without changing the asymptotic query time and without increasing space by too much.

3.1 The 3-d 4-Sided Problem

Our method is based on a simple variant of Alstrup, Brodal, and Rauhe's grid-based method [8]. Instead of a grid of dimension near $\sqrt{n} \times \sqrt{n}$ as used by Alstrup et al.'s and Karpinski and Nekrich's method [40], our key idea is to use a grid of dimension near $(n/t) \times t$ for a judiciously chosen parameter t.

Specifically, consider the problem of answering range reporting queries for 4-sided boxes in 3-d, i.e., the boxes are bounded in 4 out of the 6 sides where the unbounded sides are from different coordinate axes. W.lo.g., assume that query ranges are unbounded from below in the y and z directions. Suppose that there is a base data structure for solving the 3-d 4-sided problem with $S_0(n)$ space *in bits* and $Q_0(n, k)$ query time.

The data structure. Fix parameters t and C to be set later. Let S be a set of n points in $[U]^3$. Build a 2-d grid on the xy-plane with $n/(Ct)$ rows and t columns, so that each row contains Ct points and each column contains n/t points.[2] The number of grid cells is n/C.

We build a data structure for S as follows:

0. For each of the t columns (in left-to-right order), build a data structure recursively for the points inside the column.
1. For each of the $n/(Ct)$ rows, build a base data structure directly for the points inside the row, with $S_0(Ct)$ space and $Q_0(Ct, k)$ query time. Also build a predecessor search structure for the $n/(Ct)$ horizontal grid lines.
2. For each of the t columns, build a 3-sided reporting data structure [14] for the points inside the column, with $O(n/t)$ words of space, or $O((n/t)\lg U)$ bits of space, and $O(\lg\lg U + k)$ query time.
3. Let G be the set of at most n/C points formed by taking the z-lowest point out of each nonempty grid cell. Build a data structure for G for 4-sided queries using any known method [14] with $O((n/C)\lg^{O(1)} n)$ space and $O(\lg\lg U + k)$ query time.
4. Finally, for each nonempty grid cell, store the list of all its points sorted in increasing z-order.

Note that by unfolding the recursion, we can view our data structure as a degree-t tree T where the points in the leaves are arranged in x-order and each node stores various auxiliary data structures (items 1–4).

The space usage in bits satisfies the recurrence

$$S(n) \;=\; tS(n/t) + (n/(Ct))S_0(Ct) + O(n\lg U + (n/C)\lg^{O(1)} n).$$

For the base case, we have $S(n) = O(S_0(Ct))$ for $n < Ct$. Solving the recurrence gives

$$S(n) \;=\; O(\lg_t n \cdot [(n/(Ct))S_0(Ct) + n\lg U + (n/C)\lg^{O(1)} n])$$

(assuming that $S_0(n)/n$ is nondecreasing). The third term disappears by setting $C = \lg^c n$ for a sufficiently large constant c.

[2] Abusing notation slightly, we will not distinguish between a 2-d region (e.g., a row, column, or grid cell) and its lifting in 3-d. For simplicity, we ignore floors and ceilings.

The query algorithm. Suppose we are given a query range $q = [x_L, x_R] \times (-\infty, y_0] \times (-\infty, z_0]$ and the x-ranks of x_L and x_R w.r.t. the input point set. Let v be the lowest common ancestor of the two leaves of T whose x-range contain x_L and x_R. We can find v by performing a word operation on the two given x-ranks (no special LCA data structures are required since T is perfectly balanced).

From now on, we work exclusively at node v of the tree. There, q intersects more than one column. Say x_L and x_R are in columns j_L and j_R (which can be identified in $O(1)$ time as we know the x-ranks). Say y_0 is in row i, computable by predecessor search in $O(\lg\lg U)$ time.We can then answer the query as follows:

1. Let q_T be the ("top") portion of q inside row i. Report all points in q_T (which is 4-sided) by the base data structure at row i. The cost is $Q_0(Ct, k')$ if k' denotes the number of points in q_T.
2. Let q_L and q_R be the portions of $q - q_T$ inside columns j_L and j_R respectively. Report all points in q_L and q_R (which are 3-sided inside columns j_L and j_R respectively) by the 3-sided data structure at columns j_L and j_R. The cost is $O(\lg\lg U)$ plus the number of points in q_L and q_R.
3. Let q_I be the remaining ("interior") portion $q - (q_T \cup q_L \cup q_R)$. Find all points of G in q by querying the data structure for G. The cost is at most $O(\lg\lg U)$ plus the number of points in q_I.
4. For each point $s \in G$ found in step 3, report all points in s's grid cell with z-values below z_0 by a linear search over the cell's z-sorted list. The cost is linear in the number of points in q_I.

The overall query time is thus $Q(n, k) \;=\; Q_0(Ct, k') + O(\lg\lg U + k - k')$ for some $k' \le k$.

Bootstrapping. Assume the availability of a solution with $S_0(n) = O(n \lg U + n \lg^{1+1/\ell} n)$ and $Q_0(n, k) = O(\lg\lg U + k)$ for a constant ℓ. (For a base case with $\ell \in (0, 1]$, we can start with any known method with $O(n \lg^{O(1)} n)$ space and $O(\lg\lg U + k)$ query time [14].) Setting t with $\lg t = \lg^{\ell/(\ell+1)} n$ then yields

$$S(n) \;=\; O((\lg n)/(\lg t) \cdot [n \lg U + n \lg^{1+1/\ell} Ct]) \;=\; O(n \lg U \lg^{1/(\ell+1)} n)$$

and $Q(n, k) = O(\lg\lg U + k)$.

We need one last trick: rank space reduction. Initially, store the x-, and y-, and z-values in sorted arrays, build predecessor search structures for them, and afterwards, replace all values by their ranks. This way, we have reduced U to n, and the space (in bits) of the data structure improves to $O(n \lg U + n \lg^{1+1/(\ell+1)} n)$. For a query range q, we can initially determine the x-, y-, and z-ranks of q's endpoints in $O(\lg\lg U)$ time [59] before running the query algorithm. Incidentally, this also fulfills the assumption that the x-ranks of q's endpoints are given. After the query, we can recover the x-, y-, and z-values of each reported point by looking up the sorted arrays. The query time remains $O(\lg\lg U + k)$ (though the constant factor in the k term increases).

By bootstrapping $\lceil 1/\varepsilon \rceil$ times, we finally obtain a solution for the 3-d 4-sided problem with $O(n \lg U + n \lg^{1+\varepsilon} n)$ bits of space, i.e., $O(n \lg^{\varepsilon} n)$ words of space (by packing), and $O(\lg\lg U + k)$ query time.

3.2 The 3-d 5-Sided/6-Sided Problem

We can solve the 3-d 5-sided problem in the same way. W.l.o.g., assume that the ranges are unbounded from below in the z direction. Item 2 now stores 4-sided data structures, and space increases by a $\lg^{\varepsilon} n$ factor only as a result. The query algorithm proceeds similarly. We now have an additional bottom portion q_B, but q_T, q_B, q_L, q_R are all 4-sided, unless q lies completely inside a column (in which case we only need one query to a base data structure). Our method thus solves the 3-d 5-sided problem with $O(n \lg^{O(\varepsilon)} n)$ words of space and $O(\lg\lg U + k)$ query time.

It is known (e.g., see [50]) that the j-sided problem can be reduced to the $(j-1)$-sided problem by standard binary divide-and-conquer, where the space increases by a logarithmic factor but the query time is unchanged (if it exceeds $\lg\lg$). Thus, we can get the following result for the 3-d general (i.e., 6-sided) problem:

Theorem 3.1. *There is a data structure for 3-d orthogonal range reporting with $O(n \lg^{1+\varepsilon} n)$ space (in words) and $O(\lg\lg U + k)$ query time.*

3.3 Higher dimensions and applications.

It is known that d-dimensional orthogonal range reporting can be reduced to $(d-1)$-dimensional orthogonal range reporting by using a range tree with fan-out b, where the space increases by a $b^{O(1)} \lg_b n$ factor and the query time increases by a $\lg_b n$ factor. By setting $b = \lg^{\varepsilon} n$ (and applying rank space reduction at the beginning), Theorem 3.1 implies:

Corollary 3.2. *There is a data structure for d-dimensional orthogonal range reporting for any constant $d \geq 4$ with $O(n \lg^{d-2+\varepsilon} n)$ space and $O((\lg n/ \lg\lg n)^{d-3} \lg\lg n + k)$ query time.*

Our method also works for emptiness queries; the same bounds hold with k set to 0. Here, dominance emptiness structures are sufficient in item 2, and item 4 is unnecessary.

Range minimum queries (finding the point inside a query range with the minimum priority, assuming that each input point is given a priority value) are closely related. For example, the decision version of 2-d range minimum queries (deciding whether the minimum is at most a given value) reduces to 3-d 5-sided emptiness queries. It is no surprise then that we can obtain the same result for 2-d range minimum queries as 3-d 5-sided emptiness. Here, in item 2, we need to replace 3-d 3-sided emptiness structures with 2-d dominance range minimum structures, but 2-d dominance range minimum reduces to point location in the vertical projection of a lower envelope of 3-d orthants. This is an orthogonal 2-d point location problem, which can be solved with $O(n \lg U)$ space in bits and $O(\lg\lg U)$ query time [14]. The same analysis thus carries through.

Theorem 3.3. *There is a data structure for 2-d range minimum queries with $O(n \lg^{\varepsilon} n)$ space and $O(\lg\lg U)$ query time.*

In contrast, modifying Karpinski and Nekrich's 3-d range emptiness method [40] yields a data structure for 2-d range minimum queries with $O(n \lg^{O(1)} \lg n)$ space but $O(\lg^2 \lg n)$ query time.

Our method can also give an alternative solution to 2-d orthogonal range reporting with $O(n \lg^{\varepsilon} n)$ space and $O(\lg\lg U + k)$ time. This solution is arguably slightly simpler than Alstrup, Brodal, and Rauhe's original method [8], as their method requires constant-time 1-d range queries as a subroutine, and also requires a more intricate way of handling rank space reduction.

4 Offline Range Reporting

In this section, we present our $O(n \lg n + k)$ expected time solution for the offline 4-d dominance reporting problem on n query points and n input points, where k denotes the total output size of all n queries. In this section, we fix $w = \varepsilon \lg N$ where N denotes the maximum input size. Any w-bit word operation we introduce can be simulated in $O(1)$ time by table lookup, after preprocessing in sublinear time $2^{O(w)} = N^{O(\varepsilon)}$.

4.1 Preliminaries

We begin by describing some key subroutines and tools we need. The first subroutine is an algorithm for a special case of offline 2-d orthogonal point location. Chan and Pătraşcu [16] recently studied the offline 2-d orthogonal range counting problem and obtained a linear-time algorithm for the case when the number of points is smaller than $2^{O(\sqrt{w})}$. From this result, they then obtained an $O(n\sqrt{\lg n})$ algorithm for the general case. We apply their bit-packing technique and show that a similar result holds for orthogonal point location (see the appendix for the proof):

Lemma 4.1. *There is an algorithm for offline 2-d orthogonal point location on n query points and n disjoint axis-aligned rectangles that runs in time $O(n)$ if $n \leq 2^{O(\sqrt{w})}$ and the coordinates have been pre-sorted.*

The second subroutine is a preliminary method for the offline d-dimensional orthogonal range reporting problem. A straightforward b-ary version of the range tree, combined with a trivial method for the 1-d base case, easily gives the following bound, which with the right choice of b will turn out to be crucial in establishing our 4-d result:

Lemma 4.2. *There is an algorithm for offline d-dimensional orthogonal range reporting on n points and m boxes that runs in time $O(n \lg_b^{d-1} n + b^{d-1} m \lg_b^{d-1} n + k)$, where $b \geq 2$ is a parameter and k is the total output size, if coordinates have been pre-sorted.*

The main geometric tool we use is a randomized version of *shallow cuttings* in 3-d. Let S be a set of n points in 3-d. Pick a random sample R where each point of S is included independently with probability $1/K$ for a fixed parameter K. Define the *staircase polyhedron* $\mathcal{P}(R)$ to be the lower envelope of the orthants $O_s = [x, \infty) \times [y, \infty) \times [z, \infty)$ over all the points $s = (x, y, z) \in R$. Note that the vertices of $\mathcal{P}(R)$ include all the minimal points of R (and possibly extra points not in R); this orthogonal polyhedron $\mathcal{P}(R)$ has $O(|R|)$ number of vertices (by Euler's formula) and can be computed in time $O(|R| \lg |R|)$ by adapting a standard algorithm for 3-d minima [33, 56] (the time bound can be improved on the RAM). Let $\mathcal{VD}(R)$ denote the cells in the *vertical decomposition* of the region underneath $\mathcal{P}(R)$. The decomposition is defined as follows: take each horizontal face (a polygon) of $\mathcal{P}(R)$ and form a 2-d vertical decomposition of the face by adding y-vertical line segments at its vertices; finally, extend each resulting subface (a rectangle) downward to form a cell touching $z = -\infty$. The decomposition $\mathcal{VD}(R)$ has $O(|R|)$ size and can be computed in $O(|R|)$ additional time. For each cell $\Delta \in \mathcal{VD}(R)$, we define its *conflict list* S_Δ to consist of all points $s \in S$ with O_s intersecting Δ; equivalently, S_Δ consists of all points in S that are dominated by the top-upper-right corner v_Δ of Δ. The decomposition $\mathcal{VD}(R)$ and its conflict lists, which together we refer to as a *randomized shallow cutting* of S, satisfy some desirable properties:

Lemma 4.3. *For a random sample R of S with $\mathbb{E}[|R|] = n/K$,*

(a) $\max_{\Delta \in \mathcal{VD}(R)} |S_\Delta| = O(K \lg N)$ *with probability at least* $1 - 1/N$ *for any* $N \geq n$*;*

(b) $\mathbb{E}\left[\sum_{\Delta \in \mathcal{VD}(R)} |S_\Delta|\right] = O(n)$*;*

(c) *if a query point q dominates exactly k points of S, then q is covered by $\mathcal{VD}(R)$ (i.e., lies below $\mathcal{P}(R)$) with probability at least $1 - k/K$.*

Proof. (a) and (b) follow from the general probabilistic results by Clarkson and Shor [26, 28]; (c) is obvious by a union bound: if q is above $\mathcal{P}(R)$, then some point of S dominated by q must be chosen in R. ☐

Matoušek [48] provided a deterministic version of shallow cuttings satisfying similar, slightly stronger properties (without the extra logarithmic factor in (a) and with probability 1 in (c) for $k \leq K$). Originally, shallow cuttings were developed for halfspace range reporting and defined in terms of arrangements of planes rather than orthants; the first application to dominance range reporting was proposed by Afshani [1]. (A similar concept specific to the case of dominance called *t-approximate boundary* had also appeared [60, 50].)

For our offline problem, however, preprocessing cost matters and the above simpler randomized version is more suitable than its deterministic counterpart. Note that it is not advisable to use shallow cuttings in 4-d directly, since the number of vertices in the staircase polyhedron $\mathcal{P}(R)$ can be quadratic in $|R|$ in 4-d.

4.2 Offline 3-d Dominance Reporting

We warm up by illustrating how randomized shallow cuttings can help solve the offline dominance reporting problem in the 3-d case. The derived solution playes a key role in our 4-d solution. We assume that the given n input points and n query points have been pre-sorted.

Algorithm. We pick a random sample R of the input points, where each point is sampled with probability $1/K$ with $K := \lg n$. We first compute $\mathcal{P}(R)$ and $\mathcal{VD}(R)$.

We next compute the conflict lists for all the cells of $\mathcal{VD}(R)$ as follows. For each input point s, it suffices to identify all cells whose conflict lists include s. We first find the cell $\Delta \in \mathcal{VD}(R)$ containing s; this reduces to a 2-d point location query in the xy-projection of $\mathcal{VD}(R)$. The top-upper-right corner v_Δ of Δ gives us an initial vertex that dominates s. We observe that all vertices of the polyhedron $\mathcal{P}(R)$ that dominate the point s form a connected subgraph in the graph (the 1-skeleton) induced by the polyhedron. Furthermore, the degree of each node in the graph is at most 3. Thus we can perform a breadth-first search from the initial vertex found to generate all vertices of $\mathcal{P}(R)$ dominating v, yielding all conflict lists that include v. The total time over all input points v, excluding the initial point location queries, is linear in the total size of all conflict lists.

For each query point q, we find the cell of $\mathcal{VD}(R)$ containing q; this again reduces to a 2-d point location query in the projection of $\mathcal{VD}(R)$. If no cell is found (i.e., q is above $\mathcal{P}(R)$), then we say that q is *bad*; otherwise it is *good*. For each cell $\Delta \in \mathcal{VD}(R)$, we run an existing algorithm A_0 to solve the offline 3-d dominance reporting subproblem for the input points in the conflict list of Δ and the query points inside Δ.

This answers all good queries correctly. To finish, we recursively solve the offline 3-d dominance reporting problem on the bad queries and all the input points, where the roles of queries and input points are now reversed. Note that we also reverse the dominance relation, or equivalently, negate all coordinates. After recursing twice, however, we terminate by switching to a known $O(n \lg n + k)$-time algorithm (e.g., [45, 56]).

Analysis. Assume that the offline 2-d point location on n pre-sorted rectangles and query points takes $O(nQ_{\rm PL}(n))$ time for some non-decreasing function $Q_{\rm PL}(\cdot)$. Assume that the initial algorithm A_0 for offline 3-d dominance reporting on n pre-sorted input and query points takes $O(nQ_0(n) + k)$ (expected) time for some non-decreasing function $Q_0(\cdot)$. Note that this implies that the running time for n input points and m query points is $O((n+m)Q_0(n) + k)$, by dividing the query points into $\lceil m/n \rceil$ groups of size at most n when $m > n$.

Our algorithm spends expected time at most $O((n/K) \lg n) = O(n)$ to compute $\mathcal{P}(R)$ and $\mathcal{VD}(R)$. Performing point locations on the xy-projection of $\mathcal{VD}(R)$ (a subdivision of expected size $O(n/K)$) for both the input and query points takes time at most $O(nQ_{\rm PL}(n))$. Observe here that the input and query points have been pre-sorted, and so the rectangles can also be pre-sorted in linear time. Constructing the conflict lists by the breadth-first searches takes expected time $O(n)$ since they have expected size $O(n)$ by Lemma 4.3(b). Also by Lemma 4.3(a), every conflict list has size $O(K \lg n)$ w.h.p.; if this condition is violated, we can afford to switch to a trivial polynomial upper bound on the running time. Since the total expected size of the 3-d dominance reporting subproblems at the cells is $O(n)$, these subproblems can be solved in total expected time $O(nQ_0(O(K \lg n))) = O(nQ_0(O(\lg^2 n)))$ plus the output size.

One technicality arises: by our assumption, the coordinates of the input and query points in each subproblem should be pre-sorted first. For the x-coordinates, this can be accomplished by scanning through the global sorted x-list, and for each input or query point s in order, appending s to the end of the linked lists for the cells s participates in. The y- and z-sorted lists can be similarly dealt with. The time required is linear.

By Lemma 4.3(c), the probability that a query with output size k_i is bad is at most k_i/K. Thus, the expected number of bad queries is at most k/K for total output size k. After recursing twice, the expected number of queries and input points both decrease to $O(k/K)$. The $O(n \lg n + k)$ algorithm would then finish in expected time $O((k/K) \lg n + k) = O(k)$. We conclude that our algorithm runs in overall expected time $O(n[Q_{\rm PL}(n) + Q_0(O(\lg^2 n))] + k)$.

For example, we can use $Q_{\rm PL}(n) = O(\lg \lg n)$ by the point location method from [14] (actually in the offline setting, we can just use a plane sweep algorithm with a dynamic van Emde Boas trees), and $Q_0(n) = O(\lg n)$ by a known method for 3-d offline dominance reporting [36]. Then our algorithm would run in expected time $O(n \lg \lg n + k)$. We now show that an even better result is possible when n is small.

The case of few points. First consider the case $n \le w^{O(1)}$. By Lemma 4.1, $Q_{\text{PL}}(n) = O(1)$. We can solve 3-d dominance reporting for $O(\lg^2 n) = o(w/\lg w)$ points in linear time, since after rank space reduction, the input can be packed into $o(w)$ bits and the answer can deduced from a word operation. Thus, $Q_0(O(\lg^2 n)) = O(1)$. We therefore get an $O(n+k)$-time algorithm.

Next consider the case $n \le 2^{O(\sqrt{w})}$. By Lemma 4.1, $Q_{\text{PL}}(n) = O(1)$. Since $\lg^2 n \le w^{O(1)}$, by bootstrapping with the first case, we can set $Q_0(O(\lg^2 n)) = O(1)$. We therefore get an $O(n+k)$-time algorithm.

Theorem 4.4. *There is an algorithm for offline 3-d dominance reporting on n input points and n query points that runs in expected time $O(n \lg\lg n + k)$ if the coordinates have been pre-sorted. The time bound improves to $O(n+k)$ if in addition, $n \le 2^{O(\sqrt{w})}$.*

4.3 Offline 4-d Dominance Reporting

We are now ready to present our offline 4-d algorithm. Our algorithm follows the same basic approach employed by most data structural upper bounds for orthogonal range searching: we construct a range tree on the input points and solve an offline 3-d problem in each node of the tree. Naively, using the $O(n \lg\lg n + k)$ algorithm from Theorem 4.4 would imply only an $O(n \lg n \lg\lg n + k)$ algorithm (which nevertheless is an improvement over previous results). We need several additional ideas to achieve the final $O(n \lg n + k)$ result.

Algorithm. Construct a complete binary tree (range tree) T using the input points ordered by their last coordinate as leaves. Associate each query point to the leaf node containing its successor input point w.r.t. the last coordinate, and project all input and query points on to the first three dimensions. Each internal node u in T naturally defines an offline 3-d dominance reporting problem, using the query points in the right subtree as queries (the query points of u), and the input points in the left subtree as input (the input points of u). Clearly the combined output of all these 3-d problems constitutes the output for the 4-d problem.

To speed up the solution of these 3-d problems, our first idea is to use randomized shallow cuttings once again, but this time with a different choice of parameter K. Pick a random sample of all the n input points, where each point is included with probability $1/K$ with $K := 2^{\sqrt{w}}$. For each node u in T, let R_u denote the sample of the input points of u. We first compute $\mathcal{P}(R_u)$ and $\mathcal{VD}(R_u)$. We next compute the conflict lists for all the cells of $\mathcal{VD}(R_u)$ as in Section 4.2: namely, we find the cell of $\mathcal{VD}(R_u)$ containing each input point of u by point location, and then use breadth-first searches to generate the conflict lists in time linear in their total size. For each query point q of u, we find the cell of $\mathcal{VD}(R_u)$ containing q by point location. If for a query q, there is at least one ancestor node where q is a query point and no cell is found, we say that q is *bad*. For each cell $\Delta \in \mathcal{VD}(R_u)$, we run the algorithm from Section 4.2 to solve the offline 3-d dominance reporting subproblem for the input points of u in the conflict list of Δ and the query points of u inside Δ which are not bad.

This answers all queries that are not bad in any node. To finish, we recursively solve the offline 4-d dominance reporting problem on query points that are bad in at least one node and all the input points, where the roles of queries and input points are now reversed. After recursing twice, we terminate by switching to a known $O(n \lg^2 n + k)$-time algorithm (e.g., [45, 56, 58]).

Analysis, excluding point location. Our algorithm spends expected time at most $O((n/K)\lg n) = o(n)$ to compute $\mathcal{P}(R_u)$ and $\mathcal{VD}(R_u)$ per level of the tree. The breadth-first searches take expected time $O(n)$ per level. By Lemma 4.3(a), every conflict list has size $O(K \lg n) = 2^{O(\sqrt{w})}$ w.h.p.; if this condition is violated at any node, we can afford to switch to a trivial polynomial upper bound on the running time. Since the total expected size of the 3-d dominance reporting subproblems at the cells is $O(n)$, these subproblems can be solved in total expected time $O(n)$ per level, plus the output size, by applying Theorem 4.4 in the "few points" case. One technicality arises: the coordinates of the input and query points in each subproblem should be pre-sorted first. As in Section 4.2, this can be accomplished by scanning through the global sorted x-, y-, and z-lists in linear time. The total time excluding point location cost is then $O(n)$ per level, i.e., $O(n \lg n)$, plus the output size.

By Lemma 4.3(c), the probability that a query with output size k_i is bad at one or more nodes is at most k_i/K. Thus, the expected total number of bad queries at all nodes is at most k/K for total output size k. After recursing twice, the expected number of queries and input points both decrease to $O(k/K)$. The $O(n \lg^2 n + k)$ algorithm would then finish in expected time $O((k/K) \lg^2 n + k) = O(k)$.

Point location cost. At each node u, we need to perform point locations on the xy-projection of $\mathcal{VD}(R_u)$ for all input points and query points of u. Unfortunately, the current best offline 2-d orthogonal point location algorithm in general requires $O(\lg \lg n)$ time per query, which would result in suboptimal total time $O(n \lg n \lg \lg n)$. We suggest the following key idea: solve all the 2-d point location subproblems collectively, by transforming them into one single 3-d problem!

Specifically, consider point locations for the query points of u; locations of the input points of u can be dealt with similarly. The query points for which we must perform a point location in $\mathcal{VD}(R_u)$ are precisely those in the right subtree of u. These queries lie in a consecutive range of leaves, say ℓ_i through ℓ_j, counted from left to right. We now transform each rectangle $r = [x_1, x_2] \times [y_1, y_2]$ in the xy-projection of $\mathcal{VD}(R_u)$ into the 3-d rectangle $r' = [x_1, x_2] \times [y_1, y_2] \times [i, j]$ and collect the set B of all such 3-d boxes over all nodes in T. Similarly, we transform each query point q to another 3-d query point q'. If q has coordinates (x, y, z) and lies in leaf ℓ_i, we map q to the point $q' = (x, y, i)$. We then collect the set A of all transformed query points, and solve an offline 3-d orthogonal range reporting problem with the points in A and the boxes in B. From the output of this offline problem, we can obtain for each query point in A the set of boxes in B that contain it. This gives the answers to all the original 2-d point location queries.

Since the subdivisions $\mathcal{VD}(R_u)$ have total expected size $O(n/K)$ per level of the tree, the expected number of boxes in B is $O((n/K) \lg n)$. On the other hand, the number of points in A is n, and the total output size of the 3-d problem is $O(n \lg n)$, since each point in A lies in $O(\lg n)$ boxes in B. By applying Lemma 4.2 with $b = K^\varepsilon$, we can solve the offline 3-d orthogonal range reporting problem in expected time

$$O(n \lg_b^2 n + b^2 (n/K) \lg n \lg_b^2 n + n \lg n) \ = \ O(n (\lg n / \lg K)^2 + n \lg n) \ = \ O(n \lg n),$$

due to the fortuitous choice of $K = 2^{\sqrt{w}}$. We finally conclude

Theorem 4.5. *There is an algorithm for offline 4-d dominance reporting on n input points and n query points that runs in expected time $O(n \lg n + k)$, where k is the total output size.*

4.4 Remarks

Background on shallow cuttings. Matoušek [48] provided a deterministic version of shallow cuttings satisfying similar, slightly stronger properties as in Lemma 4.3 (without the extra logarithmic factor in (a) and with probability 1 in (c) for $k \leq K$). Originally, shallow cuttings were developed for halfspace range reporting and defined in terms of arrangements of planes rather than orthants; the first application to dominance range reporting was proposed by Afshani [1]. For our offline problem, however, preprocessing cost matters and the above simpler randomized version is more suitable than its deterministic counterpart. Note that it is not advisable to use shallow cuttings in 4-d directly, since the number of vertices in the staircase polyhedron $\mathcal{P}(R)$ can be quadratic in $|R|$ in 4-d.

Higher dimensions and applications. The d-dimensional problem reduces to the $(d-1)$-dimensional problem at the expense of a logarithmic factor increase, by standard divide-and-conquer. Theorem 4.5 thus implies:

Corollary 4.6. *There is an algorithm for offline d-dimensional dominance reporting on n input points and n query points that runs in expected time $O(n \lg^{d-3} n + k)$ for any constant $d \geq 4$, where k is the total output size.*

Our method also works for offline dominance emptiness; the same bounds hold with k set to 0. In fact, the algorithms can be slightly simplified: a query point that dominates no input points is good with probability 1 by Lemma 4.3(c), and so there is no need to recurse on the bad queries.

The problem of reporting enclosure pairs for d-dimensional boxes immediately reduces to $(2d)$-dimensional dominance reporting. The d-dimensional maxima problem obviously reduces to answering n d-dimensional offline dominance emptiness queries.

For a less obvious application, consider the computation of the L_∞*-minimum spanning tree* of n points. A reduction by Krznaric, Levcopoulos, and Nilsson [42] showed that this problem can be reduced to the *bichromatic* L_∞*-closest pair* problem: given n red points and n blue points, find a pair of red and blue points with the smallest L_∞-distance. Their reduction does not increase the asymptotic running time, if it exceeds $n \lg n$. A randomized optimization technique by Chan [13] showed that the problem can be further reduced to the following decision problem, without increasing the asymptotic expected running time: given n red points and n blue points and a value r, decide whether the L_∞-distance is at most r. By drawing hypercubes centered at the blue points of side length $2r$, this problem in turn is equivalent to deciding whether some blue hypercube contains some red point. Build a grid of side length r. We can assign points and hypercubes to grid cells via hashing in linear expected time. Inside each cell, the blue hypercubes are d-sided. So, the problem reduces to a collection of offline dominance emptiness subproblems with linear total size.

Corollary 4.7. *The maxima problem, the bichromatic* L_∞*-closest pair problem, and the* L_∞*-minimum spanning tree problem in any constant dimension* $d \geq 4$ *can be solved in* $O(n \lg^{d-3} n)$ *expected time.*

References

[1] P. Afshani. On dominance reporting in 3D. In *Proc. 16th European Symposium on Algorithms*, pages 41–51, 2008.

[2] P. Afshani, L. Arge, and K. D. Larsen. Orthogonal range reporting in three and higher dimensions. In *Proc. 50th IEEE Symposium on Foundations of Computer Science*, pages 149–158, 2009.

[3] P. Afshani, L. Arge, and K. D. Larsen. Orthogonal range reporting: Query lower bounds, optimal structures in 3d, and higher dimensional improvements. In *Proc. 26th ACM Symposium on Computational Geometry*, pages 240–246, 2010.

[4] P. Afshani, J. Barbay, and T. M. Chan. Instance-optimal geometric algorithms. In *Proc. 50th IEEE Symposium on Foundations of Computer Science*, pages 129–138, 2009.

[5] P. K. Agarwal. Range searching. In J. E. Goodman and J. O'Rourke, editors, *CRC Handbook of Discrete and Computational Geometry*. CRC Press, Inc., 2004.

[6] 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.

[7] M. Ajtai, M. L. Fredman, and J. Komlós. Hash functions for priority queues. *Information and Control*, 63(3):217–225, 1984. See also FOCS'83.

[8] S. Alstrup, G. S. Brodal, and T. Rauhe. New data structures for orthogonal range searching. In *Proc. 41st IEEE Symposium on Foundations of Computer Science*, pages 198–207, 2000.

[9] L. Arge, V. Samoladas, and J. S. Vitter. On two-dimensional indexability and optimal range search indexing. In *Proc. 18th ACM Symposium on Principles of Database Systems*, pages 346–357, New York, NY, USA, 1999. ACM.

[10] J. L. Bentley. Multidimensional divide-and-conquer. *Communications of the ACM*, 23(4):214–229, 1980.

[11] J. L. Bentley, K. L. Clarkson, and D. B. Levine. Fast linear expected-time algorithms for computing maxima and convex hulls. In *Proc. 1st ACM/SIAM Symposium on Discrete Algorithms*, pages 179–187, 1990.

[12] J. L. Bentley and D. Wood. An optimal worst case algorithm for reporting intersections of rectangles. *IEEE Transactions on Computers*, C-29:571–577, 1980.

[13] T. M. Chan. Geometric applications of a randomized optimization technique. *Discrete and Computational Geometry*, 22(4):547–567, 1999.

[14] T. M. Chan. Persistent predecessor search and orthogonal point location in the word RAM. In *Proc. 22nd ACM/SIAM Symposium on Discrete Algorithms*, 2011. To appear.

[15] T. M. Chan and M. Pătraşcu. Transdichotomous results in computational geometry, I: point location in sublogarithmic time. *SIAM Journal on Computing*, 39:703–729, 2009.

[16] T. M. Chan and M. Pătraşcu. Counting inversions, offline orthogonal range counting, and related problems. In *Proc. 21st ACM/SIAM Symposium on Discrete Algorithms*, pages 161–173, 2010.

[17] T. M. Chan and M. Pătraşcu. Transdichotomous results in computational geometry, II: offline search. *ACM Transactions on Algorithms*, submitted, 2010. Preliminary version in STOC'07.

[18] B. Chazelle. Filtering search: a new approach to query answering. *SIAM Journal on Computing*, 15(3):703–724, 1986.

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

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

[21] B. Chazelle. Lower bounds for orthogonal range searching: part II. the arithmetic model. *Journal of the ACM*, 37(3):439–463, 1990.

[22] B. Chazelle. An optimal algorithm for intersecting three-dimensional convex polyhedra. *SIAM Journal on Computing*, 21(4):671–696, 1992.

[23] B. Chazelle. Lower bounds for off-line range searching. In *Proc. 27th ACM Symposium on Theory of Computing*, pages 733–740, 1995.

[24] B. Chazelle and L. J. Guibas. Fractional cascading: I. A data structuring technique. *Algorithmica*, 1:133–162, 1986.

[25] B. Chazelle and L. J. Guibas. Fractional cascading: II. Applications. *Algorithmica*, 1:163–191, 1986.

[26] K. L. Clarkson. New applications of random sampling in computational geometry. *Discrete and Computational Geometry*, 2:195–222, 1987.

[27] K. L. Clarkson. More output-sensitive geometric algorithms. In *Proc. 35th IEEE Symposium on Foundations of Computer Science*, pages 695–702, 1994.

[28] K. L. Clarkson and P. W. Shor. Applications of random sampling in computational geometry, II. *Discrete and Computational Geometry*, 4:387–421, 1989.

[29] J. Fischer. Optimal succinctness for range minimum queries. In *Proc. 9th Latin American Theoretical Informatics Symposium*, pages 158–169, 2010.

[30] F. W. Fredman and D. E. Willard. Surpassing the information theoretic bound with fusion trees. *Journal of Computer and System Sciences*, 48(3):424–436, 1993.

[31] M. L. Fredman. A lower bound on the complexity of orthogonal range queries. *Journal of the ACM*, 28(4):696–705, 1981.

[32] M. L. Fredman and D. E. Willard. Surpassing the information theoretic bound with fusion trees. *Journal of Computer and System Sciences*, 47(3):424–436, 1993. See also STOC'90.

[33] H. N. Gabow, J. L. Bentley, and R. E. Tarjan. Scaling and related techniques for geometry problems. In *Proc. 16th ACM Symposium on Theory of Computing*, pages 135–143, 1984.

[34] M. J. Golin. A provably fast linear-expected-time maxima-finding algorithm. *Algorithmica*, 11:501–524, 1994.

[35] R. Grossi, A. Orlandi, R. Raman, and S. S. Rao. More haste, less waste: Lowering the redundancy in fully indexable dictionaries. In *Proc. 26th Symposium on Theoretical Aspects of Computer Science*, pages 517–528, 2009.

[36] P. Gupta, R. Janardan, M. Smid, and B. Dasgupta. The rectangle enclosure and point-dominance problems revisited. *International Journal of Computational Geometry & Applications*, 7:437–455, 1997.

[37] Y. Han and M. Thorup. Integer sorting in $O(n\sqrt{\lg\lg n})$ expected time and linear space. In *Proc. 43rd IEEE Symposium on Foundations of Computer Science*, pages 135–144, 2002.

[38] 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*, 49(1):35–55, 2002.

[39] J. JaJa, C. W. Mortensen, and Q. Shi. Space-efficient and fast algorithms for multidimensional dominance reporting and counting. In *Proc. 15th International Symposium on Algorithms and Computation*, pages 558–568, 2004.

[40] M. Karpinski and Y. Nekrich. Space efficient multi-dimensional range reporting. In *Proc. 15th Annual Combinatorics and Computing Conference*, pages 215–224, 2009.

[41] D. G. Kirkpatrick and R. Seidel. Output-size sensitive algorithms for finding maximal vectors. In *Proc. 1st ACM Symposium on Computational Geometry*, pages 89–96, 1985.

[42] D. Krznaric, C. Levcopoulos, and B. J. Nilsson. Minimum spanning trees in d dimensions. *Nordic Journal of Computing*, 6(4):446–461, 1999.

[43] H. T. Kung, F. Luccio, and F. P. Preparata. On finding the maxima of a set of vectors. *Journal of the ACM*, 22:469–476, 1975.

[44] G. Lagogiannis, C. Makris, and A. Tsakalidis. A new algorithm for rectangle enclosure reporting. *Information Processing Letters*, 72:177–182, 1999.

[45] D. T. Lee and F. P. Preparata. An improved algorithm for the rectangle enclosure problem. *Journal of Algorithms*, 3:218–224, 1982.

[46] G. S. Lueker. A data structure for orthogonal range queries. In *Proc. 19th IEEE Symposium on Foundations of Computer Science*, pages 28–34, 1978.

[47] C. Makris and A. Tsakalidis. Algorithms for three-dimensional dominance searching in linear space. *Information Processing Letters*, 66(6):277–283, 1998.

[48] J. Matoušek. Reporting points in halfspaces. *Computational Geometry: Theory and Applications*, 2(3):169–186, 1992.

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

[50] Y. Nekrich. A data structure for multi-dimensional range reporting. In *Proc. 23rd ACM Symposium on Computational Geometry*, pages 344–353, 2007.

[51] Y. Nekrich. Orthogonal range searching in linear and almost-linear space. *Computational Geometry: Theory and Applications*, 42:342–351, 2009.

[52] M. H. Overmars. Efficient data structures for range searching on a grid. *Journal of Algorithms*, 9:254–275, June 1988.

[53] M. Pătraşcu. Succincter. In *Proc. 49th IEEE Symposium on Foundations of Computer Science*, pages 305–313, 2008.

[54] M. Pătraşcu. Unifying the landscape of cell-probe lower bounds. In *Proc. 49th IEEE Symposium on Foundations of Computer Science*, pages 434–443, 2008.

[55] M. Pătraşcu and M. Thorup. Time-space trade-offs for predecessor search. In *Proc. 38th ACM Symposium on Theory of Computing*, pages 232–240, 2006.

[56] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, New York, NY, 1985.

[57] S. Subramanian and S. Ramaswamy. The P-range tree: a new data structure for range searching in secondary memory. In *Proc. 6th ACM/SIAM Symposium on Discrete Algorithms*, pages 378–387, 1995.

[58] V. Vaishnavi and D. Wood. Data structures for the rectangle containment and enclosure problems. *Computer Graphics and Image Processing*, 13:372–384, 1980.

[59] P. van Emde Boas. Preserving order in a forest in less than logarithmic time and linear space. *Information Processing Letters*, 6:80–82, 1977.

[60] D. E. Vengroff and J. S. Vitter. Efficient 3-D range searching in external memory. In *Proc. 28th ACM Symposium on Theory of Computing*, pages 192–201, 1996.

[61] D. E. Willard. Lower bounds for the addition-subtraction operations in orthogonal range queries and related problems. *Information and Computation*, 82(1):45–64, 1989.

A Appendix

A.1 Succinct Rank Queries (Proof of Lemma 2.3)

The proof is rather standard. We will store a "checkpoint" once every $\Sigma \lg n$ positions in the array: a record with Σ entries (of $\lg n$ bits each) that indicates how many elements of each kind we have prior to that position. Then, for each element in the array, we can simply write $A[i]$ and the number of elements equal to $A[i]$ between the last checkpoint and i. This uses $O(\lg(\Sigma \lg n))$ bits per element, so it fits our space bound if $\Sigma \geq \sqrt{\lg n}$. The query simply adds the counter stored with $A[i]$ and the appropriate counter from the last checkpoint.

If the alphabet is smaller, we employ a 2-level scheme. We store a checkpoint as above every $\Sigma \lg n$ positions. Additionally, every $\Sigma \lg \lg n$ positions, we store a minor checkpoint: a record of $\Sigma \lg(\Sigma \lg n) = O(\Sigma \lg \lg n)$ bits which indicates the number of elements of each kind from the last checkpoint to the minor checkpoint. A query retrieves the appropriate counters from the last checkpoint and the last minor checkpoint, and then must solve the rank problem between the last checkpoint and the query position. Since $\Sigma \lg \lg n \leq \sqrt{\lg n} \cdot \lg \lg n$, the array entries between minor checkpoints fit in $O(\sqrt{\lg n} \cdot \lg^2 \lg n)$ bits. Thus, we can simply store the array entries in plain form, and use a precomputed table of space $n^{o(1)}$ to answer rank queries between minor checkpoints in constant time.

A.2 Offline 2-d Orthogonal Point Location for Few Points (Proof of Lemma 4.1)

The proof follows the bit-packing approach of Chan and Pătraşcu [16]. First scan through the sorted input lists to reduce all coordinates to rank space, that is, every coordinate of a query and rectangle is an integer of value $O(n)$.

We reduce our problem to a number of 1-d disjoint-intervals stabbing problems (given a set of points and disjoint intervals, return for each point the interval containing it if the interval exists). Essentially we construct a segment tree on the rectangles (where we divide according to x-coordinates) and solve a 1-d problem in each node: Consider a trie of depth $O(\lg n)$ over the binary alphabet. For each rectangle $r = [x_1, x_2] \times [y_1, y_2]$, let ℓ_1 and ℓ_2 denote the leaves corresponding to the binary representation of x_1 and x_2. Now consider the two paths from these leaves to their lowest common ancestor. For each node u on the path to ℓ_1 where ℓ_1 lies in the left child's subtree, associate the interval $[y_1, y_2]$ to the right child. For ℓ_2, do the same, but with the roles of left and right reversed. Observe that the y-intervals associated with each node in the trie are disjoint, by the disjointness of the rectangles. Our first task is to compute for each node in the trie, a sorted list of the associated y-intervals, where the list has been packed into words to allow $O(w/\lg n)$ consecutive intervals to be stored in one word.

We construct these lists essentially by external-memory radix sorting. We start at the root node where we are given the complete input set S in sorted order of bottom y-coordinates. We scan over this list and distribute the rectangles to two sets, S_ℓ and S_r, one for the left child of the root, and one for the right. These lists are again packed into words. The set S_ℓ contains those rectangles $[x_1, x_2] \times [y_1, y_2]$ for which the left child lies on the path from the root to either of the leaves corresponding to the binary representation of x_1 or x_2. The set S_r is similar. Observe that this can be determined directly from the binary representation of x_1 and x_2. Furthermore, if $[x_1, x_2]$ completely contains the range of x-coordinates stored in the leaves associated with either the left or the right subtree, we append $[y_1, y_2]$ to a list stored for the root of that subtree. These lists are also packed into words. Finally, we recurse on the left and right subtree, using S_ℓ and S_r as input respectively.

Since the rectangles in S are given in sorted y-order and the y-intervals associated with each node are disjoint, this correctly constructs the desired lists. Furthermore, observe that we can handle all $O(w/\lg n)$ rectangles stored in one word in $O(1)$ time using table lookups. Thus we spend $O((n \lg n)/w)$ time on each of $O(\lg n)$ levels of the trie to construct the desired lists; the total time is $O((n \lg^2 n)/w)$.

We now associate each query (x, y) to the set of nodes on the path from the root to the leaf corresponding to the binary representation of x. Using the same approach as for the rectangles, we obtain a y-sorted list of the associated queries in each node of the trie in total time $O((n \lg^2 n)/w)$. To finish, we solve the 1-d disjoint-intervals stabbing problem in each node of the trie by scanning the list of associated intervals and the list of associated queries simultaneously (in order of y-coordinates). Note that we produce output only when an interval contains a query point. Using table lookups when performing the scan, we may advance at least one word in one of the lists in $O(1 + k')$ time, where k' denotes the output size between the queries and the intervals in the two considered words. Over the entire trie, the total output size is $O(n)$ and the total cost is $O((n \lg^2 n)/w + n) = O(n)$ for $n \leq 2^{O(\sqrt{w})}$.

A.3 An Alternative Algorithm for a Special Case of 4-d Offline Dominance Emptiness

In this subsection, we give an alternative *deterministic* $O(n \lg n)$-time algorithm for a special case of 4-d offline dominance emptiness: given n red points and n blue points, decide whether there exists a red point p and a blue point q such that p is dominated by q. The original 4-d offline dominance emptiness problem is stronger: there, we want to know for every blue point q whether there exists a red point dominated by q. This special case is sufficient, for example, to solve the bichromatic L_∞-closest pair and the L_∞-minimum spanning tree problem discussed in Section 4.4; however, it is not sufficient to solve the maxima problem. The alternative algorithm has the advantage that it avoids "bit tricks", though it requires a nontrivial subroutine—an algorithm of Chazelle [22] for intersecting convex polyhedra.

We first consider the problem in 3-d. It has been observed that techniques for halfspace range searching can often be adapted to dominance range searching [1]. We first point out an explicit way to reduce dominance to halfspace range searching. Surprisingly, this reduction has not appeared before to the authors' knowledge (although the idea behind the reduction, which is based on an exponentially spaced grid, is commonplace). Specifically, fix a constant $r > 3$. Assume that all the points have positive integer coordinates (we can initially sort the coordinates once at the beginning and reduce to rank space). Transform each red point $p = (i, j, k)$ to the point $p^* = (r^i, r^j, r^k)$. Transform each blue point $q = (a, b, c)$ to the halfspace $q^* = \{(x, y, z) : x/r^a + y/r^b + z/r^c \leq 3\}$. It is easy to see that p is dominated by q iff p^* lies inside q^*: if $i \leq a$, $j \leq b$, and $k \leq c$, then $r^i/r^a + r^j/r^b + r^k/r^c \leq 3$, but if $i > a$, $j > b$, or $k > c$, then $r^i/r^a + r^j/r^b + r^k/r^c \geq r > 3$.

Let P be the convex hull of the transformed red points and Q be the intersection of the complements of the transformed blue halfspaces. Then the answer to our red/blue dominance problem is no iff every transformed red point lies in the complement of every transformed blue halfspace, i.e., P lies inside Q, i.e., $P \cap Q = P$. Chazelle [22] has given a linear-time algorithm for intersecting two convex polyhedra. Thus, the 3-d problem can be solved in linear time, provided that the polyhedra P and Q are given.

Now, to solve the 4-d problem, we build a binary range tree T according to the last coordinate as before and obtain a series of 3-d subproblems of total size $O(n \lg n)$. Observe that we can pre-compute the red convex hulls P at all the nodes of T bottom-up in $O(n \lg n)$ time, by repeatedly using Chazelle's linear-time algorithm for merging two convex hulls (computing the convex hull of two convex polyhedra is dual to intersecting two convex polyhedra). Similarly, we can pre-compute the blue halfspace intersections Q at all the nodes of T in $O(n \lg n)$ time, again by repeatedly using Chazelle's algorithm for intersecting two halfspace intersections.

This immediately gives a solution to the 4-d problem with overall running time $O(n \lg n)$.

Remarks: Precision issues seem to arise since the coordinates of the transformed points and halfspaces involve large numbers, but we can simulate any primitive operation on these points and halfspaces by treating r as a symbolic variable that approaches infinity. It would be interesting to see if we can directly merge or intersect staircase polyhedra without going through the transformation and invoking Chazelle's algorithm. Note that the above approach does not work at all for the offline dominance reporting problem, or for that matter, the offline dominance emptiness problem (in the 3-d subproblem, we do not know the answer for any non-maximal blue query point whose halfspace does not appear on ∂Q).