Space-Efficient and Fast Algorithms for Multidimensional Dominance Reporting and Counting

Joseph JaJa[1], Christian W. Mortensen[2⋆], and Qingmin Shi[1]

[1] Institute of Advanced Computer Studies, University of Maryland, College Park, MD 20742, USA {joseph,qshi}@umiacs.umd.edu
[2] IT University of Copenhagen, Rued Langgaardsvej 7, 2300 København S, Denmark cworm@itu.dk.

Abstract. We present linear-space sub-logarithmic algorithms for handling the *3-dimensional dominance reporting* and the *2-dimensional dominance counting* problems. Under the RAM model as described in [M. L. Fredman and D. E. Willard. "Surpassing the information theoretic bound with fusion trees", *Journal of Computer and System Sciences*, 47:424–436, 1993], our algorithms achieve $O(\log n/\log\log n + f)$ query time for the 3-dimensional dominance reporting problem, where f is the output size, and $O(\log n/\log\log n)$ query time for the 2-dimensional dominance counting problem. We extend these results to any constant dimension $d \geq 3$, achieving $O(n(\log n/\log\log n)^{d-3})$ space and $O((\log n/\log\log n)^{d-2} + f)$ query time for the reporting case and $O(n(\log n/\log\log n)^{d-2})$ space and $O((\log n/\log\log n)^{d-1})$ query time for the counting case.

1 Introduction

The d-dimensional dominance reporting (resp. counting) problem for a set S of d-dimensional points is to store S in a data structure such that given a query point q the points in S that dominate q can be reported (resp. counted) quickly. A point $p = (p_1, p_2, \ldots, p_d)$ dominates a point $q = (q_1, q_2, \ldots, q_d)$ if $p_i \geq q_i$ for all $i = 1, \ldots, d$. A number of geometric retrieval problems involving iso-oriented objects can be reduced to these problems (see for example [EO82]). For the rest of the introduction we let n denote the number of points in S, f denote the number of points reported by a dominance reporting query, and $\epsilon > 0$ be an arbitrary small constant. The results of this paper can be summarized by the following two theorems.

Theorem 1. *For any constant $d \geq 3$ there exists a data structure for the d-dimensional dominance reporting problem using $O(n(\log n/\log\log n)^{d-3})$ space such that queries can be answered in $O((\log n/\log\log n)^{d-2} + f)$ time.*

⋆ Part of this work was done while the author was visiting the Max-Planck-Institut für Informatik, Saarbrücken, as a Marie Curie doctoral fellow.

Theorem 2. *For any constant $d \geq 2$ there exists a data structure for the d-dimensional dominance counting problem using $O(n(\log n/\log\log n)^{d-2})$ space such that queries can be answered in $O((\log n/\log\log n)^{d-1})$ time.*

Note that a d-dimensional range counting query (in which each coordinate of a point to be reported is bounded from two sides instead of one as in dominance counting) can be handled by combining the results of a constant number (depending on d) of dominance counting queries of the same dimension. Hence the results in Theorem 2 are valid for range counting queries as well.

In this paper, we assume the RAM model as described by Fredman and Willard in [FW93], which allows the construction of q-heaps [FW94] (to be discussed in Section 2). In both Theorems 1 and 2 we assume coordinates of points are integer valued. But for $d \geq 4$ in Theorem 1 and $d \geq 3$ in Theorem 2 our results actually hold for real-valued coordinates.

Our success in proving the above theorems is based on a novel generalization of the concept of dimensions. This definition, given in Section 2, allows a k-dimensional point ($k \leq d$) in the standard sense to be appended with $d-k$ "special" coordinates, each of which can take only $\lceil \log^{\epsilon} n \rceil$ different values. Our approach is based on the fact that we can prove stronger results for k-dimensional reporting (see Lemma 1) and counting queries (see Lemma 4) than what has been done before. That is, in addition to the constraints on the first k coordinates specified by a standard dominance query, our solutions satisfy the additional constraints on the $d-k$ "special" coordinates. These results will in turn lead to efficient extensions to higher dimensions.

1.1 Relation to Prior Work

In [CE87], Chazelle and Edelsbrunner proposed two linear-space algorithms for the 3-dimensional dominance reporting problem. The first achieves $O(\log n + f\log n)$ query time and the second achieves $O(\log^2 n + f)$ query time. These two algorithms were later improved by Makris and Tsakalidis [MT98] to yield $O((\log\log U)^2 \log\log\log U + f\log\log U)$ query time for a special case where coordinates of the points are integers from a bounded universe $[0,\ldots,U]$ and $O(\log n + f)$ query time for the general case. The previous best linear-space algorithm for the 2-dimensional dominance counting problem is due to Chazelle [Cha88] and achieves $O(\log n)$ query time. An external version of Chazelle's scheme was proposed by Govindarajan, Arge, and Agarwal [GAA03], which achieves linear space and $O(\log_B n)$ I/Os (B being the disk block size) for handling range counting queries in $\mathcal{R}^2$. They also extended their algorithm to handle higher dimensional range counting queries, introducing a factor of $O(\log_B n)$ to both the space and the query complexity for each additional dimension.

In [SJ03a], Shi and JaJa achieved $O(\log n/\log\log n + f)$ query time for the reporting case and $O(\log n/\log\log n)$ query time for the counting case, but at the expense of increasing the space cost in both cases by a factor of $\log^{\epsilon} n$. Like our solution, these solutions require coordinates of points to be integers.

It follows that, compared with previous solutions that require linear space and, for the reporting case, constant time per reported point, our results improve the query time by a factor of $\log\log n$. Further, the standard techniques for extending the structures into higher dimensions either require a $\log n$ factor on both the query time and the space usage for each dimension [Ben80] or a $\log^{1+\epsilon} n$ factor on the space usage and a $\log n/\log\log n$ factor on the query time for each dimension [ABR00] (while still only using constant time for each reported point). In this paper, we improve the cost of extensions to higher dimensions to a factor of $\log n/\log\log n$ per dimension for both query time and space usage.

1.2 Paper Outline

We start with preliminaries in Section 2. In Section 3 we give a solution to a slightly generalized 3-dimensional dominance reporting problem, thus proving Theorem 1 for $d = 3$. In Section 4 we give a solution to a slightly generalized 2-dimensional dominance counting problem and prove Theorem 2 for $d = 2$. Finally, we extend our results to higher dimensions in Section 5.

2 Preliminaries

If an array A is indexed by a set M we will use the notation $A[m]$ for the element of A at index $m \in M$. For integers i and j we let $[i..j]$ denote the set of integers k for which $i \leq k \leq j$. If M is a set and $k \geq 0$ is an integer we let M^k denote the set $M \times \cdots \times M$ where M is repeated k times. When stating theorems, we define $i/0 = \infty$ when $i > 0$.

For the rest of this paper, we assume n is the number of points for the dominance reporting or counting structure we are ultimately designing. While developing the ultimate structures, we will construct structures with fewer than n points. We will assume we can use time $O(n)$ to precompute a constant number of tables with size $O(n)$ which only depend on the word size and on n.

We say a point $p = (p_1, \ldots, p_d)$ has *dimension* (d', d, ϵ) if $p_{d'+1}, \ldots p_d \in [1..\lceil \log^{\epsilon} n \rceil]$. Here we assume $1 \leq d' \leq d$ and $0 < \epsilon < 1$ are all constants. We refer to p_i, $1 \leq i \leq d$, as the i-coordinate of p. We say a set S has dimension (d', d, ϵ) if all the elements of S are (d', d, ϵ)-dimensional points. When creating a data structure containing a set S with dimension (d', d, ϵ), we assume that no two different points in S share any of their first d' coordinates. That is, if $p = (p_1, \ldots, p_d) \in S$ and $r = (r_1, \ldots, r_d) \in S$ and $p \neq r$ then $p_1 \neq r_1, \ldots, p_{d'} \neq r_{d'}$. We define the *$i$-successor of an integer s* as the point $p = (p_1, \ldots, p_d) \in S$ such that $p_i \geq s$ and p_i is minimized. We define the *i-rank of an integer s* as the number of points $(p_1, \ldots, p_d) \in S$ for which $p_i \leq s$, and the *i-rank of a point* in S as the i-rank of its i-coordinate. We say that i-coordinate is in *rank space* if for all points $p = (p_1, \ldots, p_d) \in S$ the i-rank of p_i is equal to p_i.

We will use the q-heap data structure of Fredman and Willard [FW94]. A q-heap allows a set S of words where S has cardinality $O(\log^{1/4} n)$ to be stored such that elements can be inserted in and deleted from S in constant time and

such that given a query integer q the largest element of S smaller than q can be identified in constant time. The q-heap requires a precomputed table with size $O(n)$ computable in $O(n)$ time. Note that the size of this table only depends on the word size and on n. Further, the q-heap requires an msb operation as discussed in Section **??**.

3 Three-dimensional Dominance Reporting

The goal of this section is to design a data structure to solve the dominance reporting problem for a set with dimension $(3, d, \epsilon)$ for $d \geq 3$, in particular proving Theorem 1 for $d = 3$. We give this structure in Section 3.2. First, we give, in Section 3.1, a solution to a variant of the dominance reporting problem for a set with dimension $(2, d, \epsilon)$, where $d \geq 2$.

3.1 $(2, d, \epsilon)$-dimensional 3-sided Reporting

We define the *3-sided reporting* problem for a set S with dimension $(2, d, \epsilon)$ as a generalization of the dominance reporting problem as follows. A query must have the form $((q_1, q_1'), q_2, \ldots, q_d)$ where $q_1 \leq q_1'$. The answer to such a query is the set of points $p = (p_1, \ldots, p_d) \in S$ for which $q_1 \leq p_1 \leq q_1'$ and $p_2 \geq q_2, \ldots, p_q \geq q_d$. We have:

Lemma 1. *Let $d \geq 2$ and $0 < \epsilon < 1/(d-2)$ be constants and let S be a $(2, d, \epsilon)$-dimensional point set of size $m \leq n$, where the 1-coordinate is in rank space. Then there exists a solution to the 3-sided reporting problem for S using $O(m)$ space such that queries reporting f points can be answered in $O(1 + f)$ time.*

For $d = 2$ this problem is well studied (see for example the survey by Alstrup et al. [AGKR02]). We show the lemma for any $d \geq 2$ by using an extension of a technique mentioned in [AGKR02]. Actually, we show Lemma 1 as a corollary to Lemma 2:

Lemma 2. *Under the assumptions of Lemma 1, we can create a structure using $O(m)$ space such that, given a query $((q_1, q_1'), q_3, \ldots q_d)$, we can, in constant time, either find the point $(p_1, \ldots, p_d) \in S$ with $q_1 \leq p_1 \leq q_1'$ and $p_3 \geq q_3, \ldots, p_d \geq q_d$ such that p_2 is maximized, or determine that no such point exists.*

We obtain Lemma 1 from Lemma 2 as follows. Suppose we are given a query $((q_1, q_1'), q_2, \ldots q_d)$ in Lemma 1. We then ask the query $((q_1, q_1'), q_3, \ldots q_d)$ in Lemma 2. If we do not get a point we stop, else let $p = (p_1, \ldots, p_d) \in S$ be the point we get. If $p_2 < q_2$ we stop. Otherwise, we report p and recursively ask the two queries $((q_1, p_1 - 1), q_3, \ldots, q_d)$ and $((p_1 + 1, q_1'), q_3, \ldots, q_d)$ in Lemma 2.

The rest of this section is devoted to the proof of Lemma 2. We assume m is a power of two and define $c = \lceil \log^{\epsilon} n \rceil$. We sort the points in increasing order by their 1-coordinates and group them into blocks each with $c^{d-2} \log m$ elements. We define for each block b a *1-range* $[b.l..b.r]$, where $b.l$ (resp. $b.r$) is the smallest

(resp. largest) 1-coordinate of a point in b. We create a binary tree T built on the blocks b sorted in increasing order by $b.l$. We associate with each leaf v of T two arrays v.left and v.right, both indexed by $[0..\log m] \times [1..c]^{d-2}$. Let u be the ancestor of v whose height is h (the height of a leaf is 0). Then $v.\text{left}[h, q_3, \ldots q_d]$ (resp. $v.\text{right}[h, q_3, \ldots q_d]$) stores the point $p = (p_1, \ldots, p_d)$ such that (i) p belongs to a block corresponding to a leaf between v and the leftmost (resp. rightmost) leaf node of the subtree rooted at u; (ii) $p_3 \geq q_3, \ldots p_d \geq q_d$; and (iii) p_2 is maximized, provided that such a point p exists. We note that, since each of the $O(m/(c^{d-2} \log m))$ leaves of T contains two arrays both with $O(c^{d-2} \log m)$ elements, the total space used so far is $O(m)$.

The data structure we just described has already enabled us to answer a query $((q_1, q_1'), q_3, \ldots q_d)$ where $q_1 = b.l$ and $q_1' = b'.r$ for two blocks $b \neq b'$. We first locate, in constant time, the nearest common ancestor u of the two leaves corresponding to b and b'. This can be done in constant time using the msb operation or table lookup. Let h be the height of u. Then the point that satisfies the query (if any) must be in either $b.\text{right}[h-1, q_3, \ldots, q_d]$ or $b'.\text{left}[h-1, q_3, \ldots, q_d]$.

In order to be able to answer general queries where the range $[q_1, q_1']$ may partially intersect the 1-ranges of some blocks, we create an extra structure for each block b as follows. We build a constant-height tree $b.T$ with degree $\Theta(\log^\delta n)$, for a constant $\delta > 0$ to be determined later, over the points of b sorted in increasing order by their 1-coordinates. At each internal node $v \in b.T$ we keep for each pair (v_1, v_2) of its children, where v_1 is to the left of v_2, an array $v.R_{(v_1,v_2)}$ indexed by $[1..c]^{d-2}$. The entry $(q_3, \ldots, q_d)$ of this array identifies the point $p = (p_1, \ldots, p_d)$ in b such that (i) p is in a leaf below a child of v between v_1 and v_2; (ii) $p_3 \geq q_3, \ldots, p_d \geq q_d$; and (iii) p_2 is maximized, provided that such a point exists. Since within b a point can be uniquely identified using $O(\log \log n)$ bits, the total bit-cost of each internal node of $v.T$ is $O(c^{d-2} \log \log n \log^{2\delta} n)$. It follows that an internal node fits into a single word (and thus the total space usage will be linear), if $2\delta + (d-2)\epsilon < 1$, which can be satisfied by choosing $\delta = (1 - (d-2)\epsilon)/3$.

We now describe how to answer a query of the form $((q_1, q_1'), q_3, \ldots, q_d)$ where $q_1, q_1' \in [b.l..b.r]$ for some block b. Let u be the nearest common ancestor of the leaves in $b.T$ corresponding to q_1 and q_1', and let Π_1 and Π_2 be respectively the paths from u to these two leaves. The answer to the query can be chosen from a constant number of "candidate points", each picked at a node on Π_1 or Π_2 from one of its associated arrays. For example, without loss of generality, consider the node u. Suppose its ith and jth children are respectively on paths Π_1 and Π_2, and assume $i < j - 1$. Then the candidate point contributed by u can be found in constant time at $u.R_{(v_1,v_2)}[q_3, \ldots, q_d]$, where v_1 and v_2 are respectively the $(i+1)$th and $(j-1)$th children of u.

Finally suppose we are given a query $((q_1, q_1'), q_3, \ldots, q_d)$ where $q_1 \in [b.l..b.r]$ and $q_1' \in [b'.l..b'.r]$ for two different blocks b and b', which can be easily identified in constant time. The output of the query is then one of the three points

returned by the queries $((q_1, b.r), q_2, \ldots, q_d)$, $((b'.l, q'_1), q_2, \ldots, q_d)$, and $(b.r + 1, b'.l - 1, q_2, \ldots, q_d)$, the handling of which has already been described.

3.2 $(3, d, \epsilon)$-dimensional Dominance Reporting

In this section we show the following lemma:

Lemma 3. *Let $d \geq 3$ and $0 < \epsilon < \min(1/4, 1/(d-2))$ be constants and assume S is a $(3, d, \epsilon)$-dimensional point set of size $m \leq n$. Then there exists a solution to the dominance reporting problem for S using $O(m)$ space such that a query reporting f points can be answered in $O(\log m / \log\log n + f)$ time.*

We say a point $p = (p_1, \ldots, p_d)$ 2-*dominates* a point $q = (q_1, \ldots, q_d)$ if $p_i \geq q_i$, for $1 \leq i \leq 2$. We say a subset M of a point set P is 2-*maximal* if $p \in M$, $q \in P$ and $p \neq q$ implies that p does not 2-dominate q.

We build a search tree T with degree $c = \lceil \log^{\epsilon} n \rceil$ and height $O(\log m / \log\log n)$ over the points from S sorted in increasing order by their 3-coordinates. At each internal node $v \in T$ we store the keys to guide the search in T in a q-heap. For each internal or leaf node $v \in T$ we define $M(v)$ to be the largest 2-maximal set of the points stored in the leaves of the subtree rooted at v, excluding those in $M(v')$ for any ancestor v' of v. We keep each point $(p_1, \ldots, p_d) \in M(v)$ as the $(2, d-1, \epsilon)$-dimensional point $(p_1, p_3, p_4, \ldots, p_d)$ in the 3-sided reporting structure $D(v)$ of Lemma 1. For the moment we ignore the requirement in Lemma 1 that the 1-coordinate must be in rank space. At each internal node v, we also store another data structure $G(v)$ containing the points in $M(v')$ for all the children v' of v. A point $(p_1, \ldots, p_d)$ from the ith child of v (counted from the left) is stored in $G(v)$ as the $(2, d, \epsilon)$-dimensional point $(p_1, p_2, i, p_4, \ldots, p_d)$. $G(v)$ is also a 3-sided reporting structure of Lemma 1, again ignoring the rank space requirement.

Now suppose we are given a query $q = (q_1, \ldots, q_d)$. We first identify the path Π in T from the root to the leaf corresponding to the 3-successor of q_3. We can find Π in time $O(\log m / \log\log n)$ by using the q-heaps associated with the nodes of T. We then visit the root of T as described in the following. Actually, we will describe how to visit an arbitrary node $v \in T$. We first report the points from $M(v)$ which dominate q by performing a query $q' = ((q_1, q'_1), q_3, q_4, \ldots, q_d)$ in $D(v)$, where q'_1 is the 1-coordinate of the 2-successor of q_2 in $M(v)$ (we will explain later how to find this 2-successor in constant time). The points returned by this query are exactly the points in $M(v)$ that dominate q. This is due to the fact that for any two points $r = (r_1, r_2, \ldots, r_d)$ and $s = (s_1, s_2, \ldots, s_d)$ in $M(v)$, $r_1 > s_1$ if and only if $r_2 < s_2$ (see [MT98] for more details). Next, suppose the kth child of v is on Π (we set $k = 0$ if v is not on Π). We then perform the query $q'' = ((q_1, \infty), q_2, k+1, q_3, \ldots, q_d)$ in $G(v)$. If the answer to q'' contains a point $(p_1, p_2, i, p_4, \ldots, p_d)$, we recursively visit the ith child of v. If v has a child on Π we also recursively visit that child (such child exists if and only if v is an internal node on Π).

We now address the issue that the 1-coordinate of the points in $M(v)$ and $G(v)$ for a node v in T has to be in rank space in order for Lemma 1 to be

applicable, and that the 2-successor of q_2 in $M(v)$ has to be identified in constant time. The first issue is resolved by replacing the 1-coordinate of each point in $M(v)$ (resp. $G(v)$) with its 1-rank with respect to $M(v)$ (resp. $G(v)$). Accordingly, before the query q' (resp. q'') is applied to $M(v)$ (resp. $G(v)$) we replace q_1 with its 1-rank in $M(v)$ (resp. $G(v)$) (and, in the case of query q', replace q'_1 with the 1-rank of the 2-successor of q_2). Computing the 1-rank and the 2-successor of an integer in $M(v)$ and $G(v)$ for each node visited is exactly the *iterative search* problem defined in [CG86], which can be handled in $O(1)$ time if v is not the root of T and in $O(\log m/\log\log n)$ time if v is the root, using the fast fractional cascading technique of [SJ03b], which requires $O(m)$ space. (Notice that the standard fractional cascading technique described in [CG86] will not work here because the degree of T is not a constant.)

Note that a point in S is stored at most twice, once in $D(v)$ for a node v and once in $G(u)$, where u is the parent of v. The data structures of types D and G are all linear-space data structures. Therefore the overall space usage is $O(m)$. Further, it is easy to see that the number of nodes of T visited is $O(\log m/\log\log n + f)$ and that searching $D(v)$ and $G(v)$ at each such node v takes $O(1)$ time per reported point, hence the claimed querying complexity.

4 Two-dimensional Dominance Counting

The goal of this section is to design a data structure to solve the dominance counting problem for a set with dimension $(2, d, \epsilon)$ for $d \geq 2$, thus proving Theorem 2 for $d = 2$. We give this structure in Section 4.2. But first, we give, in Section 4.1, a solution with sub-linear space usage for a set with dimension $(1, d, \epsilon)$, for $d \geq 1$, where the 1-coordinate is in rank space.

4.1 $(1, d, \epsilon)$-dimensional Dominance Counting

This section is devoted to the proof of the following lemma:

Lemma 4. *Let $d \geq 1$ and $0 < \epsilon < 1/(d-1)$ be constants and assume S is an $(1, d, \epsilon)$-dimensional point set of size $m \leq n$, where the 1-coordinate is in rank space. Then there exists a solution to the dominance counting problem for S using $O(m \log\log n)$ bits of space such that queries can be answered in constant time.*

We assume m is a power of two and define $c = \lceil \log^{\epsilon} n \rceil$. We sort the points in increasing order by their 1-coordinates and group them into blocks each with $c^{d-1} \log m$ elements. Further, we partition each block into subblocks each with c^{d-1} elements. We label each block (resp. subblock) with the largest 1-coordinate of a point in that block (resp. subblock). For each block or subblock b we keep an array b.count indexed by $[1..c]^{d-1}$. For each block b we set $b.\text{count}[q_2, \ldots, q_d]$ to be the number of points in S dominating $(i+1, q_2, \ldots, q_d)$ where i is the label of b. For each subblock b' of a block b we set $b'.\text{count}[q_2, \ldots, q_d]$ to be the number of points in b dominating $(i+1, q_2, \ldots, q_d)$ where i is the label of b'. Finally,

we encode the points of a subblock b' in $O(c^{d-1}\log c) = o(\log n)$ bits which we keep in a single word. This is possible since each of the c^{d-1} points in b' can be encoded in $O(\log c)$ bits.

Suppose we are given a query $q = (q_1, \ldots, q_d)$. We first identify, in constant time, the block (resp. subblock) b (resp. b') with the smallest label greater than or equal to q_1. We now describe how to find the number e of points in b' that dominate q. Notice that a query q with respect to a subblock can be described in $O(\log\log n)$ bits. To compute e in constant time, all we need is to append q to the description of b' and use the result to look up a global table, which requires $O(n)$ words since $O(c^{d-1}\log c\log\log n) = o(\log n)$. The answer to the query q is then $e + b.\text{count}(q_2, \ldots, q_d) + b'.\text{count}(q_2, \ldots, q_d)$.

We now analyze the space usage of the structure. Each array of the $m/(c^{d-1}\log m)$ blocks contains c^{d-1} elements each of $\log m$ bits. It follows that the space used by these arrays is $O(m)$ bits. Each array of the m/c^{d-1} subblocks also contains c^{d-1} elements but each element is at most $c^{d-1}\log m$. It follows that each element can be represented by $O(\log\log n)$ bits so the total space used by the arrays associated with the subblocks is $O(m\log\log n)$ bits. Finally, each point of a subblock can be represented by $O(\log\log n)$ bits and it follows that the total space usage becomes $O(m\log\log n)$ as claimed.

4.2 $(2, d, \epsilon)$-dimensional Dominance Counting

In this section we show:

Lemma 5. *Let $d \geq 2$ and $0 < \epsilon < \min(1/4, 1/(d-1))$ be constants and let S be a $(2, d, \epsilon)$-dimensional point set of size $m \leq n$. Then there exists a solution to the dominance counting problem for S using $O(m)$ space such that queries can be answered in $O(\log m/\log\log n)$ time.*

We will prove the lemma under the assumption that the first coordinate is in rank space. Since the targeted query time is $O(\log m/\log\log n)$ time, we can easily remove this assumption by transforming the 1-coordinates of the points in S as well as the 1-coordinate of the query point into rank space by creating a search tree with degree $(\log^{\delta} n)$ on the 1-coordinates of the points, where $\delta < 1/4$, and storing the keys at each node of this tree in a q-heap.

We create a search tree T with degree $c = \lceil \log^{\epsilon} n \rceil$ and height $O(\log m/\log\log n)$ over the points from S sorted in increasing order by their 2-coordinates. At each internal node $v \in T$ we store the keys to guide the search in v in a q-heap. We define $G(v)$ to be the set of points stored in the subtree rooted at v. Let $p = (p_1, \ldots, p_d) \in G(v)$ be a point stored at a leaf descendant of the jth child of v, and let i be the 1-rank of p in $G(v)$. Then we store p as $(i, j, p_3, \ldots, p_d)$ in a structure $v.D$ of Lemma 4 with dimension $(1, d, \epsilon)$.

Now assume that we are given a query $q = (q_1, \ldots, q_d)$. We first identify the path Π in T from the root to the leaf storing the 2-successor of q_2. We can find Π in time $O(\log m/\log\log n)$ by using the q-heaps stored at the nodes of T. The answer to q is the sum of the answers to $\Theta(\log m/\log\log n)$ $(1, d, \epsilon)$-dimensional dominance counting queries, each applied to the structure $v.D$ for a node v on

Π. For the root w, suppose the jth child of w is also on Π. Then the query applied to $w.D$ is $(q_1, j+1, q_3, \ldots, q_d)$. For the leaf that is on Π, we check the point stored there directly. Now consider a non-root internal node v on Π. Let u be its parent. (Note that u is also on Π.) Suppose the jth child of v is also on Π. Note that we already know the 1-rank $r_1(u)$ of q_1 in $G(u)$ ($r_1(w) = q_1$). The query applied to $v.D$ is $(r_1(v), j+1, q_3, \ldots, q_d)$, where $r_1(v)$ is the 1-rank of q_1 in $G(v)$. The value of $r_1(v)$ can be computed by performing a constant number of $(1, d, \epsilon)$-dimensional dominance queries in $u.D$. In fact, let $G_>(v)$ denote the union of $G(v')$ for all the right siblings v' of v, and let $G_\geq(v) = G(v) \cup G_>(v)$. Then $r_1(v)$ is the difference between the 1-rank of q_1 in $G_\geq(v)$ and the 1-rank of q_1 in $G_>(v)$. The 1-rank of q_1 in $G_\geq(v)$ is $|G_\geq(v)| - k_\geq(v) + 1$, where $k_\geq(v)$ is the number of points in $G_\geq(v)$ whose 1-coordinate is greater than or equal to q_1. $|G_\geq(v)|$ can be computed by performing the query $(0, j, q_3, \ldots, q_d)$ in $u.D$ and $k_\geq(v)$ can be computed by performing the query $(r_1(u), j, q_3, \ldots, q_d)$ in $u.D$. The 1-rank of q_1 in $G_>(v)$ can be computed similarly.

Since we only use constant time at each node of Π, the overall query time is $O(\log m / \log\log n)$. Furthermore, we store each of the m points of S in $O(\log m / \log\log n)$ structures, each of which uses $O(\log\log n)$ bits. Therefore the total space usage becomes $O(m)$ as claimed.

5 Higher Dimensional Dominance Reporting and Counting

In this section we give a general construction in Lemma 6 which can be used to obtain a structure for a $(d, d, 0)$-dimensional point set from a structure for a (d', d, ϵ)-dimensional point set, where $d' < d$. We then use this construction to prove Theorem 1 and 2 from Lemma 3 and 5 respectively. A similar construction was given in [Mor03] for orthogonal range reporting for a dynamic set of points.

Let $(G, +)$ be a semi-group and let S be a set with dimension (d', d, ϵ). Assume each point $p \in S$ has an associated semigroup element $g(p) \in G$. The (d', d, ϵ)-*dimensional dominance semigroup* problem is defined as follows. Given a (d', d, ϵ)-dimensional query point q, find the semigroup sum $\sum_{p \in S: p \text{ dominates } q} g(p)$. We will show:

Lemma 6. *Assume $2 \leq d' \leq d$ and $0 < \epsilon < 1/4$ are constants. Assume we have a data structure structure D' for a $(d'-1, d, \epsilon)$-dimensional dominance semigroup problem of size m' for any $m' \leq n$. Then, we can derive a structure D for the (d', d, ϵ)-dimensional dominance semigroup problem of size $m \leq n$. Further:*

1. *For every point in D we store $O(\log m / \log\log n)$ points in structures of type D'.*
2. *Given a query in D we can answer it by performing $O(\log m / \log\log n)$ queries in the structures of type D' and return the semigroup sum as the result.*

The space usage besides the space usage in item 1 is $O(n)$ and the queries to be performed in item 2 can be determined in constant time per query.

The dominance reporting problem can be seen as a special case of the dominance semigroup problem if we define the elements in G as point sets, $g(p) = \{p\}$, and select $+$ to be the union operator on sets. Theorem 1 then follows by applying Lemma 6 $d-3$ times to Lemma 3. Similarly, the dominance counting problem can be seen as a special case of the dominance semigroup problem, where the elements in G are non-negative integers, $g(p) = 1$, and $+$ is the integer addition. Theorem 2 then follows by applying Lemma 6 $d-2$ times to Lemma 5.

We now prove Lemma 6. Let S be a set of m (d', d, ϵ)-dimensional points. We build a tree T with degree $c = \lceil \log^{\epsilon} m \rceil$ over the points in S sorted by their d'-coordinates. At each internal node $v \in T$ we keep a dominance semigroup structure $v.D'$ with dimension $(d'-1, d, \epsilon)$ containing the points stored in the subtree rooted at v. A point $p = (p_1, \ldots, p_d)$ from the ith child of v is stored in $v.D'$ as $p' = (p_1, \ldots, p_{d'-1}, i, p_{d'+1}, \ldots, p_d)$ with $g(p') = g(p)$. Suppose now we are given a query $q = (q_1, \ldots, q_d)$ in D with dimension (d', d, ϵ). We first identify the path Π in T from the root to the leaf corresponding to the d'-successor of $q_{d'}$, which can be found in time $O(\log n / \log\log n)$ using the q-heaps in the nodes of T. For each internal node $v \in \Pi$, assume that the j_vth child of v is also on Π. The answer to the query in D is then the semigroup sum of the queries $(q_1, \ldots, q_{d'-1}, j_v + 1, q_{d'+1}, \ldots, q_d)$ in $v.D'$ for every $v \in \Pi$. This finishes the proof of Lemma 6.

References

[ABR00] Stephen Alstrup, Gerth Støling Brodal, and Theis Rauhe. New data structures for orthogonal range searching. In *Proceedings of IEEE Symposium on Foundations of Computer Science*, pages 198–207, Redondo Beach, CA, 2000.

[AGKR02] S. Alstrup, C. Gavoille, H. Kaplan, and T. Rauhe. Nearest common ancestors: A survey and a new distributed algorithm. In *Proceedings of the 14th ACM Symp. on Parallel Algorithms and Architecture (SPAA)*, pages 258–264, August 2002.

[Ben80] Jon Louis Bentley. Multidimensional divide-and-conquer. *Communications of the ACM*, 23(4):214–229, April 1980.

[CE87] Bernard Chazelle and H. Edelsbrunner. Linear space data structures for two types of range search. *Discrete Comput. Geom.*, 3:113–126, 1987.

[CG86] Bernard Chazelle and Leonidas J. Guibas. Fractional Cascading: I. A data structure technique. *Algorithmica*, 1(2):133–162, 1986.

[Cha88] Bernard Chazelle. A functional approach to data structures and its use in multidimensional searching. *SIAM Journal on Computing*, 17(3):427–463, June 1988.

[EO82] H. Edelsbrunner and M.H. Overmars. On the equivalence of some rectangle problems. *Information Processing Letters*, 14:124–127, 1982.

[FW93] Michael L. Fredman and Dan E. Willard. Surpassing the information theoretic bound with fusion trees. *Journal of Computer and System Sciences*, 47:424–436, 1993.

[FW94] Michael L. Fredman and Dan E. Willard. Trans-dichotomous algorithms for minimum spanning trees and shortest paths. *Journal of Computer and System Sciences*, 48:533–551, 1994.

[GAA03] Sathish Govindarajan, Pankaj K. Agarwal, and Lars Arge. CRB-Tree: an efficient indexing scheme for range-aggregate queries. In *Proceedings of the 9th International Conference on Database Theory*, Siena, Italy, 2003.

[Mor03] Christian Worm Mortensen. Fully-dynamic orthogonal range reporting on RAM (Preliminary version). Technical Report TR-2003-22, The IT University of Copenhagen, 2003.

[MT98] C. Makris and A. K. Tsakalidis. Algorithms for three-dimensional dominance searching in linear space. *Information Processing Letters*, 66(6):277–283, 1998.

[SJ03a] Qingmin Shi and Joseph JaJa. Fast algorithms for 3-d dominance reporting and counting. Technical Report CS-TR-4437, Institute of Advanced Computer Studies (UMIACS), University of Maryland, 2003.

[SJ03b] Qingmin Shi and Joseph JaJa. Fast fractional cascading and its applications. Technical Report CS-TR-4502, Institute of Advanced Computer Studies (UMIACS), University of Maryland, 2003.