Path and Ancestor Queries over Trees with Multidimensional Weight Vectors

Meng He
Faculty of Computer Science, Dalhousie University, Canada
mhe@cs.dal.ca

Serikzhan Kazi
Faculty of Computer Science, Dalhousie University, Canada
skazi@dal.ca

──── **Abstract** ────

We consider an ordinal tree T on n nodes, with each node assigned a d-dimensional weight vector $\mathbf{w} \in \{1, 2, \ldots, n\}^d$, where $d \in \mathbb{N}$ is a constant. We study path queries as generalizations of well-known *orthogonal range queries*, with one of the dimensions being tree topology rather than a linear order. Since in our definitions d only represents the number of dimensions of the weight vector without taking the tree topology into account, a path query in a tree with d-dimensional weight vectors generalize the corresponding $(d+1)$-dimensional orthogonal range query. We solve *ancestor dominance reporting* problem as a direct generalization of dominance reporting problem, in time $\mathcal{O}(\lg^{d-1} n + k)$ and space of $\mathcal{O}(n \lg^{d-2} n)$ words, where k is the size of the output, for $d \geq 2$. We also achieve a tradeoff of $\mathcal{O}(n \lg^{d-2+\epsilon} n)$ words of space, with query time of $\mathcal{O}((\lg^{d-1} n)/(\lg \lg n)^{d-2} + k)$, for the same problem, when $d \geq 3$. We solve *path successor problem* in $\mathcal{O}(n \lg^{d-1} n)$ words of space and time $\mathcal{O}(\lg^{d-1+\epsilon} n)$ for $d \geq 1$ and an arbitrary constant $\epsilon > 0$. We propose a solution to *path counting problem*, with $\mathcal{O}(n(\lg n/ \lg \lg n)^{d-1})$ words of space and $\mathcal{O}((\lg n/ \lg \lg n)^d)$ query time, for $d \geq 1$. Finally, we solve *path reporting problem* in $\mathcal{O}(n \lg^{d-1+\epsilon} n)$ words of space and $\mathcal{O}((\lg^{d-1} n)/(\lg \lg n)^{d-2} + k)$ query time, for $d \geq 2$. These results match or nearly match the best tradeoffs of the respective range queries. We are also the first to solve path successor even for $d = 1$.

1 Introduction

The problem of preprocessing a weighted tree, i.e., a tree in which each node is associated with a weight value, to support various queries evaluating a certain function on the node weights of a given path, has been extensively studied [2, 6, 12, 17, 8, 15, 4]. For example, in *path counting* (resp. *path reporting*), the nodes of the given path with weights lying in the given query interval are counted (resp. reported). These queries address the needs of fast information retrieval from tree-structured data such as XML and tree network topology.

For many applications, meanwhile, a node in a tree is associated with not just a single weight, but rather with a vector of weights. Consider a simple scenario of an online forum thread, where users can rate responses and respond to posts. Induced is a tree-shaped structure with posts representing nodes, and replies to a post being its children. One can imagine enumerating all the ancestor posts of a given post that are not too short and have sufficiently high average ratings. Ancestor dominance query, which is among the problems we consider, provides an appropriate model in this case.

We define a *d-dimensional weight vector* $\mathbf{w} = (w_1, w_2, \ldots, w_d)$ to be a vector with d components, each in rank space $[n]$, [1] i.e. $\mathbf{w} \in [n]^d$, with w_i being referred to as the ith weight of $\mathbf{w}$. We then consider an ordinal tree T on n nodes, each node x of which is assigned a d-dimensional weight vector $\mathbf{w}(x)$. The queries we will define all give a d-dimensional orthogonal range $Q = \prod_{i=1}^{d}[q_i, q_i']$, and a weight vector $\mathbf{w}$ is in Q *iff* for any $i \in [1, d]$, $q_i \leq w_i \leq q_i'$ holds. In our queries, then, we are given a pair of vertices $x, y \in T$, and an arbitrary orthogonal range Q. With $P_{x,y}$ being the path from x to y in the tree T, the goal is to preprocess the tree T for the following types of queries:

- *Path Counting*: return $|\{z \in P_{x,y} \mid \mathbf{w}(z) \in Q\}|$.
- *Path Reporting*: enumerate $\{z \in P_{x,y} \mid \mathbf{w}(z) \in Q\}$.
- *Path Successor*: return $\mathtt{argmin}\{w_1(z) \mid z \in P_{x,y} \text{ and } \mathbf{w}(z) \in Q\}$.[2]
- *Ancestor Dominance Reporting*: a special case of path reporting, in which y is the root of the tree and $q_i' = +\infty$ for all $i \in [d]$. That is, the query reports the ancestors of x whose weight vectors dominate the vector $\mathbf{q} = (q_1, q_2, \ldots, q_d)$.

This indeed is a natural generalization of the traditional weighted tree, which we refer to as "scalarly-weighted", to the case when the weights are multidimensional vectors. At the same time, when the tree degenerates into a single path, these queries become respectively $(d+1)$-dimensional orthogonal range counting, reporting, successor, as well as $(d+1)$-dimensional dominance reporting queries. Thus, the queries we study are generalizations of these fundamental geometric queries in high dimensions. We also go along with the state-or-art in orthogonal range search by considering weights in rank space, since the case in which weights are from a larger universe can be reduced to it [10].

1.1 Previous Work

Path Queries in Weighted Trees. For scalarly-weighted trees, Chazelle [6] gave an $\mathcal{O}(n)$-word *emulation dag*-based data structure that answers path counting queries in $\mathcal{O}(\lg n)$ time;[3] it works primarily with topology of the tree and is thus oblivious to the distribution of weights. Later, He et al. [15] proposed a solution with $nH(W_T) + o(n \lg \sigma)$ bits of space and $\mathcal{O}(\frac{\lg \sigma}{\lg \lg n} + 1)$ query time, when the weights are from $[\sigma]$; here, $H(W_T)$ is the entropy of the multiset of the weights in T. When $\sigma \ll n$, this matters.

He et al. [15] introduced and solved path reporting problem using linear space and $\mathcal{O}((1+k)\lg \sigma)$ query time, and $\mathcal{O}(n \lg \lg \sigma)$ words of space but $\mathcal{O}(\lg \sigma + k \lg \lg \sigma)$ query time, in the word-RAM model; henceforth we reserve k for the size of the output. Patil et al. [21] presented a succinct data structure for path reporting with $n \lg \sigma + 6n + o(n \lg \sigma)$ bits of space and $\mathcal{O}((\lg n + k) \lg \sigma)$ query time. An optimal-space solution with $nH(W_T) + o(n \lg \sigma)$ bits of space and $\mathcal{O}((1 + k)(\frac{\lg \sigma}{\lg \log n} + 1))$ reporting time is due to He et al. [15]. One of the tradeoffs proposed by Chan et al. [4], requires $\mathcal{O}(n \lg^{\epsilon} n)$ words of space for the query time of $\mathcal{O}(\lg \lg n + k)$.

Orthogonal Range Queries. Dominance reporting in $3D$ was solved by Chazelle and Edelsbrunner [7] in linear space with either $\mathcal{O}((1+k)\lg n)$ or $\mathcal{O}(\lg^2 n + k)$ time, in pointer-machine (PM) model, with the latter being improved to $\mathcal{O}(\lg n \lg \lg n + k)$ by Makris and Tsakalidis [18]. Same authors [18] developed, in the word-RAM, a linear-size, $\mathcal{O}(\log n + k)$

---

[1] Throughout this paper, $[n]$ stands for $\{1, 2, \ldots, n\}$ for any positive integer n.
[2] For path successor, we assume that $q_1' = \infty$; if not, we need only check whether the 1st weight of the returned node is at most q_1'.
[3] $\lg x$ denotes $\log_2 x$ in this paper.

and $\mathcal{O}((\lg \lg n \lg \lg \lg n + k) \lg \lg n)$ query-time data structures for the unrestricted case and for points in rank space, respectively. Nekrich [19] presented a word-RAM data structure for points in rank space, supporting queries in $\mathcal{O}((\lg \lg n)^2 + k)$ time, and occupying $\mathcal{O}(n \lg n)$ words; this space was later reduced to linear by Afshani [1], retaining the same query time. Finally, in the same model, a linear-space solution with $\mathcal{O}(\log \log n + k)$ query time was designed for $3D$ dominance reporting in rank space [1, 3]. In the PM model, Afshani [1] also presented an $\mathcal{O}(\log n + k)$ query time, linear-space data structure for the points in $\mathbb{R}^3$.

For the word-RAM model, JáJá et al. [16] generalized the range counting problem for $d \geq 2$ dimensions and proposed a data structure with $\mathcal{O}(n(\lg n / \lg \lg n)^{d-2})$ words of space and $\mathcal{O}((\lg n / \lg \lg n)^{d-1})$ query time. Chan et al. [5] solved orthogonal range reporting in $3D$ rank space in $\mathcal{O}(n \lg^{1+\epsilon} n)$ words of space and $\mathcal{O}(\lg \lg n + k)$ query time.

Nekrich and Navarro [20] proposed two tradeoffs for range successor, with either $\mathcal{O}(n)$ or $\mathcal{O}(n \lg \lg n)$ words of space, and respectively with $\mathcal{O}(\lg^\epsilon n)$ or $\mathcal{O}((\lg \lg n)^2)$ query time. Zhou [23] later improved upon the query time of the second tradeoff by a factor of $\lg \lg n$, within the same space. Both results are for points in rank space.

1.2 Our Results

As d-dimensional path queries generalize the corresponding $(d+1)$-dimensional orthogonal range queries, we compare results on them to show that our bounds match or nearly match the best results or some of the best tradeoffs on geometric queries in Euclidean space. We present solutions for the (we assume d is a positive integer constant):

- ancestor dominance reporting problem, in $\mathcal{O}(n \lg^{d-2} n)$ words of space and $\mathcal{O}(\lg^{d-1} n + k)$ query time for $d \geq 2$. When $d = 2$, this matches the space bound for $3D$ dominance reporting of [1, 3], while still providing efficient query support. When $d \geq 3$, we also achieve a tradeoff of $\mathcal{O}(n \lg^{d-2+\epsilon} n)$ words of space, with query time of $\mathcal{O}(\lg^{d-1} n/(\lg \lg n)^{d-2} + k)$;

- path successor problem, in $\mathcal{O}(n \lg^{d-1} n)$ words and $\mathcal{O}(\log^{d-1+\epsilon} n)$ query time, for an arbitrarily small positive constant ϵ, and $d \geq 2$. These bounds match the first tradeoff for range successor of Nekrich and Navarro [20]. [4] Previously this problem has not been studied even on scalarly-weighted trees;

- path counting problem, in $\mathcal{O}(n(\frac{\log n}{\log \log n})^{d-1})$ words of space and $\mathcal{O}((\frac{\log n}{\log \log n})^d)$ query time for $d \geq 1$. This matches the best bound for range counting in $d + 1$ dimensions [16];

- path reporting problem, in $\mathcal{O}(n \lg^{d-1+\epsilon} n)$ words of space and $\mathcal{O}((\lg^{d-1} n)/(\lg \lg n)^{d-2} + k)$ query time, for $d \geq 2$. When $d = 2$, the space matches that of the corresponding result of Chan et al. [5] on $3D$ range reporting, while the first term in the query complexity is slowed down by a sub-logarithmic factor.

To achieve our results, we introduce a framework for solving range sum queries in arbitrary semigroups and extend base-case data structures to higher dimensions using universe reduction. A careful design with results hailing from succinct data structures and tree representations has been necessary, both for building space- and time-efficient base data structures, and for porting, using *tree extractions*, the framework of range trees decompositions from general point-sets to tree topologies (Lemma 5). We employ a few novel techniques, such as extending the notion of *maximality* in Euclidean sense to tree topologies, and providing the means of efficient computation thereof (Section 5). Given a weighted tree T, we propose efficient means of zooming into the nodes of T with weights in the given range in the range tree (Lemma 13). Given the ubiquitousness of the concepts, these technical contributions are likely to be of independent interest.

---

[4] which can be generalized to higher dimensions via standard techniques based on range trees

Described further are our solutions to ancestor dominance reporting and path successor problems, while our data structures for path counting and path reporting are deferred to the full version of this paper.

2 Preliminaries

Notation.   Given a d-dimensional weight vector $\mathbf{w} = (w_1, w_2, \ldots, w_d)$, we define vector $\mathbf{w}_{i,j}$ to be $(w_i, w_{i+1}, \ldots, w_j)$. We extend the definition to a range $Q = \prod_{i=1}^{d}[q_i, q_i']$ by setting $Q_{i,j} = \prod_{k=i}^{j}[q_k, q_k']$. We use the symbol $\succeq$ for domination: $\mathbf{p} \succeq \mathbf{q}$ *iff* $\mathbf{p}$ dominates $\mathbf{q}$. With $d' \leq d$ and $0 < \epsilon < 1$ being constants, a weight vector $\mathbf{w}$ is said to be $(d', d, \epsilon) - dimensional$ *iff* $\mathbf{w} \in [n]^{d'} \times [[\lceil \lg^\epsilon n \rceil]]^{d-d'}$; i.e., each of its first d' weights is drawn from $[n]$, while each of its last $d - d'$ weights is in $[[\lceil \lg^\epsilon n \rceil]]$. When stating theorems, we define $i/0 = \infty$ for $i > 0$.

During a preorder traversal of a given tree T, the ith node visited is said to have *preorder rank i*. Preorder ranks are commonly used to identify tree nodes in various succinct data structures which we use as building blocks. Thus, we also identify a node by its preorder rank, i.e., node i in T is the node with preorder rank i in T. The path between the nodes $x, y \in T$ is denoted as $P_{x,y}$, both ends inclusive. For a node $x \in T$, its set of ancestors, denoted as $\mathcal{A}(x)$, includes x itself; $\mathcal{A}(x) \setminus \{x\}$ is then the set of *proper ancestors* of x. Given two nodes $x, y \in T$, where $y \in \mathcal{A}(x)$, we set $A_{x,y} \triangleq P_{x,y} \setminus \{y\}$.

Succinct Representations of Ordinal Trees.   Succinct representations of unlabeled and labeled ordinal trees is a widely researched area. In a labeled tree, each node is associated with a label over an alphabet. Such a label can serve as a scalar weight; in our solutions, however, they typically categorize tree nodes into different classes. Hence we call these assigned values labels instead of weights. We summarize the previous result used in our solutions, in which a node (resp. ancestor) with label α is called an α-*node* (resp. α-*ancestor*):

▶ **Lemma 1** ([15, 13]). *Let T be an ordinal tree on n nodes, each having a label drawn from $[\sigma]$, where $\sigma = \mathcal{O}(\lg^\epsilon n)$ for some constant $0 < \epsilon < 1$. Then, T can be represented in $n(\lg \sigma + 2) + o(n)$ bits of space to support the following operations, for any node $x \in T$, in $\mathcal{O}(1)$ time:* $\mathtt{child}(T, x, i)$*, the i-th child of x;* $\mathtt{depth}(T, x)$*, the number of ancestors of x;* $\mathtt{level\_anc}(T, x, i)$*, the i-th lowest proper ancestor of x;* $\mathtt{pre\_rank}_\alpha(T, x)$*, the number of α-nodes that precede x in preorder;* $\mathtt{pre\_select}_\alpha(T, i)$*, the i-th α-node in preorder; and* $\mathtt{level\_anc}_\alpha(T, x, i)$*, the i-th lowest α-ancestor of x.*

Lemma 1 also includes a result on representing an unlabeled ordinal tree, which corresponds to $\sigma \equiv 1$, in $2n + o(n)$ bits [13]. Another important special case is that of $\sigma = 2$; here, T is referred to as a $0/1$-*labeled tree*, and the storage space becomes $3n + o(n)$ bits.

Tree Extraction.   *Tree extraction* [15] filters out a subset of nodes while preserving the underlying ancestor-descendant relationship among the nodes. Namely, given a subset X of tree nodes called *extracted nodes*, an *extracted tree T_X* can be obtained from the original tree T as follows. Consider each node $v \notin X$ in an arbitrary order; let p be v's parent. We remove v and all its incident edges from T, and plug all its children $v_1, v_2, \ldots, v_k$ (preserving their left-to-right order) into the slot now freed from v in p's list of children. After removing all the non-extracted nodes, if the resulting forest F_X is a tree, then $T_X \equiv F_X$. Otherwise, we create a dummy root r and insert the roots of the trees in F_X as the children of r, in the original left-to-right order. The preorder ranks and depths of r are both 0, so that those of non-dummy nodes still start at 1. An original node $x \in X$ of T and its copy, x', in T_X

are said to *correspond* to each other; x' is also said to be the T_X-*view* of x, and x is the T-*source* of x'. The T_X-view of a node $y \in T$ (y is not required to be in X) is more generally defined to be the node $y' \in T_X$ corresponding to the lowest extracted ancestor of y, i.e. to the lowest node in $\mathcal{A}(y) \cap X$.

Representation of a Range Tree on Node Weights by Hierarchical Tree Extraction.
Range trees are widely used in solutions to query problems in Euclidean space. He et al. [15] further applied the idea of range trees to scalarly-weighted trees. They defined a conceptual range tree on node weights and represented it by a hierarchy of tree extractions. We summarize its workings when the weights are in rank space.

We first define a conceptual range tree on $[n]$ with branching factor f, where $f = \mathcal{O}(\lg^\epsilon n)$ for some constant $0 < \epsilon < 1$. Its root represents the entire range $[n]$. Starting from the root level, we keep partitioning each range, $[a, b]$, at the current lowest level into f child ranges $[a_1, b_1], \ldots, [a_f, b_f]$, where $a_i = \lceil (i-1)(b-a+1)/f \rceil + a$ and $b_i = \lceil i(b-a+1)/f \rceil + a - 1$. This ensures that, if weight $j \in [a, b]$, then j is contained in the child range with subscript $\lceil f(j-a+1)/(b-a+1) \rceil$, which can be determined in $\mathcal{O}(1)$ time. We stop partitioning a range when its size is 1. This range tree has $h = \lceil \log_f n \rceil + 1$ levels. The root is at level 1 and the bottom level is level h.

For $1 \leq l < h$, we construct an auxiliary tree T_l for level l of this range tree as follows: Let $[a_1, b_1], \ldots, [a_m, b_m]$ be the ranges at level l. For a range $[a, b]$, let $F_{a,b}$ stand for the extracted forest of the nodes of T with weights in $[a, b]$. Then, for each range $[a_i, b_i]$, we extract F_{a_i, b_i} and plug its roots as children of a dummy root r_l, retaining the original left-to-right order of the roots within the forest. Between forests, the roots in $F_{a_{i+1}, b_{i+1}}$ are the right siblings of the roots in F_{a_i, b_i}, for any $i \in [m-1]$. We then label the nodes of T_l using the reduced alphabet $[f]$, as follows. Note that barring the dummy root r_l, there is a bijection between the nodes of T and those of T_l. Let $x_l \in T_l$ be the node corresponding to $x \in T$. In the range tree, let $[a, b]$ be the level-l range containing the weight of x. Then, at level $l+1$, if the weight of x is contained in the jth child range of $[a, b]$, then $x_l \in T_l$ is labeled j. Each T_l is represented by Lemma 1 in $n(\lg f + 2) + o(n)$ bits, so the total space cost of all the T_l's is $n \lg n + (2n + o(n)) \log_f n$ bits. When $f = \omega(1)$, this space cost is $n + o(n)$ words. This completes the outline of hierarchical tree extraction. Henceforth, we shorthand as $\mathscr{T}_v$ the extraction from T of the nodes with weights in v's range, for a node v of the range tree. The following lemma maps x_l to x_{l+1}:

▶ **Lemma 2** ([15]). *Given a node $x_l \in T_l$ and the range $[a, b]$ of level l containing the weight of x, node $x_{l+1} \in T_{l+1}$ can be located in $\mathcal{O}(1)$ time, for any $l \in [h-2]$.*

Later, Chan et al. [4] augmented this representation with *ball-inheritance data structure* to map an arbitrary x_l back to x:

▶ **Lemma 3** ([4]). *Given a node $x_l \in T_l$, where $1 \leq l < h$, the node $x \in T$ that corresponds to x_l can be found using $\mathcal{O}(n \lg n \cdot \mathtt{s}(n))$ bits of additional space and $\mathcal{O}(\mathtt{t}(n))$ time, where (a) $\mathtt{s}(n) = \mathcal{O}(1)$ and $\mathtt{t}(n) = \mathcal{O}(\lg^\epsilon n)$; or (b) $\mathtt{s}(n) = \mathcal{O}(\lg \lg n)$ and $\mathtt{t}(n) = \mathcal{O}(\lg \lg n)$; or (c) $\mathtt{s}(n) = \mathcal{O}(\lg^\epsilon n)$ and $\mathtt{t}(n) = \mathcal{O}(1)$.*

Path Minimum in (Scalarly-)Weighted Trees.   In a weighted tree, path minimum query asks for the node with the smallest weight in the given path. We summarize the best result on path minimum; in it, $\alpha(m, n)$ and $\alpha(n)$ are the inverse-Ackermann functions:

▶ **Lemma 4** ([4]). *An ordinal tree T on n weighted nodes can be indexed (a) using $\mathcal{O}(m)$ bits of space to support path minimum queries in $\mathcal{O}(\alpha(m, n))$ time and $\mathcal{O}(\alpha(m, n))$ accesses to the weights of nodes, for any integer $m \geq n$; or (b) using $2n + o(n)$ bits of space to support path minimum queries in $\mathcal{O}(\alpha(n))$ time and $\mathcal{O}(\alpha(n))$ accesses to the weights of nodes. In particular, when $m = \Theta(n \lg^{**} n)$,[5] one has $\alpha(m, n) = \mathcal{O}(1)$, and therefore (a) includes the result that T can be indexed in $\mathcal{O}(n \lg^{**} n)$ bits of space to support path minimum queries in $\mathcal{O}(1)$ time and $\mathcal{O}(1)$ accesses to the weights of nodes.*

3 Reducing to Lower Dimensions

This section presents a general framework for reducing the problem of answering a d-dimensional query to the same query problem in $(d - 1)$ dimensions, by generalizing the standard technique of range tree decomposition for the case of tree topologies weighted with multidimensional vectors. To describe this framework, we introduce a *d-dimensional semigroup path sum query problem* which is a generalization of all the query problems we consider in this paper. Let $(G, \oplus)$ be a semigroup and T a tree on n nodes, in which each node x is assigned a d-dimensional weight vector $\mathbf{w}(x)$ and a semigroup element $g(x)$, with the semigroup sum operator denoted as $\oplus$. Then, in a d-dimensional semigroup path sum query, we are given a path $P_{x,y}$ in T, an orthogonal query range Q in d-dimensional space, and we are asked to compute $\sum_{z \in P_{x,y} \text{ and } \mathbf{w}(z) \in Q} g(z)$. When the weight vectors of the nodes and the query range are both from a (d', d, ϵ)-dimensional space, the (d', d, ϵ)-*dimensional semigroup path sum query problem* is defined analogously.

The following lemma presents our framework for solving a *d-dimensional semigroup path sum query problem*; its counterpart in (d', d, ϵ)-dimensional space is given in Section 4.

▶ **Lemma 5.** *Let d be a positive integer constant. Let $G^{(d-1)}$ be an $\mathbf{s}(n)$-word data structure for a $(d-1)$-dimensional semigroup path sum problem of size n. Then, there is an $\mathcal{O}(\mathbf{s}(n) \lg n + n)$-word data structure $G^{(d)}$ for a d-dimensional semigroup path sum problem of size n, whose components include $\mathcal{O}(\lg n)$ structures of type $G^{(d-1)}$, each of which is constructed over a tree on $n + 1$ nodes. Furthermore, $G^{(d)}$ can answer a d-dimensional semigroup path sum query by performing $\mathcal{O}(\lg n)$ $(d-1)$-dimensional queries using these components and returning the semigroup sum of the answers. Determining which queries to perform on structures of type $G^{(d-1)}$ requires $\mathcal{O}(1)$ time per query.* [6]

Proof. We define a conceptual range tree R with branching factor 2 over the dth weights of the nodes of T and represent it using hierarchical tree extraction as in Section 2. For each level l of the range tree, we define a tree T_l^* with the same topology as T_l. We assign $(d-1)$-dimensional weight vectors and semigroup elements to each node, x', in T_l^* as follows. If x' is not the dummy root, then $\mathbf{w}(x')$ is set to be $(w_1(x), \ldots, w_{d-1}(x))$, where x is the node of T corresponding to x'. We also set $g(x') = g(x)$. If x' is the dummy root, then its first $(d-1)$ weights are $-\infty$, while $g(x')$ is set to an arbitrary element of the semigroup. We then construct a data structure, G_l, of type $G^{(d-1)}$, over T_l^*. The data structure $G^{(d)}$ thus

---

[5] $\lg^{**} n$ stands for the number of times an iterated logarithm function $\lg^*$ needs to be applied to n in order for the result to become at most 1.

[6] It may be tempting to simplify the statement of the lemma by defining $\mathbf{t}(n)$ as the query time of $G^{(d-1)}$ and claiming that $G^{(d)}$ can answer a query in $\mathcal{O}(\mathbf{t}(n) \lg n)$ time. However, this bound is too loose when applying this lemma to reporting queries.

comprises the structures T_l and G_l, over all l. The range tree has $\mathcal{O}(\lg n)$ levels, each T_l^* has $n+1$ nodes, and the G_ls are the $\mathcal{O}(\lg n)$ structures of type $G^{(d-1)}$ referred to in the statement. As all the structures T_l occupy $n + o(n)$ words, $G^{(d)}$ occupies $\mathcal{O}(\mathsf{s}(n) \lg n + n)$ words.

Next we show how to use $G^{(d)}$ to answer queries. Let $P_{x,y}$ be the query path and $Q = \prod_{j=1}^{d} [q_j, q_j']$ be the query range. To answer the query, we first decompose $P_{x,y}$ into $A_{x,z}$, $\{z\}$, and $A_{y,z}$, where z is the lowest common ancestor of x and y, found in $\mathcal{O}(1)$ time via LCA in T_1. It suffices to answer three path semigroup sum queries using each subpath and Q as query parameters, as the semigroup sum of the answers to these queries is the answer to the original query. Since the query on subpath $\{z\}$ reduces to checking whether $\mathbf{w}(z) \in Q$, we show how to answer the query on $A_{x,z}$; the query on $A_{y,z}$ is then handled similarly. To answer the query on $A_{x,z}$, we perform a standard top-down traversal in the range tree to identify up to two nodes at each level representing ranges that contain exactly one of q_d or q_d'. Let, thus, v be the node that we are visiting, in the range tree R. We maintain *current nodes*, x_v and z_v (initialized as respectively x and z) local to the current level l; they are the nodes in T_l that correspond to $\mathcal{T}_v$-view of the original query nodes x and z. Nodes x_v and z_v are kept up-to-date in $\mathcal{O}(1)$ time as we descend the levels of the range tree. Namely, when descending to the jth ($j \in \{0,1\}$) child of the node v, we identify, via Lemma 2, the corresponding nodes in T_{l+1}, for nodes $\mathtt{level\_anc}_j(T_l, x_v, 1)$ and $\mathtt{level\_anc}_j(T_l, z_v, 1)$.

For each node v identified at each level l, such that v's range contains q_d but not q_d', we check if it is its left child-range that contains q_d. If so, we perform a $(d-1)$-dimensional semigroup range sum query with the following parameters: (i) the query range $[q_1, q_1'] \times [q_2, q_2'] \times \ldots \times [q_{d-1}, q_{d-1}']$ (i.e. we drop the last range); and (ii) the query path is A_{x_u, z_u}, where x_u and z_u are the nodes in T_{l+1} corresponding to the $\mathcal{T}_u$-views of x and z, with u being the right child of v; this is analogous to updating x_v and z_v, i.e. applying Lemma 2 to nodes $\mathtt{level\_anc}_1(T_l, x_v, 1)$, $\mathtt{level\_anc}_1(T_l, z_v, 1)$. For each node whose range contains q_d' but not q_d, a symmetrical procedure is performed by considering its left child.

The semigroup sum of the answers to these $\mathcal{O}(\lg n)$ queries is the answer to the original query. ◀

4 Space Reduction Lemma for Non-Constant Branching Factor

This section presents a general framework for reducing the problem of answering a (d', d, ϵ)-dimensional query to the same query problem in $(d'-1, d, \epsilon)$ dimensions, by generalizing the approach of [16] for the case of trees weighted with multidimensional vectors.

▶ **Lemma 6.** *Let d and d' be positive integer constants such that $d' \le d$, and ϵ be a constant in $(0,1)$. Let $G^{(d'-1)}$ be an $s(n)$-word data structure for a $(d'-1, d, \epsilon)$-dimensional semigroup path sum problem of size n. Then, there is an $\mathcal{O}(s(n) \lg n / \lg \lg n + n)$-word data structure $G^{(d')}$ for a (d', d, ϵ)-dimensional semigroup path sum problem of size n, whose components include $\mathcal{O}(\lg n / \lg \lg n)$ structures of type $G^{(d'-1)}$, each of which is constructed over a tree on $n+1$ nodes. Furthermore, $G^{(d')}$ can answer a (d', d, ϵ)-dimensional semigroup path sum query by performing $\mathcal{O}(\lg n / \lg \lg n)$ $(d'-1, d, \epsilon)$-dimensional queries using these components and returning the semigroup sum of the answers. Determining which queries to perform on structures of type $G^{(d'-1)}$ requires $\mathcal{O}(1)$ time per query.*

Proof. We define a conceptual range tree over the d'th weights of the nodes of T and represent it using hierarchical tree extraction as in Section 2. For each level l of the range tree, we define a tree T_l^* with the same topology as T_l. We assign $(d'-1, d, \epsilon)$-dimensional weight vectors and semigroup elements to each node, x', in T_l^*, as follows. If x' is not the dummy

root, then $\mathbf{w}(x')$ is set to be $(w_1(x), \ldots, w_{d'-1}(x), \lambda(T_l, x'), w_{d'+1}(x), \ldots, w_d(x))$, where x is the corresponding node of x' in T, and $\lambda(T_l, x')$ is the label assigned to x' in T_l. We also set $g(x') = g(x)$. If x' is the dummy root, then its first $d' - 1$ weights are $-\infty$ and last $d - d' + 1$ weights are $-\lceil \lg^\epsilon \rceil$, while $g(x')$ is set to an arbitrary element of the semigroup. We further construct a data structure, G_l, of type $G^{(d'-1)}$, over T_l^*. The data structure $G^{(d')}$ then comprises the structures T_l and G_l, over all l. The range tree has $\mathcal{O}(\lg n / \lg \lg n)$ levels and each T_l^* has $n + 1$ nodes, and the structures G_l are the $\mathcal{O}(\lg n / \lg \lg n)$ structures of type $G^{(d'-1)}$ referred to in the statement. As all the T_ls occupy $n + o(n)$ words, $G^{(d')}$ occupies $\mathcal{O}(s(n) \lg n / \lg \lg n + n)$ words.

Next we show how to use $G^{(d')}$ to answer queries. Let $P_{x,y}$ be the query path and $Q = \prod_{j=1}^{d} [q_j, q_j']$ be the query range. As discussed in the proof of Lemma 5, it suffices to describe the handling of the path $A_{x,z}$, where z is the lowest common ancestor of x and y.

To answer the query on $A_{x,z}$, we perform a top-down traversal in the range tree to identify the up to two nodes at each level representing ranges that contain at least one of $q_{d'}$ and $q_{d'}'$. For each node v identified at each level l, we perform a $(d' - 1, d, \epsilon)$-dimensional semigroup range sum query with parameters computed as follows: (i) the query path is P_{x_v, z_v}, where x_v and z_v are the nodes in T_l corresponding to the $\mathscr{T}_v$-views of x and z; and (ii) the query range is $Q_v = [q_1, q_1'] \times [q_2, q_2'] \times \ldots \times [q_{d'-1}, q_{d'-1}'] \times [i_v..j_v] \times [q_{d'+1}, q_{d'+1}'] \times \ldots \times [q_d, q_d']$, such that the children of v representing ranges that are entirely within $[q_{d'}, q_{d'}']$ are children $i_v, i_v + 1, \ldots, j_v$ (child i refers to the ith child); no queries are performed if such children do not exist. The semigroup sum of these $\mathcal{O}(\lg n / \lg \lg n)$ queries is the answer to the original query. It remains to show that the parameters of each query are computed in $\mathcal{O}(1)$ time per query. By Section 2, i_v and j_v are computed in $\mathcal{O}(1)$ time via simple arithmetic, which is sufficient to determine Q_v. Nodes x_v and z_v are computed in $\mathcal{O}(1)$ time each time we descend down a level in the range tree: Initially, when v is the root of the range tree, x_v and z_v are nodes x and z in T_1. When we visit a child, v_j, of v whose range contains at least one of $q_{d'}$ and $q_{d'}'$, we compute (via Lemma 2) x_{v_j} as the node in T_{l+1} corresponding to the node `level_anc`$_j(T_l, x_v, 1)$ in T_l, which uses constant time. Node z_{v_j} is located similarly.   ◀

5   Ancestor Dominance Reporting

In Lemma 7 we solve the $(1, d, \epsilon)$-dimensional *path dominance reporting problem*, which asks one to enumerate the nodes in the query path whose weight vectors dominate the query vector. The strategy employed in Lemma 7 is that of zooming into the extraction dominating the query point in the last $(d - 1)$ weights, and therein reporting the relevant nodes based on the 1st weight and tree topology only.

▶ **Lemma 7.** *Let $d \geq 1$ be a constant integer and $0 < \epsilon < \frac{1}{d-1}$ be a constant number. A tree T on $m \leq n$ nodes, in which each node is assigned a $(1, d, \epsilon)$-dimensional weight vector, can be represented in $m + o(m)$ words, so that a path dominance reporting query can be answered in $\mathcal{O}(1 + k)$ time, where k is the number of the nodes reported.*

Proof. We represent T using Lemma 1. For any $(0, d - 1, \epsilon)$-dimensional vector $\mathbf{g} = (g_1, g_2, \ldots, g_{d-1})$, we consider a conceptual scalarly-weighted tree $E_{\mathbf{g}}$ by first extracting the node set $G = \{x \mid x \in T \text{ and } \mathbf{w}_{2,d}(x) \succeq \mathbf{g}\}$ from T. The weight of a non-dummy node in $E_{\mathbf{g}}$ is the 1st weight of its T-source. If $E_{\mathbf{g}}$ has a dummy root, then its weight is $-\infty$.

Instead of storing $E_{\mathbf{g}}$ explicitly, we create the following structures, the first two of which are built for any possible $(0, d - 1, \epsilon)$-dimensional vector $\mathbf{g}$:

- A 0/1-labeled tree $T_{\mathbf{g}}$ (using Lemma 1) with the topology of T, in which a node u has label 1 *iff* u is extracted when constructing $E_{\mathbf{g}}$;
- A succinct index $I_{\mathbf{g}}$ for path maximum queries in $E_{\mathbf{g}}$ (using Lemma 4(a));
- An array W_1 where $W_1[x]$ stores the 1st weight of the node x in T;
- A table C which stores pointers to $T_{\mathbf{g}}$ and $I_{\mathbf{g}}$ for each possible $\mathbf{g}$.

For any node x' in $E_{\mathbf{g}}$, its T-source x can be computed using $x = \mathtt{pre\_select}_1(T_{\mathbf{g}}, x')$. Then, the weight of x' is $W_1[x]$. With this $\mathcal{O}(1)$-time access to node weights in $E_{\mathbf{g}}$, by Lemma 4 we can use $I_{\mathbf{g}}$ to answer path maximum queries in $E_{\mathbf{g}}$ in $\mathcal{O}(1)$ time.

We now show how to answer a path dominance reporting query in T. Let $P_{x,y}$ and $\mathbf{q} = (q_1, q_2, \ldots, q_d)$ be respectively the path and weight vector given as query parameters. First, we use C to locate $T_{\mathbf{q}'}$ and $I_{\mathbf{q}'}$, where $\mathbf{q}' = \mathbf{q}_{2,d}$. As discussed in the proof of Lemma 5, it suffices to show how to answer the query with $A_{x,z}$ as the query path, where $z = \mathtt{LCA}(T, x, y)$. To that end, we fetch the $T_{\mathbf{q}'}$-view, x', of x, as $x' = \mathtt{pre\_rank}_1(T_{\mathbf{q}'}, \mathtt{level\_anc}_1(T_{\mathbf{q}'}, x, 1))$, and analogously the view, z', of z. Next, $I_{\mathbf{q}'}$ locates a node $t' \in A_{x',z'}$ with the maximum weight. If the weight of t' is less than q_1, then no node in $A_{x,y}$ can possibly have a weight vector dominating $\mathbf{q}$, and our algorithm is terminated without reporting any nodes. Otherwise, the T-source t of t' is located as $t = \mathtt{pre\_select}_1(T_{\mathbf{q}'}, t')$. The node $t \in T$ then claims the following two properties: (i) as $T_{\mathbf{q}'}$ contains a node corresponding to t, one has $\mathbf{w}_{2,d}(t) \succeq \mathbf{q}'$; and (ii) as $w_1(t)$ equals the weight of t', it is at least q_1. We therefore have that $\mathbf{w}(t) \succeq \mathbf{q}$ and hence report t. Afterwards, we perform the same procedure recursively on paths $A_{x',t'}$ and $A_{s',z'}$ in $E_{\mathbf{q}'}$, where $s' = \mathtt{pre\_rank}_1(T_{\mathbf{q}'}, \mathtt{level\_anc}_1(T_{\mathbf{q}'}, t, 1))$.

To analyze the running time, the key observation is that we perform path maximum queries using $I_{\mathbf{q}'}$ at most $2k + 1$ times. Since both each query itself and the operations performed to identify the query path use $\mathcal{O}(1)$ time, our algorithm runs in $\mathcal{O}(1 + k)$ time.

To analyze the space cost, we observe that W_1 occupies m words. The total number of possible $(0, d-1, \epsilon)$-dimensional vectors is $\mathcal{O}(\lg^{(d-1)\epsilon} n)$. Since each $T_{\mathbf{g}}$ uses $\mathcal{O}(m)$ bits and each $I_{\mathbf{g}}$ uses $\mathcal{O}(m \lg^{**} m)$ bits, the total space space cost of storing $T_{\mathbf{g}}$'s and $I_{\mathbf{g}}$'s for all possible $\mathbf{g}$'s is $\mathcal{O}((m + m \lg^{**} m) \lg^{(d-1)\epsilon} n) = \mathcal{O}(m \lg^{**} m \lg^{(d-1)\epsilon} n) \le \mathcal{O}(m \lg^{**} n \lg^{(d-1)\epsilon} n) = o(m \lg n)$ bits for any constant $0 < \epsilon < 1/(d-1)$, which is $o(m)$ words. Furthermore, C stores $\mathcal{O}(\lg^{(d-1)\epsilon} n)$ pointers. To save the space cost of each pointer, we concatenate the encodings of all the $T_{\mathbf{g}}$s and $I_{\mathbf{g}}$s and store them in a memory block of $o(m \lg n)$ bits. Thus, each pointer stored in C can be encoded in $\mathcal{O}(\lg(m \lg n))$ bits, and the table C thus uses $\mathcal{O}((\lg m + \lg \lg n) \log^{(d-1)\epsilon} n) = \mathcal{O}(\lg m \log^{(d-1)\epsilon} n) + \mathcal{O}(\lg \lg n \log^{(d-1)\epsilon} n) = o(\lg m \lg n) + o(\lg n) = o(\lg m \lg n)$ bits for any constant $0 < \epsilon < 1/(d-1)$, which is $o(\lg m)$ words. Finally, the encoding of T using Lemma 1 is $2m + \mathcal{O}(m)$ bits. Therefore, the total space cost is $m + \mathcal{O}(m)$ words.   ◀

We next design a solution to the 2-dimensional ancestor dominance reporting problem, by first generalizing the notion of 2-dominance in Euclidean space to weighted trees. More precisely, in a tree T in which each node is assigned a d-dimensional weight vector, we say that a node x 2-*dominates* another node y *iff* $x \in \mathcal{A}(y)$ and $w_1(x) \ge w_1(y)$. Then a node x is defined to be 2-*maximal iff* no other node in T 2-dominates x.

The following property is then immediate: Given a set, X, of 2-maximal nodes, let T_X be the corresponding extraction from T. Let the weight of a node $x' \in T_X$ be the 1st weight of its T-source x. Then, in any upward path of T_X, the node weights are strictly decreasing. In such a tree as T_X, the *weighted ancestor problem* [9] is defined. In this problem, one is given a weighted tree with monotonically decreasing node weights along any upward path. We preprocess such a tree to answer *weighted ancestor queries*, which, for any given node

x and value κ, ask for the highest ancestor of x whose weight is at least κ. Farach and Muthukrishnan [9] presented an $\mathcal{O}(n)$-word solution that answers this query in $\mathcal{O}(\lg \lg n)$ time, for an n-node tree weighted over $[n]$. With an easy reduction we can further achieve the following result:

▶ **Lemma 8.** *Let T be a tree on $m \leq n$ nodes, in which each node is assigned a weight from $[n]$. If the node weights along any upward path are strictly decreasing, then T can be represented using $\mathcal{O}(m)$ words to support weighted ancestor queries in $\mathcal{O}(\lg \lg n)$ time.*

Proof. Let W be the set of weights actually assigned to the nodes of T. We replace the weight, h, of any node x in T by the rank of h in W, which is in $[m]$. We then represent the resulting tree T' in $\mathcal{O}(m)$ words to support a weighted ancestor query in T' in $\mathcal{O}(\lg \lg m)$ time [9]. We also construct a y-fast trie [22], Y, on the elements of W; the rank of each element is also stored with this element in Y. Y uses $\mathcal{O}(m)$ space. Given a weighted ancestor query over T, we first find the rank, κ, of the query weight in W in $\mathcal{O}(\lg \lg n)$ time by performing a predecessor query in Y, and κ is further used to perform a query in T' to compute the answer.   ◀

To design our data structures, we define a conceptual range tree with branching factor $f = \lceil \lg^\epsilon n \rceil$ over the 2nd weights of the nodes in T and represent it using hierarchical tree extraction as in Section 2. Let v be a node in this range tree. In $\mathscr{T}_v$, we assign to each node the weight vector of the T-source and call the resulting weighted tree $T(v)$. We then define $M(v)$ as follows: If v is the root of the range tree, then $M(v)$ is the set of all the 2-maximal nodes in T. Otherwise, let u be the parent of v. Then a node, t, of $T(v)$ is in $M(v)$ *iff* t is 2-maximal in $T(v)$ and its corresponding node in $T(u)$ is not 2-maximal in $T(u)$. Thus, for any node x in T, there exists a unique node v in the range tree such that there is a node in $M(v)$ corresponding to x.

We further conceptually extract two trees from T_l : (i) M_l is an extraction from T_l of the node set $\{x \mid x \in T_l$ and there exists a node u at level l of the range tree, s.t. x has a corresponding node in $M(u)\}$; while (ii) N_l is an extraction from T_l of the node set $\{x \mid x \in T_l$ and $\exists$ a node v at level $l + 1$ of the range tree, s.t. x has a corresponding node $\in M(v)\}$. $T_M(v)$ is the tree formed by extracting $M(v)$ from $T(v)$, and $T_G(v)$ is the tree formed by extracting from $T(v)$ the node set $\{x \mid x \in T(v)$ and there exists a child, t of v s.t. there is a node in $M(t)$ corresponding to $x\}$. Then, for each level l, we also create the following data structures (when defining these structures, we assume that the root, r_l, of T_l corresponds to a dummy node η in T with weight vector $(-\infty, -\infty)$; the node η is omitted when determining the rank space, preorder ranks, and depths in T):

- D_l, a 1-dimensional path dominance reporting structure (using Lemma 7) over the tree obtained by assigning weight vectors to the nodes of M_l as follows: each node x' of M_l is assigned a scalar weight $w_2(x)$, where x is the node of T corresponding to x';
- E_l, a 1-dimensional path dominance reporting structure (using Lemma 7) over the tree obtained by assigning weight vectors to the nodes of M_l as follows: each node x' of M_l is assigned a scalar weight $w_1(x)$, where x is the node of T corresponding to x';
- F_l, a $(1, 2, \epsilon)$-dimensional path dominance reporting structure (using Lemma 7) over the tree obtained by assigning weight vectors to the nodes of N_l as follows: each node x' of N_l is assigned $(w_1(x), \kappa)$, where x is the node of T corresponding to x', and κ is the label assigned to the node in T_l corresponding to x';
- A_l, a weighted ancestor query structure over M_l (using Lemma 8), when its nodes are assigned the 1st weights of the corresponding nodes in T;

- T'_l, a 0/1-labeled tree (using Lemma 1) with the topology of T_l, and a node is assigned 1 *iff* it is extracted when constructing M_l;
- T''_l, a 0/1-labeled tree (using Lemma 1) with the topology of T_l, and a node is assigned 1 *iff* it is extracted when constructing N_l;
- P_l, an array where $P_l[x]$ stores the preorder number of the node in T corresponding to a node x in M_l.

We now describe the algorithm for answering queries, and analyze its running time and space cost:

▶ **Lemma 9.** *A tree T on n nodes, in which each node is assigned a 2-dimensional weight vector, can be represented in $\mathcal{O}(n)$ words, so that an ancestor dominance reporting query can be answered in $\mathcal{O}(\lg n + k)$ time, where k is the number of the nodes reported.*

Proof. Let x and $\mathbf{q} = (q_1, q_2)$ be the node and weight vector given as query parameters, respectively. We define Π as the path in the range tree between and including the root and the leaf storing q_2. Let π_l denote the node at level l in this path. Then the root of the range tree is π_1. To answer the query, we perform a traversal of a subset of the nodes of the range tree, starting from π_1. The invariant maintained during this traversal is that a node u of the range tree is visited *iff* one of the following two conditions holds: (i) $u = \pi_l$ for some l; or (ii) $M(u)$ contains at least one node whose corresponding node in T must be reported. We now describe how the algorithm works when visiting a node, v, at level l of this range tree, during which we will show how the invariant is maintained. Let x_v denote the node in T_l that corresponds to the $\mathscr{T}_v$-view of x; x_v can be located in constant time each time we descend down one level in the range tree, as described in the proof of Lemma 5. Our first step is to report all the nodes in the answer to the query that have corresponding nodes in $M(v)$. There are two cases depending on whether $v = \pi_l$; this condition can be checked in constant time by determining whether q_2 belongs to the range represented by v. In either of these cases, we first locate the M_l-view, x'_v, of x_v by computing $x'_v = \texttt{pre\_rank}_1(T'_l, \texttt{level\_anc}_1(T'_l, x_v, 1))$.

If (i) holds, then the non-dummy ancestors of x'_v in M_l correspond to all the ancestors of x in T that have corresponding nodes in $M(v)$. We then perform a weighted ancestor query using A_l to locate the highest ancestor, y, of x'_v in M_l whose 1st weight is at least q_1. Since the 1st weights of the nodes along any upward path in M_l are decreasing, the 1st weights of the nodes in path $P_{x'_v,y}$ are greater than or equal to q_1, while those of the proper ancestors of y are strictly less. Hence, by performing a 1-dimensional path dominance reporting query in D_l using $P_{x'_v,y}$ as the query path and $\mathbf{q}' = (q_2)$ as the query weight vector, we can find all the ancestors of x'_v whose corresponding nodes in T have weight vectors dominating $\mathbf{q}$. Then, for each of these nodes, we retrieve from P_l its corresponding node in T which is further reported.

IF $v \neq \pi_l$, the maintained invariant guarantees that the 2nd weights of the nodes in $M(v)$ are greater than q_2. Therefore, by performing a 1-dimensional path dominance reporting query in $E_l(s)$ using the path between (inclusive) x'_v and the root of M_l as the query path and $\mathbf{q}'' = (q_1)$ as the query weight vector, we can find all the ancestors of x'_v in M_l whose corresponding nodes in T have weight vectors dominating $\mathbf{q}$. By mapping these nodes to nodes in T via P_l, we have reported all the nodes in the answer to the query that have corresponding nodes in $M(v)$.

After we handle both cases, the next task is to decide which children of v we should visit. Let v_i denote the ith child of v. We always visit π_{l+1} if it happens to be a child of v. To maintain the invariant, for any other child v_i, we visit it *iff* there exists at least one node in $M(v_i)$

whose corresponding node in T should be reported. To find the children that we will visit, we locate the N_l-view, x_v'', of x_v by computing $x_v'' = \mathtt{pre\_rank}_1(T_l'', \mathtt{level\_anc}_1(T_l'', x_v, 1))$. Then the non-dummy ancestors of x_v'' correspond to all the ancestors of x in T that have corresponding nodes in $\cup_{i=1,2,\ldots} M(v_i)$. We then perform a $(1, 2, \epsilon)$-dimensional path dominance reporting query in F_l using the path between (inclusive) x_v'' and the root of N_l as the query path and $(q_1, \kappa + 1)$ as the query weight vector if π_{l+1} is the κth child of v, and we set $\kappa = 0$ if π_{i+1} is not a child of v. For each node, t, returned when answering this query, if its 2nd weight in F_l is j, then t corresponds to a node in $M(v_j)$. Since the node corresponding to t in T should be included in the answer to the original query, we iteratively visit v_j if we have not visited it before (checked e.g. using an f-bit word to flag the children of v).

The total query time is dominated by the time used to perform queries using A_l, D_l, E_l and F_l. We only perform one weighted ancestor query when visiting each π_l, and this query is not performed when visiting other nodes of the range tree. Given the $\mathcal{O}(\lg n / \lg \lg n)$ levels of the range tree, all the weighted ancestor queries collectively use $\mathcal{O}(\lg \lg n \times (\lg n / \lg \lg n)) = \mathcal{O}(\lg n)$ time. Similarly, we perform one query using D_l at each level of the range tree, and the query times summed over all levels is $\mathcal{O}(\lg n / \lg \lg n + k)$. Our algorithm guarantees that, each time we perform a query using E_l, we report a not-reported hitherto, non-empty subset of the nodes in the answer to the original query. Therefore, the queries performed over all E_l's use $\mathcal{O}(k)$ time in total. Querying the F_l-structures incurs $\mathcal{O}(k)$ time cost when visiting nodes not in Π, and $\mathcal{O}(\lg n / \lg \lg n + k)$ time when visiting nodes in Π. Thus, the query times spent on all these structures throughout the execution of the algorithm sum up to $\mathcal{O}(\lg n + k)$.

We next analyze space cost of our data structures. As mentioned in Section 2, all the T_ls occupy $n + o(n)$ words. By Lemma 1, each T_l' or T_l'' uses $3n + o(n)$ bits, so over all $\lg n / \lg \lg n$ levels, they occupy $\mathcal{O}(n \lg n / \lg \lg n)$ bits, which is $\mathcal{O}(n / \lg \lg n)$ words. As discussed earlier, we know that, for any node x in T, there exists one and only one node v in the range tree such that there is a node in $M(v)$ corresponding to x. Furthermore, $M(v)$s only contain nodes that have corresponding nodes in T. Therefore, the sum of the sizes of all $M(v)$s is exactly n. Hence all the P_l's have n entries in total and thus uses n words. By Lemma 7, the size of each D_l in words is linear in the number of nodes in M_l. The sum of the numbers of nodes in M_ls over all levels of the range tree is equal to the sum of the sizes of all $M(v)$s plus the number of dummy roots, which is $n + \mathcal{O}(\lg n / \lg \lg n)$. Therefore, all the D_ls occupy $\mathcal{O}(n)$ words. By similar reasoning, all the E_ls and A_ls occupy $\mathcal{O}(n)$ words in total. Finally, it is also true that, for any node x in T, there exists a unique node v in the range tree such that there is a node in $N(v)$ corresponding to x. Thus, we can upper-bound the total space cost of all the F_ls by $\mathcal{O}(n)$ words in a similar way. All our data structures, therefore, use $\mathcal{O}(n)$ words.   ◀

Further, we describe the data structure for $(2, d, \epsilon)$-dimensional ancestor dominance reporting, and analyze its time- and space-bounds:

▶ **Lemma 10.** *Let $d \geq 2$ be a constant integer and $0 < \epsilon < \frac{1}{d-2}$ be a constant number. A tree T on n nodes, in which each node is assigned a $(2, d, \epsilon)$-dimensional weight vector, can be represented in $\mathcal{O}(n \lg^{(d-2)\epsilon} n)$ words, so that an ancestor dominance reporting query can be answered in $\mathcal{O}(\lg n + k)$ time, where k is the number of the nodes reported.*

Proof. In our design, for any $(0, d - 2, \epsilon)$-dimensional vector **g**, we consider a conceptual scalarly-weighted tree $E_\mathbf{g}$ as the tree extraction from T of the node set $\{x \mid x \in T \text{ and } \mathbf{w}_{3,d}(x) \succeq \mathbf{g}\}$. The weight of a node x' in $E_\mathbf{g}$ is the 2-dimensional weight vector $\mathbf{w}_{1,2}(x)$, where x the T-source of x'. If $E_\mathbf{g}$ has a dummy root, then its weight is $(-\infty, -\infty)$. Rather than storing $E_\mathbf{g}$ explicitly, we follow the strategy in the proof of Lemma 7 and store

a 0/1-labeled tree $T_{\mathbf{g}}$ for each possible $\mathbf{g}$. $T_{\mathbf{g}}$ is obtained from T by assigning 1-labels to the nodes of T extracted when constructing $E_{\mathbf{g}}$. We also maintain arrays W_1 and W_2 storing respectively the 1st and 2nd weights of all nodes of T, in preorder, which enables accessing the weight of an arbitrary node of $E_{\mathbf{g}}$ in $\mathcal{O}(1)$ time. Let $n_{\mathbf{g}}$ be the number of nodes in $E_{\mathbf{g}}$. We convert the node weights of each $E_{\mathbf{g}}$ to rank space $[n_{\mathbf{g}}]$. For each such $E_{\mathbf{g}}$, we build the 2-dimensional ancestor dominance reporting data structure, $V_{\mathbf{g}}$, from Lemma 9. Thus, the space usage of the resulting data structure is upper-bounded by $\mathcal{O}(n \lg^{(d-2)\epsilon} n)$ words.

Let x and $\mathbf{q} = (q_1, q_2, \ldots, q_d)$ be the node and weight vector given as query parameters, respectively. In $\mathcal{O}(1)$ time, we fetch the data structures pertaining to the range $\mathbf{q'} = \mathbf{q}_{3,d}$; this way, all the weights 3 through d of the query vector have been taken care of, and all we need to consider is the tree topology and the first two weights, q_1 and q_2, of the original query vector. We localize the query node x to $E_{\mathbf{q'}}$ via $x' = \mathtt{pre\_rank}_1(T_{\mathbf{q'}}, \mathtt{level\_anc}_1(T_{\mathbf{q'}}, x, 1))$, and launch the query in $V_{\mathbf{q'}}$ with x' as a query node, having reduced the components of the query vector (q_1, q_2) to the rank space of $E_{\mathbf{q'}}$ (the time- and space-bounds for the reductions are absorbed in the final bounds). ◄

Instantiating Section 3 with $g(x) = \{x\}$ and the semigroup sum operator $\oplus$ as the set-theoretic union operator $\cup$, Lemma 5 iteratively applied to Lemma 9 yields

▶ **Theorem 11.** *Let $d \geq 2$ be a constant integer. A tree T on n nodes, in which each node is assigned a d-dimensional weight vector, can be represented in $\mathcal{O}(n \lg^{d-2} n)$ words, so that an ancestor dominance reporting query can be answered in $\mathcal{O}(\lg^{d-1} n + k)$ time, where k is the number of the nodes reported.*

Analogously, Lemma 6 (Section 4) that is the counterpart of Lemma 5 when the range tree has a non-constant branching factor $f = \mathcal{O}(\lg^\epsilon n)$, with Lemma 10 which addresses $(2, d, \epsilon)$-dimensional ancestor reporting, together yield a different tradeoff:

▶ **Theorem 12.** *Let $d \geq 3$ be a constant integer. A tree T on n nodes, in which each node is assigned a d-dimensional weight vector, can be represented in $\mathcal{O}(n \lg^{d-2+\epsilon} n)$ words of space, so that an ancestor dominance reporting query can be answered in $\mathcal{O}((\lg^{d-1} n)/(\lg \lg n)^{d-2}+k)$ time, where k is the number of the nodes reported. Here, $\epsilon \in (0, 1)$ is a constant.*

6 Path Successor

We first solve the path successor problem when $d = 1$, and extend the result to $d > 1$ via Lemma 5.

The topology of T is stored using Lemma 1. We define a binary range tree R over $[n]$, and build the associated hierarchical tree extraction as in Section 2; T_l denotes the auxiliary tree built for each level l of R, and $\mathscr{T}_v$ denotes the tree extraction from T associated with the range of node $v \in R$. We represent R using Lemma 1, and augment it with the ball-inheritance data structure $\mathcal{B}$ from Lemma 3(a), as well as with the data structure from the following

▶ **Lemma 13.** *Let R be a binary range tree with topology encoded using Lemma 1, and augmented with ball-inheritance data structure $\mathcal{B}$ from Lemma 3(a). With additional space of $\mathcal{O}(n)$ words, the node $x_{u,l}$ in T_l corresponding to the $\mathscr{T}_u$-view of x can be found in $\mathcal{O}(\log^{\epsilon'} n)$ time, where ϵ' is an arbitrary constant in $(0, 1)$, for an arbitrary node $x \in T$ and an arbitrary node $u \in R$ residing on a level l.*

Proof. For a node $u \in R$ at a level l, and a node $x \in T$, the query can be thought of as a chain of transformations $T \to \mathscr{T}_u \to T_l$. In the first transition, $T \to \mathscr{T}_u$, given an original node $x \in T$, we are looking for its $\mathscr{T}_u$-view, x_u. That is, although $\mathscr{T}_u$ is obtained from T through a *series* of extractions, the wish is to "jump" many successive extractions at once, as if $\mathscr{T}_u$ were extracted from T *directly*. This would be trivial to achieve through storing a 0/1-labeled tree per range u, if it were not for prohibitive space-cost – number of bits more than quadratic in the number of nodes. One can avoid extra space cost altogether and use Lemma 2 directly to explicitly descend the hierarchy of extractions. In this case, the time cost is proportional to the height of the range tree, and hence becomes the bottleneck.

In turn, in the $\mathscr{T}_u \to T_l$-transition, we are looking for the identity of x_u in T_l. For this second transformation, we recall (from Section 2) that $\mathscr{T}_u$ is embedded within T_l. Moreover, the nodes of $\mathscr{T}_u$ must lie contiguously in the preorder sequence of T_l.

We overcome these difficulties with the following data structures.

For R, we maintain an annotation array I, such that $I[u]$ stores a quadruple $\langle a_u, b_u, s_u, t_u \rangle$ for an arbitrary node $u \in R$, such that (i) the weight range associated with u is $[a_u, b_u]$; and (ii) all the nodes of T with weights in $[a_u, b_u]$ occupy precisely the preorder ranks s_u through t_u in T_l. The space occupied by the annotation array I, which is $\mathcal{O}(\lg n)$ bits summed over all the $\mathcal{O}(n)$ nodes of R, is $\mathcal{O}(n)$ words.

For each level $L \equiv 0 \bmod \lceil \lg \lg n \rceil$, which we call *marked*, we maintain a data structure enabling the direct $T \to \mathscr{T}_u \to T_L$-conversion. Namely, for each individual node u on marked level L of R, we define a conceptual array A_u, which stores, in increasing order, the (original) preorder ranks of all the nodes of T whose weights are in the range represented by u. Rather than maintaining A_u explicitly, we store a succinct index, S_u, for predecessor/successor search [11] in A_u. Assuming the availability of a $o(n^\delta)$-bit universal table, where δ is a constant in $(0, 1)$, given an arbitrary value in $[n]$, this index can return the position of its predecessor/successor in A_u in $\mathcal{O}(\lg \lg n)$ time plus accesses to $\mathcal{O}(1)$ entries of A_u. The size of the index in bits is $\mathcal{O}(\lg \lg n)$ times the number of entries in A_u. For a fixed marked level L, therefore, all the S_u-structures sum up to $\mathcal{O}(n \lg \lg n)$ bits. There being $\mathcal{O}(\lg n / \lg \lg n)$ marked levels, the total space cost for the S_u-structures over all the entire tree R is $\mathcal{O}(n)$ words.

We now turn to answering the query using the data structures built. Resolving the query falls into two distinct cases, depending on whether the level l, at which the query node u resides, is marked or not.

When the level l is marked, we use the structures S_u stored therein, directly. We adopt the strategy in [14] to find $x_{u,l}$. First, for an arbitrary index i to A_u, we observe that node $A_u[i] \in T$ corresponds to node $(s_u + i - 1)$ in T_l. We thus fetch $\langle a_u, b_u, s_u, t_u \rangle$ from $I[u]$. Then the predecessor $p \in A_u$ of x is obtained through S_u via an $\mathcal{O}(\lg \lg n)$ query and $\mathcal{O}(1)$ calls to the $\mathcal{B}$-structure, which totals $\mathcal{O}(\lg^{\epsilon'} n)$ time. We then determine the lowest common ancestor $\chi \in T$ of x and p, in $\mathcal{O}(1)$ time. If the weight of χ is in $[a_u, b_u]$, then it must be present in A_u by the latter's very definition. By another predecessor query, therefore, we can find the position, j, of χ in A_u, and $(s_u + j - 1)$ is the sought $x_{u,l}$. Otherwise, a final query to S_u returns the successor χ' in A_u of χ. Let κ be the position of χ' in A_u. Then the parent of the node $(s_u + \kappa - 1)$ in T_l is $x_{u,l}$. We perform a constant number of predecessor/successor queries, and correspondingly a constant number of calls to the ball-inheritance problem. The time complexity is thus $\mathcal{O}(\lg^{\epsilon'} n)$.

When the level l is not marked, we ascend to the lowest ancestor u' of u residing on a marked level l', and reduce the problem to the previous case. More precisely, via the navigation operations (`level_anc()` to move to a parent, and `depth()` to determine the

status of a level) available through R's encoding, we climb up at most $\lceil \lg \lg n \rceil$ levels to the closest marked level l'. Let u' be therefore the ancestor of u found on that marked level l'. We find the answer to the original query, as if the query node were u'; that is, we find the node $x'_{u',l'}$ in $T_{l'}$ that corresponds to the $\mathscr{T}_{u'}$-view of the original query node x in T. Let us initialize a variable χ to be $x_{u',l'}$. We descend down to the original level l, back to the original query node u, all the while adjusting the node χ as we move down a level, analogously to the proof of Lemma 5. As we arrive, in time $\mathcal{O}(\lg \lg n)$, at node u, the variable χ stores the answer, $x_{u,l}$.

In both cases, the term $\mathcal{O}(\lg^{\epsilon'} n)$ dominates the time complexity, as climbing to/from a marked level is an additive term of $\mathcal{O}(\lg \lg n)$. Therefore, a query is answered in $\mathcal{O}(\lg^{\epsilon'} n)$ time.   ◀

Finally, each T_l is augmented with succinct indices m_l (resp. M_l) from Lemma 4(b), for path minimum (resp. path maximum) queries. As weights of the nodes of T_l, the weights of their corresponding nodes in T are used.

We now describe the algorithm for answering queries and analyze its running time, as well as give the space cost of the built data structures:

▶ **Lemma 14.** *A scalarly-weighted tree T on n nodes can be represented in $\mathcal{O}(n)$ words, so that a path successor query is answered in $\mathcal{O}(\lg^\epsilon n)$ time, where $\epsilon \in (0, 1)$ is a constant.*

Proof. Let x, y and $Q = [q_1, q_1']$ be respectively the nodes and the orthogonal range given as query's parameters. Appealing to the proof of Lemma 5, we focus only on the path $A_{x,z}$, where z is $\mathtt{LCA}(T, x, y)$. We locate in $\mathcal{O}(1)$ time the leaf L_{q_1} of R that corresponds to the singleton range $[q_1, q_1]$. Let Π be the root-to-leaf path to L_{q_1} in R; let π_l be the node at level l of Π. We binary search in Π for the deepest node $\pi_f \in \Pi$ whose associated extraction $\mathscr{T}_{\pi_f}$ contains the node corresponding to the answer to the given query.

We initialize two variables: *high* as 1 so that π_{high} is the root of R, and *low* as the height of R so that π_{low} is the leaf L_{q_1}. We first check if $\mathscr{T}_{\pi_{low}}$ already contains the answer, by fetching the node x' in T_{low} corresponding to the $\mathscr{T}_{\pi_{low}}$-view of x, using Lemma 13. If x' exists, we examine its corresponding node x'' in T (fetched via $\mathcal{B}$) to see whether x'' is on $A_{x,z}$, by performing $\mathtt{depth}$ and $\mathtt{level\_anc}$ operations in R; if it is, x'' is the final answer. If not, this establishes the invariant of the ensuing search: $\mathscr{T}_{\pi_{high}}$ contains a node corresponding to the answer, whereas $\mathscr{T}_{\pi_{low}}$ does not.

At each iteration, therefore, we set (via $\mathtt{level\_anc}$ in R) π_{mid} to be the node mid-way from π_{low} to π_{high}. We then fetch the nodes x', z' in T_{mid} corresponding to the $\mathscr{T}_{\pi_{mid}}$-views of respectively x and z, using Lemma 13. The non-existence of x' or the emptiness of $A_{x',z'}$ sets *low* to *mid*, and the next iteration of the search ensues. If z' does not exist, z' is set to the root of T_{mid}. A query to the M_{mid}-structure then locates a node in $A_{x',z'}$ for which the 1st weight, μ, of its corresponding node in T is maximized. Accounting for the mapping of a node in T_{mid} to its corresponding node in T via $\mathcal{B}$, this query uses $\mathcal{O}((\lg^{\epsilon'} n)\alpha(n))$ time. The variables are then updated as *high* $\leftarrow$ *mid* if $\mu \geq q_1$, and *low* $\leftarrow$ *mid*, otherwise.

Once π_f is located, it must hold for π_f that (i) it is its left child that is on Π [20]; and (ii) its right child, v, contains the query result, even though v represents a range of values all larger than q_1. When locating π_f, we also found the nodes in T_f corresponding to the $\mathscr{T}_{\pi_f}$-views of x and z; they can be further used to find the nodes in T_{f+1}, x^* and z^*, corresponding to the $\mathscr{T}_v$-views of x and z. We then use m_{f+1} to find the node in A_{x^*,z^*} with minimum 1st weight, whose corresponding node in T is the answer.

The total query time is dominated by that needed for binary search. Each iteration of the search is in turn dominated by the path maximum query in T_{mid}, which is $\mathcal{O}((\lg^{\epsilon'} n)\alpha(n))$. Given the $\mathcal{O}(\lg n)$ levels of R, the binary search has $\mathcal{O}(\lg \lg n)$ iterations. Therefore, the total running time is $\mathcal{O}(\lg \lg n \cdot \lg^{\epsilon'} n \cdot \alpha(n)) = \mathcal{O}(\lg^{\epsilon} n)$ if we choose $\epsilon' < \epsilon$.

To analyze the space cost, we observe that the topology of T, represented using Lemma 1, uses only $2n + o(n)$ bits. As mentioned in Section 2, all the structures T_l occupy $\mathcal{O}(n)$ words. The space cost of the structure from Lemma 13 built for R is $\mathcal{O}(n)$ words. The $\mathcal{B}$-structure occupies another $\mathcal{O}(n)$ words. The m_l- and M_l-structure occupy $\mathcal{O}(n)$ bits each, or $\mathcal{O}(n)$ words in total over all levels of R. Thus, the final space cost is $\mathcal{O}(n)$ words.   ◄

Lemmas 5 and 14 yield the following

▶ **Theorem 15.** *Let $d \geq 1$ be a constant integer. A tree T on n nodes, in which each node is assigned a d-dimensional weight vector can be represented in $\mathcal{O}(n \lg^{d-1} n)$ words, so that a path successor query can be answered in $\mathcal{O}(\lg^{d-1+\epsilon} n)$ time, for an arbitrarily small positive constant ϵ.*

Proof. We instantiate Section 3 with $g(x) = x$ and the semigroup sum operator $\oplus$ as $x \oplus y = \mathtt{argmin}_{\zeta=x,y}\{w_1(\zeta)\}$. Lemma 5 applied to Lemma 14 yields the space bound of $\mathcal{O}(n \lg^{d-1} n)$ words and query time complexity of $\mathcal{O}(\lg^{d-1+\epsilon} n)$.   ◄

───── **References** ─────

1   Peyman Afshani. On Dominance Reporting in 3D. In *ESA*, pages 41–51, 2008.

2   Noga Alon and Baruch Schieber. Optimal preprocessing for answering on-line product queries. Technical report, Tel-Aviv University, 1987.

3   Timothy M. Chan. Persistent Predecessor Search and Orthogonal Point Location on the Word RAM. In *SODA*, pages 1131–1145, 2011.

4   Timothy M. Chan, Meng He, J. Ian Munro, and Gelin Zhou. Succinct Indices for Path Minimum, with Applications. *Algorithmica*, 78(2):453–491, 2017.

5   Timothy M. Chan, Kasper Green Larsen, and Mihai Patrascu. Orthogonal range searching on the RAM, revisited. In *SoCG*, pages 1–10, 2011.

6   Bernard Chazelle. Computing on a free tree via complexity-preserving mappings. *Algorithmica*, 2(1):337–361, 1987.

7   Bernard Chazelle and Herbert Edelsbrunner. Linear Space Data Structures for Two Types of Range Search. *Discrete & Computational Geometry*, 2:113–126, 1987.

8   Erik D. Demaine, Gad M. Landau, and Oren Weimann. On Cartesian Trees and Range Minimum Queries. *Algorithmica*, 68(3):610–625, 2014.

9   Martin Farach and S. Muthukrishnan. Perfect Hashing for Strings: Formalization and Algorithms. In *CPM*, pages 130–140, 1996.

10 Harold N. Gabow, Jon Louis Bentley, and Robert E. Tarjan. Scaling and Related Techniques for Geometry Problems. In *STOC*, pages 135–143, 1984.

11 Roberto Grossi, Alessio Orlandi, Rajeev Raman, and S. Srinivasa Rao. More Haste, Less Waste: Lowering the Redundancy in Fully Indexable Dictionaries. In *STACS*, pages 517–528, 2009.

12 Torben Hagerup. Parallel Preprocessing for Path Queries Without Concurrent Reading. *Inf. Comput.*, 158(1):18–28, 2000.

13 Meng He, J. Ian Munro, and Srinivasa Rao Satti. Succinct ordinal trees based on tree covering. *ACM Trans. Algorithms*, 8(4):42:1–42:32, 2012.

14 Meng He, J. Ian Munro, and Gelin Zhou. A Framework for Succinct Labeled Ordinal Trees over Large Alphabets. *Algorithmica*, 70(4):696–717, 2014.

15   Meng He, J. Ian Munro, and Gelin Zhou. Data Structures for Path Queries. *ACM Trans. Algorithms*, 12(4):53:1–53:32, 2016.

16   Joseph JáJá, Christian Worm Mortensen, and Qingmin Shi. Space-Efficient and Fast Algorithms for Multidimensional Dominance Reporting and Counting. In *ISAAC*, pages 558–568, 2004.

17   Danny Krizanc, Pat Morin, and Michiel H. M. Smid. Range Mode and Range Median Queries on Lists and Trees. *Nord. J. Comput.*, 12(1):1–17, 2005.

18   Christos Makris and Athanasios K. Tsakalidis. Algorithms for Three-Dimensional Dominance Searching in Linear Space. *Inf. Process. Lett.*, 66(6):277–283, 1998.

19   Yakov Nekrich. A data structure for multi-dim. range reporting. In *SoCG*, pages 344–353, 2007.

20   Yakov Nekrich and Gonzalo Navarro. Sorted Range Reporting. In *SWAT*, pages 271–282, 2012.

21   Manish Patil, Rahul Shah, and Sharma V. Thankachan. Succinct representations of weighted trees supporting path queries. *J. Discrete Algorithms*, 17:103–108, 2012.

22   Dan E. Willard. Log-Logarithmic Worst-Case Range Queries are Possible in Space Theta(N). *Inf. Process. Lett.*, 17(2):81–84, 1983.

23   Gelin Zhou. Two-dimensional range successor in optimal time and almost linear space. *Inf. Process. Lett.*, 116(2):171–174, 2016.