


# Path and Ancestor Queries over Trees with Multidimensional Weight Vectors

Meng He ✉

Faculty of Computer Science, Dalhousie University, Halifax, NS, Canada

Serikzhan Kazi ✉ 

Faculty of Computer Science, Dalhousie University, Halifax, NS, Canada

---

## Abstract

We consider an ordinal tree  $T$  on  $n$  nodes, such that each node is assigned a  $d$ -dimensional weight vector  $\mathbf{w} \in \{1, 2, \dots, n\}^d$ , where  $d \in \mathbb{N}$  is a constant. We study path queries as generalizations of the 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 generalizes the corresponding  $(d + 1)$ -dimensional orthogonal range query.

Our study yields the following new results. We solve the 2D *ancestor dominance reporting* problem as a direct generalization of 3D dominance reporting problem, in time  $\mathcal{O}(\lg n + k)$  and space of  $\mathcal{O}(n)$  words, where  $k$  is the size of the output. We solve the *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 the *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 the *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 trade-offs of the respective range queries. We are also the first to solve the path successor problem even for  $d = 1$ .

**Keywords and phrases** range queries, path queries, ancestor dominance reporting, path counting, path reporting, path successor

**Digital Object Identifier** 10.57717/cgt.v4i1.39

**Related Version** A preliminary partial version of this article was published in the Proceedings of the 30<sup>th</sup> International Symposium on Algorithms and Computation (ISAAC 2019) [16].

**Funding** *Meng He*: This work was supported by Natural Sciences and Engineering Research Council (NSERC) of Canada

*Serikzhan Kazi*: This work was supported by Natural Sciences and Engineering Research Council (NSERC) of Canada

## 1 Introduction

The problem of pre-processing 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 studied extensively [2, 6, 15, 21, 9, 19, 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  $\mathbf{w} = (w_1, w_2, \dots, w_d)$  to be a vector with  $d$  components, each in rank space  $[n]$ ,<sup>1</sup> i.e.  $\mathbf{w} \in [n]^d$ , with  $w_i$  being referred to as the  $i^{\text{th}}$  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 are interested in present us with a  $d$ -dimensional orthogonal range  $Q = \prod_{i=1}^d [q_i, q'_i]$ . A weight vector  $\mathbf{w}$  is in  $Q$  iff  $\forall i \in [1, d]$  it holds that  $q_i \leq w_i \leq q'_i$ . Precisely, in our queries 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 pre-process 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  $\arg \min\{w_1(z) \mid z \in P_{x,y} \text{ and } \mathbf{w}(z) \in Q\}$ .<sup>2</sup>
- *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  $\mathbf{q} = (q_1, q_2, \dots, q_d)$ .

This is indeed a natural generalization of the traditional weighted tree, which we refer to as 1D-weighted (or simply as *weighted*, when context is clear), to the case in which 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 and 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 of the 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 [12]. We additionally assume  $d$ , the number of dimensions, is a positive integer constant.

## 1.1 Previous work

### 1.1.1 Path queries in weighted trees

For 1D-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;<sup>3</sup> it works primarily with topology of the tree and is thus oblivious to the distribution of weights. Later, He et al. [19] proposed a solution with  $nH(W_T) + \mathcal{O}(n \lg \sigma)$  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  $T$ .

He et al. [19] introduced and solved the path reporting problem using (i) linear space and  $\mathcal{O}((1+k) \lg \sigma)$  query time, or (ii)  $\mathcal{O}(n \lg \lg \sigma)$  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. [27] presented a succinct data structure for path reporting with  $n \lg \sigma + 6n + \mathcal{O}(n \lg \sigma)$  bits of space and  $\mathcal{O}((\lg n + k) \lg \sigma)$  query time. Although the latter solution uses less space than the version (i) of the former when  $\sigma \ll n$ , it suffers a logarithmic slowdown in the additive term. An optimal-space solution with  $nH(W_T) + \mathcal{O}(n \lg \sigma)$  bits of space and  $\mathcal{O}((1+k)(\frac{\lg \sigma}{\lg \log n} + 1))$  reporting time is due to He et al. [19]. One of the trade-offs proposed by Chan et al. [4] requires  $\mathcal{O}(n \lg^\epsilon n)$  words of space for the query time of  $\mathcal{O}(\lg \lg n + k)$ .

<sup>1</sup> Throughout this article,  $[n]$  stands for  $\{1, 2, \dots, n\}$  for any positive integer  $n$ .

<sup>2</sup> 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  $q'_1$ .

<sup>3</sup>  $\lg x$  denotes  $\log_2 x$  in this paper.

### 1.1.2 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  $\mathcal{O}(\lg^2 n + k)$  query time, in the pointer-machine (PM) model, with the latter being improved to  $\mathcal{O}(\lg n \lg \lg n + k)$  by Makris and Tsakalidis [22]. The same authors [22] developed, in the word-RAM, a linear-size,  $\mathcal{O}(\lg n + k)$  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 [23] 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  $\mathcal{O}(\lg \lg n + k)$  query time was designed for 3D dominance reporting in rank space [1, 3]. In the PM model, Afshani [1] also presented an  $\mathcal{O}(\lg n + k)$  query time, linear-space data structure for points in  $\mathbb{R}^3$ . For the dominance reporting problem in 4D, Nekrich [24] presented a data structure of size  $\mathcal{O}(n \lg^\epsilon n)$  and the query time of  $\mathcal{O}(\lg n / \lg \lg n + k)$ .

For the word-RAM model, JáJá et al. [20] generalized the range counting problem for  $d \geq 2$  dimensions and proposed a data structure of  $\mathcal{O}(n(\frac{\lg n}{\lg \lg n})^{d-2})$  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. For 4D orthogonal range reporting, Nekrich [25] presented a data structure using  $\mathcal{O}(n \lg^{2+\epsilon} n)$  words of space, for  $\mathcal{O}(\lg n / \lg \lg n + k)$  query time; the full version [24] of the paper extends this result to  $d \geq 4$ , obtaining a data structure with  $\mathcal{O}(n \lg^{d-2+\epsilon} n)$  words of space, for the query time of  $\mathcal{O}((\frac{\lg n}{\lg \lg n})^{d-3} + k)$ , for the  $d$ -dimensional orthogonal range reporting problem.

Nekrich and Navarro [26] proposed two trade-offs for the range successor problem, 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 [30] later improved upon the query time of the second trade-off 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 trade-offs on geometric queries in Euclidean space. We present solutions for the:

- 2D ancestor dominance reporting problem, in  $\mathcal{O}(n)$  words of space and the query time of  $\mathcal{O}(\lg n + k)$ . This matches the space bound for 3D dominance reporting of Afshani [1] and Chan [3], while still providing efficient query support.
- path successor problem, in  $\mathcal{O}(n \lg^{d-1} n)$  words and  $\mathcal{O}(\lg^{d-1+\epsilon} n)$  query time, for an arbitrarily small positive constant  $\epsilon$  and  $d \geq 1$ . These bounds match the first trade-off for range successor of Nekrich and Navarro [26].<sup>4</sup> Previously this problem has not been studied even on 1D-weighted trees;
- path counting problem, in  $\mathcal{O}(n(\frac{\lg n}{\lg \lg n})^{d-1})$  words and  $\mathcal{O}((\frac{\lg n}{\lg \lg n})^d)$  query time for  $d \geq 1$ . This matches the best bound for range counting in  $d+1$  dimensions [20];
- path reporting problem, for:

<sup>4</sup> Range successor can be generalized to higher dimensions via standard techniques based on range trees.

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

$d \geq 3$  in  $\mathcal{O}(n \lg^{d-1+\epsilon} n)$  words and  $\mathcal{O}(\frac{\lg^{d-1} n}{(\lg \lg n)^{d-2}} + k)$  query time. Our result matches both in space and query time the result that can be obtained by combining known pre-processing techniques [28] with the latest result from Nekrich [25]. The latter result, however, was published after the preliminary version of the present article and is more complex.

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 tree decompositions from general point-sets to tree topologies (Lemma 6). The notion of *maximality* in Euclidean sense is central to solutions of orthogonal dominance problems. We employ a few novel techniques and extend this notion to tree topologies and provide the means of efficient computation thereof (Section 4). Furthermore, 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 15). Given the ubiquity of the concepts, these technical contributions are likely to be of independent interest.

### 1.3 Comparison

In this section, we compare our results, presented in Section 1.2, with those that could be achieved immediately by well-known techniques such as *heavy-path decomposition* [28].

Namely, consider pre-processing the topology of the given tree  $T$  using heavy-path decomposition (henceforth HPD, for short). HPD roots the tree at an arbitrary node and computes the size of the subtree rooted at each node. The tree of size  $n$  is then laid out in an array  $L$  of length  $n$  via a depth-first search (dfs) launched from the root. The dfs is special in the sense that upon processing the children of a current node  $x$ , one first descends to the child of  $x$  with the largest subtree-size among all the children of  $x$  (henceforth called *heaviest child*). Informally, during the dfs, one maintains a *current chain*, which is extended by one while descending to the heaviest child; one initiates a new chain when descending to other children. The array  $L$  is therefore a preorder enumeration of the nodes of  $T$ , where a node in each position is either the heaviest child of its parent, or the head of a chain of its own.

Thus, HPD gives rise to a set of *chains*, each of which can be considered a  $d + 1$ -dimensional set of points. By the property of HPD, a path query can be split into  $\mathcal{O}(\lg n)$  queries on  $d + 1$ -dimensional regions. In more detail, we construct query data structures over each chain. When answering a query, we query  $\mathcal{O}(\lg n)$  of these chains, and combine the results of these  $\mathcal{O}(\lg n)$  queries.

Table 1, therefore, is obtained as follows. Each type of query we consider in this work occupies a row of its own, three columns each. The leftmost column represents the query time (resp. space occupancy) of the data structure we present in this article, while the second column stands for the query time (resp. space occupancy) of a data structure based on HPD. Finally, the third column represents the best known bounds for the corresponding problems in Euclidean space.

When calculating space bounds, the bounds for the HPD-based approach equal those of the best known bounds for  $d + 1$ -dimensional orthogonal range queries. When calculating

query time, we also multiply such a bound (path counting and path successor), or the additive term thereof (ancestor dominance and path reporting), by  $\mathcal{O}(\lg n)$ .

For example, the 2D version of the range successor problem has a trade-off of  $\mathcal{O}(n)$  space and  $\mathcal{O}(\lg^\epsilon n)$  time [26]. This implies an  $\mathcal{O}(n \lg^{d-1} n)$ -word structure with  $\mathcal{O}(\lg^{d-1+\epsilon} n)$  query time in  $\mathbb{R}^{d+1}$ , where  $d \geq 1$ , by the standard reduction via range trees. We therefore extend this approach to an HPD-processed tree  $T$  with  $\mathcal{O}(n \lg^{d-1} n)$  words of space, thereby answering  $d$ -dimensional path successor queries in  $\mathcal{O}(\lg^{d+\epsilon} n)$  time. The query time degrades by a factor of  $\mathcal{O}(\lg n)$  compared to the solution for points in  $\mathbb{R}^{d+1}$ , because we account for the logarithmic slowdown of HPD.

For the ancestor dominance problem, we list in Table 1 the result for  $d = 2$  only, for the following reasons. The 4D dominance reporting structure of Nekrich [24], which was published after the preliminary version of this article [16] and is complex, can be generalized to even higher dimensions using range trees of non-constant branching factors [20] and further combined with HPD to produce a data structure of size  $\mathcal{O}(n \lg^{\epsilon+d-3} n)$  and  $\mathcal{O}(\frac{\lg^{d-1} n}{(\lg \lg n)^{d-2}} + k)$  query time for  $d$ -dimensional ancestor dominance reporting. This result holds for  $d \geq 3$ . For this case, we can provide two data structures. Both of them, however, would be inferior to the combination of HPD with [24].

Indeed, our approach for  $d = 2$ , when extended for arbitrary  $d \geq 3$  via Lemma 6, achieves the space complexity of  $\mathcal{O}(n \lg^{d-2} n)$  words and the query time of  $\mathcal{O}(\lg^{d-1} n + k)$ . In terms of space complexity, thus, this is poly-logarithmically worse. For query time, our approach suffers a factor of poly-loglog degradation.

For dimensions higher than 2, we can also arrive, via Lemma 7 (similar to our approach of generalizing path reporting to higher dimensions), at a different data structure, of  $\mathcal{O}(n \lg^{d-2+\epsilon} n)$  words and having  $\mathcal{O}(\frac{\lg^{d-1} n}{(\lg \lg n)^{d-2}} + k)$  query time. That is, our data structure would still suffer an additional  $\lg n$  factor in space complexity, although its query time would be the same as that of HPD with [24].

Thus, we do not include this case so that Table 1 lists new results only.

## 2 Preliminaries

### 2.1 Notation

Given a  $d$ -dimensional weight vector  $\mathbf{w} = (w_1, w_2, \dots, w_d)$ , we define vector  $\mathbf{w}_{i,j}$  to be  $(w_i, w_{i+1}, \dots, w_j)$ . We extend the definition to a range  $Q = \prod_{k=1}^d [q_k, q'_k]$  by setting  $Q_{i,j} \triangleq \prod_{k=i}^j [q_k, q'_k]$ . (Henceforth  $\triangleq$  stands for the phrase “defined as”.) We use the symbol  $\succeq$  for domination:  $\mathbf{p} \succeq \mathbf{q}$  iff  $\mathbf{p}$  dominates  $\mathbf{q}$ , i.e. each of the weights of  $\mathbf{p}$  is greater than or equal to the corresponding weight of  $\mathbf{q}$ ; a point  $\mathbf{p}$   $k$ -dominates  $\mathbf{q}$  iff  $\mathbf{p}_{1,k} \succeq \mathbf{q}_{1,k}$ . 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 set  $i/0 \triangleq \infty$  for  $i > 0$ .

During a preorder traversal of a given tree  $T$ , the  $i^{\text{th}}$  node visited is said to have *preorder rank*  $i$ . (When context is clear, the expression  $x \in T$  stands for “the node  $x$  of the tree  $T$ ”.) 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. the 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  $A_{x,y} \triangleq P_{x,y} \setminus \{y\}$ .

			Our approach	HPD+ $\mathbb{R}^{d+1}$	$\mathbb{R}^{d+1}$	
Ancestor	$d=2$	Time	$\mathcal{O}(\lg n + k)$	$\mathcal{O}(\lg \lg n \lg n + k)$	$\mathcal{O}(\lg \lg n + k)$	[1, 3]
		Space	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	
Counting	$d \geq 1$	Time	$\mathcal{O}((\frac{\lg n}{\lg \lg n})^d)$	$\mathcal{O}(\frac{\lg^{d+1} n}{(\lg \lg n)^d})$	$\mathcal{O}((\frac{\lg n}{\lg \lg n})^d)$	[20]
		Space	$\mathcal{O}(n(\frac{\lg n}{\lg \lg n})^{d-1})$	$\mathcal{O}(n(\frac{\lg n}{\lg \lg n})^{d-1})$	$\mathcal{O}(n(\frac{\lg n}{\lg \lg n})^{d-1})$	
Successor	$d \geq 1$	Time	$\mathcal{O}(\lg^{d-1+\epsilon} n)$	$\mathcal{O}(\lg^{d+\epsilon} n)$	$\mathcal{O}(\lg^{d-1+\epsilon} n)$	[26]
		Space	$\mathcal{O}(n \lg^{d-1} n)$	$\mathcal{O}(n \lg^{d-1} n)$	$\mathcal{O}(n \lg^{d-1} n)$	
Reporting	$d=2$	Time	$\mathcal{O}(\lg n + k)$	$\mathcal{O}(\lg \lg n \lg n + k)$	$\mathcal{O}(\lg \lg n + k)$	[5]
		Space	$\mathcal{O}(n \lg^{1+\epsilon} n)$	$\mathcal{O}(n \lg^{1+\epsilon} n)$	$\mathcal{O}(n \lg^{1+\epsilon} n)$	
	$d \geq 3$	Time	$\mathcal{O}(\frac{\lg^{d-1} n}{(\lg \lg n)^{d-2}} + k)$	$\mathcal{O}(\frac{\lg^{d-1} n}{(\lg \lg n)^{d-2}} + k)$	$\mathcal{O}((\frac{\lg n}{\lg \lg n})^{d-2} + k)$	[25]
		Space	$\mathcal{O}(n \lg^{d-1+\epsilon} n)$	$\mathcal{O}(n \lg^{d-1+\epsilon} n)$	$\mathcal{O}(n \lg^{d-1+\epsilon} n)$	

**Table 1** Summary comparison of our results versus solutions based on heavy-path decomposition (HPD), as well as the Euclidean  $\mathbb{R}^{d+1}$  case. We state the results for the path reporting problem in the case of  $d \geq 3$  separately, because both results – our solution and HPD+ $\mathbb{R}^{d+1}$  – match, but our solution is simpler

## 2.2 Succinct representations of ordinal trees

Succinct representations of unlabeled and labeled ordinal trees is a widely researched area. The following lemma presents the previous results on unlabeled trees that will be used in our solutions.

► **Lemma 1** ([17]). *An ordinal tree  $T$  on  $n$  nodes can be represented in  $2n + \mathcal{O}(n)$  bits of space to support the following operations, for any nodes  $x, y \in T$ , in  $\mathcal{O}(1)$  time:*

- $\text{child}(T, x, i)$  the  $i^{\text{th}}$  child of  $x$ ;
- $\text{depth}(T, x)$  the number of ancestors of  $x$ ;
- $\text{level\_anc}(T, x, i)$  the  $i^{\text{th}}$  closest ancestor of  $x$  (with  $\text{level\_anc}(T, x, 1)$  being  $x$  itself);
- $\text{LCA}(T, x, y)$  the lowest common ancestor of  $x$  and  $y$ .

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  $\alpha$  is called an  $\alpha$ -node (resp.  $\alpha$ -ancestor):

► **Lemma 2** ([18]). *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  $n(\lg \sigma + 2) + \mathcal{O}(n)$  bits of space to support the following operations, for any node  $x \in T$ , in  $\mathcal{O}(1)$  time:*

- $\text{pre\_rank}_\alpha(T, x)$  the number of  $\alpha$ -nodes that precede  $x$  in preorder;

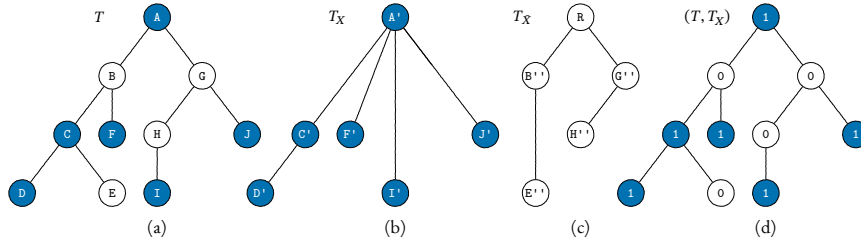


- $\text{pre\_select}_\alpha(T, i)$  the  $i^{\text{th}}$   $\alpha$ -node in preorder; and
- $\text{level\_anc}_\alpha(T, x, i)$  the  $i^{\text{th}}$  closest  $\alpha$ -ancestor of  $x$ .

Here, one has  $\text{level\_anc}_\alpha(T, x, 1) = x$  if  $x$  is an  $\alpha$ -node itself.

### 2.3 Tree extraction

*Tree extraction* [19] 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  $v_1, v_2, \dots, v_k$  (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  $\mathcal{A}(y) \cap X$ . (See Figure 1 for an example of tree extraction.)



■ **Figure 1** Tree extraction. Original tree (a), extracted tree  $T_X$  (b), extraction of the complement of  $X$ , tree  $T_{\bar{X}}$  (c) and the indicator tree  $(T, T_X)$  (d). The blue shaded nodes in  $T$  form the set  $X$ . In the tree  $T_X$ , node  $C'$  corresponds to node  $C$  in the original tree  $T$ , and node  $C'$  in the extracted tree  $T_X$  is the  $T_X$ -view of nodes  $C$  and  $E$  in the original tree  $T$ . Finally, node  $C$  in  $T$  is the  $T$ -source of the node  $C'$  in  $T_X$ . Extraction of the complement,  $T_{\bar{X}}$ , demonstrates the case of adding a dummy root  $R$ .

A common scenario of using tree extraction in our solutions is captured in the following

► **Definition 1.** For a given tree  $T$  and an extraction  $T_X$  therefrom, let  $T'$  be a tree with the topology of  $T$  and in which a node is labeled with 1 if it has been extracted into  $T_X$ , or with 0 otherwise. Then  $T'$  is referred to as the indicator tree of  $(T, T_X)$ .

Figure 1 also gives an example of an indicator tree. From Lemma 2 it follows that indicator tree of  $(T, T_X)$  occupies  $3n + o(n)$  bits of space. When translating node identifiers between  $T$  and  $T_X$ , the following fact can be readily seen:

► **Proposition 1.** Let  $T'$  be the indicator tree of  $(T, T_X)$ . Then, (i) the corresponding node  $x \in T$  of a node  $x^* \in T_X$  can be recovered as

$$x = \text{pre\_select}_1(T', x^*);$$

and (ii) the  $T_X$ -view  $x^*$  of a node  $x \in T$  can be computed as

$$x^* = 1 + \text{pre\_rank}_1(T', \text{level\_anc}_1(T', x, 1)).$$

## 2.4 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. [19] further applied the idea of range trees to 1D-weighted trees. They defined a conceptual range tree on node weights and represented it by a hierarchy of tree extractions. Let us summarize how the hierarchy is built.

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  $[a_1, b_1], \dots, [a_f, b_f]$ , 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, i.e. when  $b-a+1 = 1$ . This range tree has  $h = \lceil \log_f n \rceil + 1$  levels. The root is at level 1 and the bottom level is level  $h$ . We next describe how to construct each of the trees  $T_l$ , for each level  $l$  of the range tree.

### Construction of auxiliary trees $T_l$

The construction of  $T_l$  has two major aspects: (a) the topology of  $T_l$ ; and (b) the labels of the nodes in  $T_l$ . In the next paragraphs, we go through these steps, in order.

**Topology of  $T_l$**  For  $1 \leq l < h$ , the topology of the auxiliary tree  $T_l$  for level  $l$  is determined as follows. The root of  $T_l$  is a dummy node  $r_l$ . To define the rest of the tree structure, let  $[a_1, b_1], \dots, [a_m, b_m]$  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 the dummy root  $r_l$ , retaining the original left-to-right order of the roots within the forest. Thus, 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  $i \in [m-1]$ .

**Labels of the nodes in  $T_l$**  We then label the nodes of  $T_l$  using the reduced alphabet  $[f]$ , as follows. Recall 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  $j^{\text{th}}$  child range of  $[a, b]$ , then  $x_l \in T_l$  is labeled  $j$ .

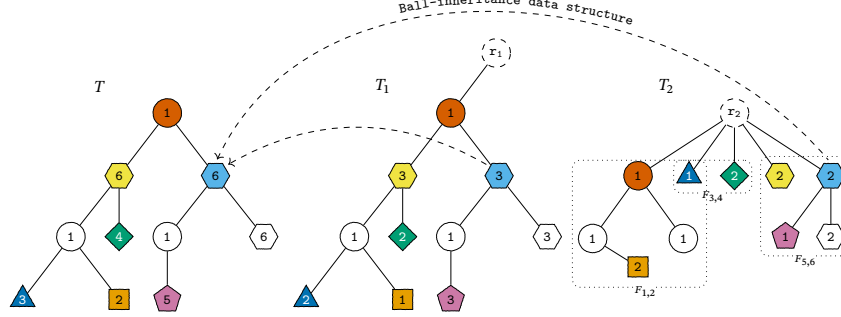
**Example** Figure 2 shows an example of hierarchical tree extraction. In  $T_1$  the nodes labeled with 3 correspond to the nodes in  $T$  with original weights 5 and 6, because when the range  $[6]$  is split into  $f = 3$  ranges  $[1, 2] \cup [3, 4] \cup [5, 6]$ , numbers 5 and 6 fall into the third range  $[5, 6]$ . Likewise, the nodes labeled 1 in  $T_1$  are precisely those that were originally labeled as 1 or 2.

Each  $T_l$  is represented by Lemma 2 in  $n(\lg f + 2) + \mathcal{O}(n)$  bits, so the total space cost of all the  $T_l$ s is  $n \lg n + (2n + \mathcal{O}(n)) \log_f n$  bits. When  $f = \omega(1)$ , this space cost is  $n + \mathcal{O}(n)$  words. This completes the description of hierarchical tree extraction.

The following lemma maps  $x_l$  to  $x_{l+1}$ :

► **Lemma 3** ([19]). *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  $l \in [h-2]$ .*





■ **Figure 2** Hierarchical tree extraction with branching factor  $f = 3$ , for a tree  $T$  weighted over an alphabet of size 6. The labels (for  $T_l$ ) and weights (for  $T$ ) are given within the nodes. For some nodes, the correspondence across the levels of the hierarchy is shown using same non-plain colours and node shapes. Rounded rectangular areas enclose the nodes of the corresponding forests. Dashed arrow maps a node in  $T_l$  to the original node, and illustrate the *ball-inheritance* data structure of Lemma 4

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

- **Lemma 4** ([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  $\mathcal{O}(n \lg n \cdot s(n))$  bits of additional space and  $\mathcal{O}(t(n))$  time, where*
- (a)  $s(n) = \mathcal{O}(1)$  and  $t(n) = \mathcal{O}(\lg^\epsilon n)$ ;
  - (b)  $s(n) = \mathcal{O}(\lg \lg n)$  and  $t(n) = \mathcal{O}(\lg \lg n)$ ; or
  - (c)  $s(n) = \mathcal{O}(\lg^\epsilon n)$  and  $t(n) = \mathcal{O}(1)$ .

In what follows, we denote as  $\mathcal{T}_v$  the extraction from  $T$  of the nodes with weights in  $v$ 's range, for a node  $v$  of the range tree  $\mathcal{R}$ .

## 2.5 Path minimum in 1D-weighted trees

Finally, to support ancestor dominance reporting and path successor, we need data structures supporting *path minimum queries*, which ask for the node with the smallest weight in the given path of a weighted tree. We summarize the best result on path minimum; in it,  $\alpha(m, n)$  and  $\alpha(n)$  are the inverse-Ackermann functions:

- **Lemma 5** ([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  $m = \Theta(n \lg^{**} n)$ ,<sup>5</sup> 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.*

<sup>5</sup>  $\lg^{**} n$  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.

### 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

$$\bigoplus_{z \in P_{x,y} \text{ and } \mathbf{w}(z) \in Q} g(z).$$

When the weight vectors of the nodes and the query range are both from a  $(d', d, \epsilon)$ -dimensional space, the  $(d', d, \epsilon)$ -dimensional semigroup path sum query problem is defined analogously.

#### 3.1 Space reduction lemma for branching factor 2

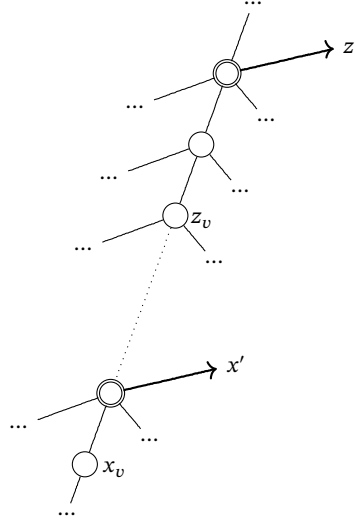
Lemma 6 presents our framework for solving a  $d$ -dimensional semigroup path sum query problem; its counterpart in  $(d', d, \epsilon)$ -dimensional space is given in Section 3.2.

The query procedure described in Lemma 6 mimics the search in a multidimensional range tree [8].

► **Lemma 6.** *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.<sup>6</sup>*

**Proof.** We define a conceptual range tree  $R$  with branching factor 2 over the  $d^{\text{th}}$  weights of the nodes of  $T$  and represent it using hierarchical tree extraction as in Section 2.4. 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  $\mathbf{w}_{1,d-1}(x)$ , 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  $G^{(d)}$  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  $G_l$ s are the  $\mathcal{O}(\lg n)$  structures of type  $G^{(d-1)}$  referred to in the statement. By the exposition of hierarchical tree extraction in Section 2.4, all the structures  $T_l$  in total occupy  $n + \mathcal{O}(n)$  words, and  $G^{(d)}$  therefore occupies  $\mathcal{O}(\mathbf{s}(n) \lg n + n)$  words.

<sup>6</sup> 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.



■ **Figure 3** An illustration of the proof of Lemma 6. Shown is a root-to-leaf path that contains both  $x_v$  and  $z_v$ . Double circles represent the nodes with weights falling into the range of  $v$ 's  $j^{\text{th}}$  child. They are retrieved via a call to  $\text{level\_anc}_j(T_l, \cdot, 1)$ . The nodes  $x'$  and  $z'$  are the nodes corresponding respectively to  $\text{level\_anc}_j(T_l, x_v, 1)$  and  $\text{level\_anc}_j(T_l, z_v, 1)$  in the next level of the hierarchy of extractions

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 analogously. To answer the query on  $A_{x,z}$ , we perform a standard top-down traversal in the range tree. Let  $v$  be the node that we are currently 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 the  $\mathcal{T}_v$ -views 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  $j^{\text{th}}$  ( $j \in \{0, 1\}$ ) child of the node  $v$ , we identify, via Lemma 3, the corresponding nodes in  $T_{l+1}$ , for the nodes  $\text{level\_anc}_j(T_l, x_v, 1)$  and  $\text{level\_anc}_j(T_l, z_v, 1)$ , as illustrated in Figure 3.

Recall (see e.g. [8]) that in a traversal over a range tree, there are five essential mutual configurations for the query range  $[q_d, q'_d]$  and the range  $[a, b]$  corresponding to the current node  $v$  of the range tree. In order for the present article to be self-sufficient, we describe each of these five cases. The first two symmetrical cases are when  $q'_d \leq c$  and when  $c + 1 \leq q_d$ , where  $c = \frac{a+b}{2}$ ; that is, the query range is completely contained within one of the subtrees of  $v$ , and the case is dealt trivially by descending into the respective child subtree, and solving the problem recursively therein. The third case,  $a < q_d \leq c < q'_d < b$ , occurs at most once during the entire traversal. It generates the remaining two symmetrical configurations: the left subtree with range  $[a, c]$  and the effective query range  $[q_d, c]$ , and the right one with the range  $[c + 1, b]$  and the effective query range  $[c + 1, q'_d]$ . It is thus sufficient to show how to handle the latter two cases.

In view of symmetry, let us consider the case when for the node  $v$  at level  $l$  of the range tree, one has  $a < q_d \leq c$  and  $q'_d = b$ . (We need not consider the case in which  $q_d \geq c + 1$  and  $d'_d = b$ , because it is subsumed in the second case described in the previous paragraph.) For the right child subtree of  $v$ , we perform a  $(d - 1)$ -dimensional semigroup range sum query with the following arguments: (i) the query range  $[q_1, q'_1] \times [q_2, q'_2] \times \dots \times [q_{d-1}, q'_{d-1}]$  (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 3 to the nodes  $\text{level\_anc}_1(T_l, x_v, 1)$  and  $\text{level\_anc}_1(T_l, z_v, 1)$ . Having performed this  $(d - 1)$ -dimensional query, we proceed to the left child of  $v$ .

In the case  $a = q_d$  and  $c + 1 \leq q'_d < b$ , a symmetrical procedure is performed by considering the left child of  $v$  for the  $(d - 1)$ -dimensional query, and descending further to the right child.

At each level  $l$  of the range tree, therefore, we perform no more than 2 queries.

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

### 3.2 Space reduction lemma for non-constant branching factor

Presented in this section is a general framework for reducing the problem of answering a  $(d', d, \epsilon)$ -dimensional query to the same query problem in  $(d' - 1, d, \epsilon)$  dimensions, by generalizing the approach of JáJá et al. [20] for the case of trees weighted with multidimensional vectors.

► **Lemma 7.** *Let  $d$  and  $d'$  be positive integer constants such that  $d' \leq d$ , and  $\epsilon$  be a constant in  $(0, 1)$ . Let  $G^{(d'-1)}$  be an  $\mathbf{s}(n)$ -word data structure for a  $(d' - 1, d, \epsilon)$ -dimensional semigroup path sum problem of size  $n$ . Then, there is an  $\mathcal{O}(\mathbf{s}(n) \lg n / \lg \lg n + n)$ -word data structure  $G^{(d')}$  for a  $(d', d, \epsilon)$ -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  $(d', d, \epsilon)$ -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 with branching factor  $f = \mathcal{O}(\lg^\epsilon n)$  over the  $d'^{\text{th}}$  weights of the nodes of  $T$  and represent it using hierarchical tree extraction as in Section 2.4. 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

$$(w_1(x), \dots, w_{d'-1}(x), \lambda(T_l, x'), w_{d'+1}(x), \dots, w_d(x)),$$

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  $-\lceil \lg^\epsilon n \rceil$ , 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. By the exposition of hierarchical tree extraction in Section 2.4, all the  $T_l$ s in total occupy  $n + \mathcal{o}(n)$  words. Therefore,  $G^{(d')}$  occupies  $\mathcal{O}(\mathbf{s}(n) \lg n / \lg \lg n + n)$  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 6, 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  $\mathcal{T}_v$ -views of  $x$  and  $z$ ; and (ii) the query range is  $Q_v = [q_1, q'_1] \times [q_2, q'_2] \times \dots \times [q_{d'-1}, q'_{d'-1}] \times [i_v, j_v] \times [q_{d'+1}, q'_{d'+1}] \times \dots \times [q_d, q'_d]$ , such that the children of  $v$  representing ranges that are entirely within  $[q_{d'}, q'_{d'}]$  are children  $i_v, i_v + 1, \dots, j_v$  (“child  $i$ ” refers to the  $i^{\text{th}}$  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 2.4,  $i_v$  and  $j_v$  are computed in  $\mathcal{O}(1)$  time via simple arithmetic, which is sufficient to construct  $Q_v$ . 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 3)  $x_{v_j}$  as the node in  $T_{l+1}$  corresponding to the node `level_ancj`( $T_l, x_v, 1$ ) in  $T_l$ , which uses constant time. Node  $z_{v_j}$  is located similarly. ◀

## 4 Ancestor dominance reporting

This section proposes a solution to the ancestor dominance reporting problem. Our solution is designed around the *layers of maxima* paradigm [7]. We consider a partial ordering of the nodes of a weighted tree, in which two nodes are in relation *iff* one is the ancestor of the other, and its weight is greater than that of the other. We then design data structures that allow iteration through a layer of *maximal* elements of this partial ordering, as well as switching from one layer to the next.

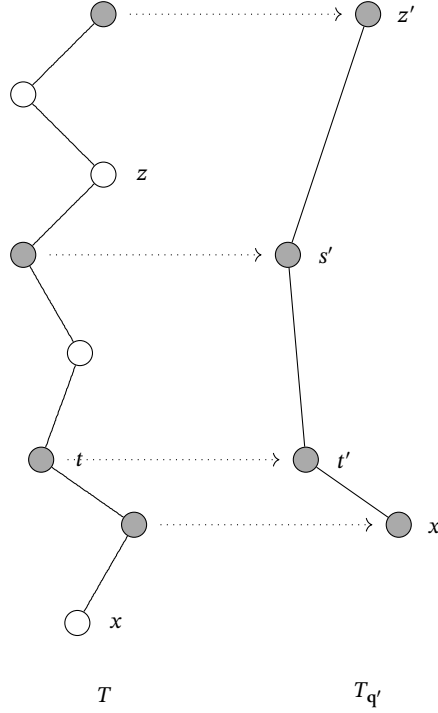
This section comprises Sections 4.1–4.4. In Section 4.1, we solve the  $(1, d, \epsilon)$ -dimensional *path dominance reporting problem*, which asks one to enumerate the nodes in the query path whose weight vectors dominate the query vector. Then, in Section 4.2, we motivate our data structures for solving the 2D ancestor dominance reporting problem using the above-mentioned concepts from its Euclidean counterpart. In Section 4.3, we design the data structures for solving the 2D ancestor dominance reporting problem, and analyze their space cost. Finally, Section 4.4 describes the query algorithm on the data structures built in Section 4.3, and presents our result for the 2D ancestor dominance reporting problem.

From this section onward, the results presented in Section 2.5 will be used.

### 4.1 Path dominance in $(1, d, \epsilon)$

The strategy employed in Lemma 8 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  $1^{\text{st}}$  weight and tree topology only.

► **Lemma 8.** *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  $m + \mathcal{O}(m)$  words, so that a path dominance reporting query can be answered in  $\mathcal{O}(1 + k)$  time, where  $k$  is the number of reported nodes.*



■ **Figure 4** Illustration to the proof of Lemma 8. A part of the tree containing  $P_{x,z}$  is shown. The shaded nodes in  $T$  represent the nodes belonging to the extraction  $T_{q'}$ . The node  $t$  is the nodes with the largest  $1^{st}$  weight among the extracted nodes in  $A_{z,x}$ . Dotted arrows show corresponding nodes

**Proof.** We represent  $T$  using Lemma 2. For any vector  $\mathbf{g} = (g_1, g_2, \dots, g_{d-1})$  in  $(0, d-1, \epsilon)$  dimensions, we consider a conceptual 1D-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  $1^{st}$  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  $\mathbf{g}$ :

- The indicator tree (Definition 1)  $T_{\mathbf{g}}$  of  $(T, E_{\mathbf{g}})$ ;
- A succinct index  $I_{\mathbf{g}}$  for path maximum queries in  $E_{\mathbf{g}}$  (using Lemma 5(a));
- An array  $W_1$  where  $W_1[x]$  stores the  $1^{st}$  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  $x$  equals  $\text{pre\_select}_1(T_{\mathbf{g}}, x')$ . 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 5 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  $\mathbf{q} = (q_1, q_2, \dots, q_d)$  be respectively the path and the 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 6, it suffices to show how to answer the query with  $A_{x,z}$  as the query path, where  $z = \text{LCA}(T, x, y)$ . To that end, we fetch the  $E_{\mathbf{q}'}$ -view,  $x'$ , of  $x$ , using Proposition 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 in Proposition 1. 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 duly report  $t$ . Afterwards, we perform the same procedure recursively on paths  $A_{x',t'}$  and  $A_{s',z'}$  in  $E_{\mathbf{q}'}$ , where  $s'$  is the parent of  $t'$  in  $E_{\mathbf{q}'}$  and can be computed as the  $E_{\mathbf{q}'}$ -view of the parent of  $t$ , using Proposition 1. See Figure 4 for an illustration.

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 cost of storing  $T_{\mathbf{g}}$ s and  $I_{\mathbf{g}}$ s for all possible vectors  $\mathbf{g}$  is  $\mathcal{O}((m + m \lg^{**} m) \lg^{(d-1)\epsilon} n) = \mathcal{O}(m \lg^{**} m \lg^{(d-1)\epsilon} n) \leq \mathcal{O}(m \lg^{**} n \lg^{(d-1)\epsilon} n) = \mathcal{O}(m \lg n)$  bits for any constant  $0 < \epsilon < 1/(d - 1)$ , or  $\mathcal{O}(m)$  words. Furthermore,  $C$  stores  $\mathcal{O}(\lg^{(d-1)\epsilon} n)$  pointers. To reduce the space cost for 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  $\mathcal{O}(m \lg n)$  bits. Thus, each pointer stored in  $C$  can be encoded in  $\mathcal{O}(\lg(m \lg n))$  bits, and the table  $C$  thus uses  $\mathcal{O}((\lg m + \lg \lg n) \lg^{(d-1)\epsilon} n) = \mathcal{O}(\lg m \lg^{(d-1)\epsilon} n) + \mathcal{O}(\lg \lg n \lg^{(d-1)\epsilon} n) = \mathcal{O}(\lg m \lg n) + \mathcal{O}(\lg n) = \mathcal{O}(\lg m \lg n)$  bits for any constant  $0 < \epsilon < 1/(d - 1)$ , which is  $\mathcal{O}(\lg m)$  words. Finally, the encoding of  $T$  using Lemma 2 is  $2m + \mathcal{O}(m)$  bits. Therefore, the total space cost is  $m + \mathcal{O}(m)$  words.  $\blacktriangleleft$

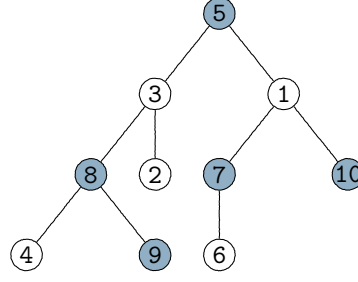
## 4.2 2-maximal nodes

In order to motivate the data structures in this section, let us consider the following what we shall call *naïve approach* for 2D dominance reporting in Euclidean space. Given a set of points, let us build a range tree over the  $y$ -axis. Each node of the range tree is associated with a *subset* of points stored at the leaves of its subtree: namely, the set of “maximal” points in the sense that no two points dominate each other. It is then clear that each point of the original point-set is accounted for – namely, it is associated with a (unique) node of the range tree. Secondly, at each node  $u$  of the range tree, it is sufficient to keep the points sorted by, say,  $y$  coordinate, so that reporting is done in  $\mathcal{O}(\lg n + k)$  time. Finally, for a query  $\mathbf{q} = (a, b)$ , during the traversal of the range tree, we need to consider the right sibling of the (range tree) node  $u$  *iff* its largest  $x$ -coordinate is at least  $a$ .

We design a solution to the 2-dimensional ancestor dominance reporting problem by largely imitating the naïve approach. First, we generalize the notion of 2-dominance in Euclidean space, to weighted trees. 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  $w_1(x) > w_1(y)$ . Then a node  $x$  is defined to be *2-maximal* *iff* no other node in  $T$  2-dominates  $x$ . An example of a 2-maximal set of nodes is given in Figure 5.

Secondly, we imitate the “summary” feature of the naïve approach, i.e. the ability to efficiently decide whether to consider the right sibling of a range-tree node. In our case, the fan-out of the range tree is not 2, and therefore we bootstrap using the data structures presented in Section 4.1. Namely, we extract the nodes that can *possibly* dominate a query tuple. The key is that a certain component of the weight vector is re-coded according to the range-tree subtree the respective tuple belongs to. Then, the extraction is pre-processed using the data structure from Section 4.1. Whenever a query to this data structure returns a node, we know precisely which siblings one needs to explore. These intuitions aid our further, more precise, discussions.





■ **Figure 5** The 2-maximal set in a tree  $T$  weighted over  $\sigma = 10$ . The numbers inside the circles represent the assigned weights. The 2-maximal set is shown in shaded circles. The shaded nodes in any upward path form a decreasing sequence

The following property of the maximal nodes is 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 1<sup>st</sup> weight of its  $T$ -source  $x$ . Then, in any upward path of  $T_X$ , the node weights are strictly decreasing, and therefore the *weighted ancestor problem* [10] is defined. In this problem, one is given a weighted tree with monotonically decreasing node weights along any upward path. We pre-process such a tree to answer *weighted ancestor queries*, which, for any given node  $x$  and value  $\kappa$ , ask for the highest ancestor of  $x$  whose weight is at least  $\kappa$ . Farach and Muthukrishnan [10] 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 9.** *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 [10]. We also construct a  $y$ -fast trie [29],  $Y$ , 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,  $\kappa$ , of the query weight in  $W$  in  $\mathcal{O}(\lg \lg n)$  time by performing a predecessor query in  $Y$ , and  $\kappa$  is further used to perform a query in  $T'$  to compute the answer. ◀

### 4.3 Data structures for 2D ancestor dominance reporting

We tackle the 2-dimensional ancestor dominance problem with the following data structures. We define a conceptual range tree  $R$  with branching factor  $f = \lceil \lg^\epsilon n \rceil$  over the  $2^{nd}$  weights of the nodes in  $T$  and represent  $T$  using hierarchical tree extraction as in Section 2.4. Let  $v$  be a node in this range tree  $R$ , and let  $R_l$  denote the level  $l$  of  $R$ . In  $\mathcal{T}_v$ , 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  $x$ . Furthermore, for a non-leaf node  $v \in R$  we define the set  $N(v)$  as the set  $\{t \in T \mid \exists \text{ a child } v' \text{ of } v \text{ such that } t \in M(v')\}$ .

We further conceptually extract two trees from  $T_l$  : (i)  $M_l$  is an extraction from  $T_l$  of the node set

$$\{x \mid x \in T_l \text{ and } \exists \text{ a node } u \in R_l \text{ s.t. } x \text{ has a corresponding node in } M(u)\};$$

while (ii)  $N_l$  is an extraction from  $T_l$  of the node set

$$\{x \mid x \in T_l \text{ and } \exists \text{ a node } v \in R_l \text{ s.t. } x \text{ has a corresponding node in } N(v)\}.$$

Figure 6 provides an example of the trees  $M_l$  and  $N_l$ .

Furthermore, for each level  $l$ , we also create the following sets of data structures (when defining these structures, we assume that the root,  $r_l$ , of  $T_l$  corresponds to a dummy node  $s'$  in  $T$  with weight vector  $(-\infty, -\infty)$ ; the node  $s'$  is omitted when determining the rank space, preorder ranks, and depths in  $T$ ). Each set of the data structures can be conceptualized as a pair, consisting of a reporting structure proper and a certain navigational, auxiliary data structure; below we introduce them in this particular order.

One set comprises the structures  $D_l$  and  $A_l$ , defined as follows:

- $D_l$  is a 1-dimensional path dominance reporting structure (Lemma 8) 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  $x'$ ;
- $A_l$  is a weighted ancestor query structure over  $M_l$  (Lemma 9), when its nodes are assigned the  $1^{st}$  weights of the corresponding nodes in  $T$ .

Another set comprises the structures  $E_l$  and  $F_l$ , defined as follows:

- $E_l$  is a 1-dimensional path dominance reporting structure (Lemma 8) 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  $x'$ ;
- $F_l$  is a  $(1, 2, \epsilon)$ -dimensional path dominance reporting structure (Lemma 8) 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  $\kappa$  is the label assigned to the node in  $T_l$  corresponding to  $x'$ .

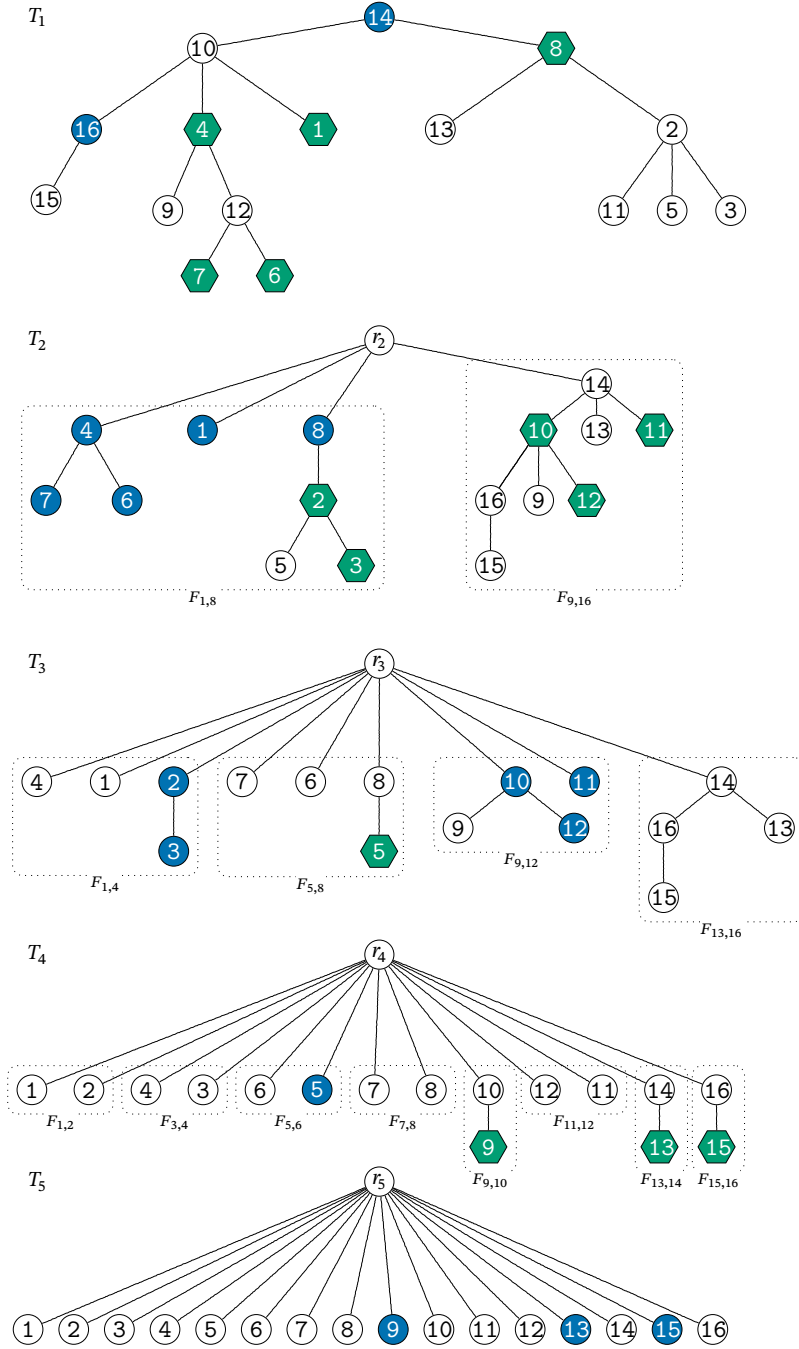
Finally, the following data structures are also maintained:

- $T'_l$ , the indicator tree of  $(T_l, M_l)$ ;
- $T''_l$ , the indicator tree of  $(T_l, N_l)$ ;
- $P_l$ , an array where  $P_l[x]$  stores the preorder number of the node in  $T$  corresponding to a node  $x$  in  $M_l$ .

The features of some of these data structures are summarized in Table 2. The following Lemma 10 states the space cost of our data structures.

► **Lemma 10.** *The data structures built in Section 4.3 occupy  $\mathcal{O}(n)$  words of space.*

**Proof.** As mentioned in Section 2.4, all the  $T_l$ s occupy  $n + \mathcal{O}(n)$  words. By Lemma 2, each  $T'_l$  or  $T''_l$  uses  $3n + \mathcal{O}(n)$  bits, so over all the  $\lg n / \lg \lg n$  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 the  $M(v)$ s is precisely  $n$ . Hence all the  $P_l$ s have  $n$  entries in total and thus uses  $n$  words. By Lemma 8, the size of each  $D_l$  in words is linear in



■ **Figure 6** Hierarchical tree extraction with branching factor 2 for a tree weighted over  $\{1, 2, \dots, 16\}$  (weights are given with the nodes). For each  $l \in \{1, 2, 3, 4, 5\}$ , blue, circle-shaped nodes in  $T_l$  are the nodes extracted to construct  $M_l$ , while green, hexagon-shaped nodes in  $T_l$  are the nodes extracted to construct  $N_l$

	Nodes	Assigned Weights	Query	Source
$D_l$	$\forall x' \in M_l$	$w_2(x)$	1D ancestor dominance reporting	Lemma 8
$A_l$	$\forall x' \in M_l$	$w_1(x)$	weighted ancestor	Lemma 9
$E_l$	$\forall x' \in M_l$	$w_1(x)$	1D ancestor dominance reporting	Lemma 8
$F_l$	$\forall x' \in N_l$	$(w_1(x), \text{label}(x'')), x'' \in T_l$	$(1, 2, \epsilon)$ -dimensional ancestor dominance	Lemma 8

■ **Table 2** Summary table for the data structures  $D_l, A_l, E_l$ , and  $F_l$  built in Section 4.3. Denoted by  $\mathbf{w}(x)$  is the original weight of the node  $x \in T$  that corresponds to  $x'$ . In the last row,  $x''$  is the node in  $T_l$  corresponding to  $x$

the number of nodes in  $M_l$ . The sum of the numbers of nodes in the  $M_l$ s over all levels of the range tree is equal to the sum of the sizes of all the  $M(v)$ s plus the number of dummy roots, which is  $n + \mathcal{O}(\lg n / \lg \lg n)$ . Therefore, all the  $D_l$ s occupy  $\mathcal{O}(n)$  words. By similar reasoning, all the  $E_l$ s and  $A_l$ s 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  $F_l$ s by  $\mathcal{O}(n)$  words in a similar way. All our data structures, therefore, use  $\mathcal{O}(n)$  words. ◀

#### 4.4 Supporting 2D ancestor dominance reporting

When delving into details of the search algorithm, it may be useful to recall the discussions in the beginning of Section 4.2. A synopsis of the somewhat more detailed exposition given therein could then go as follows. The search in the 2D case proceeds by eliminating the second weight from consideration and heeding the first weight only. One employs two strategies to that end, each necessitated by the anatomy of range trees. Recall again that when descending down a path in the range tree, the nodes of interest are those lying on the path together with their right siblings. If a (range tree) node is strictly to the right of the search path, we are left with only the first weight to worry about. Otherwise, the second weight is eliminated using the monotonicity property of the maximal nodes. Finding out the right siblings to explore is a “meta” query of its own, which is now  $(1, 2, \epsilon)$ -dimensional owing to “small” second weights.

Having thus dealt with a higher-level view, we now describe the algorithm for answering queries in detail, and analyze its time complexity:

► **Lemma 11.** *The data structures built in Section 4.3 answer an ancestor dominance reporting query in  $\mathcal{O}(\lg n + k)$  time, where  $k$  is the number of reported nodes.*

**Proof.** Let  $x$  and  $\mathbf{q} = (q_1, q_2)$  be the node and the weight vector given as query parameters, respectively. We define  $\Pi$  as the path in the range tree between and including the root and the leaf storing  $q_2$ . Let  $\pi_l$  denote the node at level  $l$  in  $\Pi$ ; then the root of the range tree is  $\pi_1$ . To answer the query, we perform a traversal of a subset of the nodes of the range tree, starting from  $\pi_1$ . 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 shall show how the invariant is maintained. Let  $x_v$  denote the node in  $T_l$  that corresponds to the  $\mathcal{T}_v$ -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 6. Our first step is to report all the nodes in the answer to the query that have corresponding nodes in

$M(v)$ . It is clear that since those nodes occur *only* in  $M(v)$ , this is the only place to discover them. There are two cases depending on whether (i)  $v = \pi_l$  or (ii)  $v \neq \pi_l$ ; 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  $x_v$ , using Proposition 1.

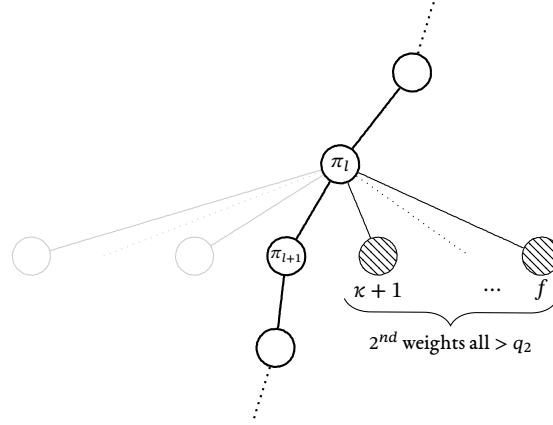
If (i) holds, then the non-dummy ancestors of  $x'_v$  in  $M_l$  correspond to all the ancestors of  $x$  in  $T$  that have corresponding nodes in  $M(v)$ . We then perform a weighted ancestor query using  $A_l$  to locate the highest ancestor,  $y$ , of  $x'_v$  in  $M_l$  whose  $1^{st}$  weight is at least  $q_1$ . Since the  $1^{st}$  weights of the nodes along any upward path in  $M_l$  are decreasing, the  $1^{st}$  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 (ii) holds, the maintained invariant guarantees that the  $2^{nd}$  weights of the nodes in  $M(v)$  are greater than  $q_2$ . Therefore, by performing a 1-dimensional path dominance reporting query in  $E_l$  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  $i^{th}$  child of  $v$ . We always visit  $\pi_{l+1}$  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  $x_v$ , using Proposition 1. Then the non-dummy ancestors of  $x''_v$  correspond to all the ancestors of  $x$  in  $T$  that have corresponding nodes in  $\cup_{i=1,2,\dots} M(v_i)$ . 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  $\pi_{l+1}$  is the  $\kappa^{th}$  child of  $v$ , and we set  $\kappa = 0$  if  $\pi_{l+1}$  is not a child of  $v$ . For each node,  $t$ , returned when answering this query, if its  $2^{nd}$  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 using an  $f$ -bit word to flag the children of  $v$ ). Figure 7 illustrates the considerations in this paragraph.

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  $\pi_l$ , 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  $\Pi$ , and  $\mathcal{O}(\lg n / \lg \lg n + k)$  time when visiting nodes in  $\Pi$ . Indeed,  $F_l$  is built using Lemma 8 and therefore has  $\mathcal{O}(1 + k)$  query time. Per a range-tree node in  $\Pi$  we pay  $\mathcal{O}(1)$  to initiate the search, hence the  $\mathcal{O}(\lg n / \lg \lg n)$  additive factor. On the other hand, the  $\mathcal{O}(1)$ -time cost of



■ **Figure 7** An illustration for the proof of Lemma 11. When visiting  $\pi_{l+1}$ , we use the  $A_l$  structure to adjust query parameters. When deciding which children, among  $\kappa+1, \dots, f$ , to visit, we use the  $F_l$  structure

initiating the search is charged to a tree node that has been reported during the visit of the current range-tree node outside of  $\Pi$ .

We thus conclude that the query times spent on all these structures throughout the execution of the algorithm sum up to  $\mathcal{O}(\lg n + k)$ . ◀

Finally, Lemma 10 and Lemma 11 are combined to produce

► **Theorem 12.** *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 reported nodes.*

## 5 Path successor

We first solve the path successor problem when  $d = 1$ , and extend the result to  $d > 1$  via Lemma 6.

On a high level, the data structure for the one-dimensional path successor is built as follows. For a tree weighted with scalars, one considers a range tree over its scalar weights; henceforth,  $R$  denotes this range tree. Befitting the range trees paradigm, each node of the thus-constructed range tree  $R$  contains certain summary structures. First, they maintain the corresponding tree extraction, for an efficient lookup of the path component of the query. Secondly, they allow fast successor queries. As previously, all the tree extractions  $T_u$  corresponding to a single level  $l$  of the range tree  $R$  are accommodated inside a single data structure  $T_l$ .

The query algorithm proceeds as a binary search over a certain path in the range tree  $R$ : This path leads from the root of  $R$  to the leaf containing the query weight. The goal is to locate summary structures that can be used to compute the answer.

For the binary search to work, one needs to be able to map the query path to the paths in summary structures. In Section 5.1, we describe a certain subroutine that maps a query node to a node in the summary structure. Then, the query algorithm itself is described in Section 5.2.

### 5.1 Core subroutine for node mapping

This section focuses on the following situation. One is given a node  $x \in T$ , as well as a node  $u \in R$ , belonging to the level  $l$  of the range tree  $R$ . The goal is to locate the node  $x_{u,l}$  in  $T_l$  that corresponds to the  $\mathcal{T}_u$ -view of  $x$ .

To achieve these results, we build the following data structures. The topology of  $T$  is stored using Lemma 2. We define a binary range tree  $R$  over  $[n]$ , and build the associated hierarchical tree extraction as in Section 2.4; as in Section 4,  $T_l$  denotes the auxiliary tree built for each level  $l$  of  $R$ , and  $\mathcal{T}_v$  denotes the tree extraction from  $T$  associated with the range of node  $v \in R$ . We represent  $R$  using Lemma 2, and augment it with the ball-inheritance data structure  $\mathcal{B}$  from Lemma 4(a), as well as with the following data structures.

First, for  $R$ , we maintain an annotation  $I$ , such that  $I[u]$  stores a quadruple  $\langle a_u, b_u, s_u, t_u \rangle$  for an arbitrary node  $u \in R$  at level  $l$ , 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  $T_l$ .

Furthermore, for each level  $L$  that is a multiple of  $\lceil \lg \lg n \rceil$ , which we call a *marked* level, we build the following data structures. 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 [14] in  $A_u$ . Assuming the availability of a  $\mathcal{O}(n^\delta)$ -bit universal table, where  $\delta$  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  $A_u$ .

► **Lemma 13.** *The data structures built in Section 5.1 occupy  $\mathcal{O}(n)$  words of space.*

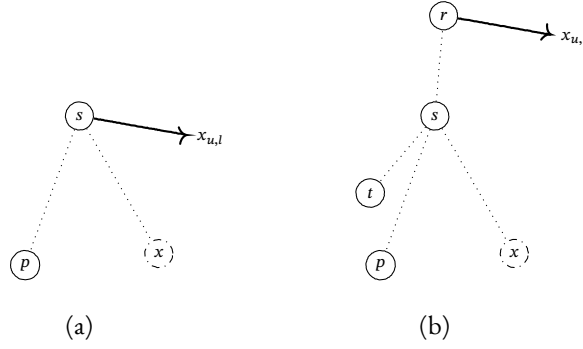
**Proof.** The space occupied by the annotation array  $I$  is clearly  $\mathcal{O}(n)$  words, because the number of nodes in  $R$  is  $\mathcal{O}(n)$ .

The size of the index in bits is  $\mathcal{O}(\lg \lg n)$  times the number of entries in  $A_u$ . For a marked (for that matter, any) level  $L$  all the  $S_u$ -structures thus 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 the entire tree  $R$  is  $\mathcal{O}(n)$  words. ◀

Prior to proceeding to Lemma 14, which is concerned with the actual query algorithm, some notes are in order.

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  $T \rightarrow \mathcal{T}_u \rightarrow T_l$ . In the first transition,  $T \rightarrow \mathcal{T}_u$ , given an original node  $x \in T$ , we are looking for its  $\mathcal{T}_u$ -view,  $x_u$ . That is, although  $\mathcal{T}_u$  is (conceptually) obtained from  $T$  through a *series* of extractions (i.e. as we construct the range tree), the wish is to “jump” many successive extractions at once, as if  $\mathcal{T}_u$  were extracted from  $T$  *directly*. This would be trivial to achieve through storing an indicator tree per range  $u$ , i.e. for each pair  $(T, \mathcal{T}_u)$ , if it were not for a prohibitive space-cost – the number of bits at least quadratic in the number of nodes. One can avoid extra space cost altogether and directly use Lemma 3 to explicitly traverse the hierarchy of extractions. In this case, the time cost is proportional to the height of the range tree, and hence becomes the bottleneck. To overcome this difficulty, we use the predecessor/successor structure  $S_u$  to speed up the process, while the strategy of marking a subset of levels keeps the space usage linear.





**Figure 8** An illustration to the proof of Lemma 14. Node  $x$ , which could have weight in  $u$ 's range or not, is represented by a dash-dotted circle. The subfigures (a) and (b) respectively represent the cases in which the weight of  $s$  is in and not in  $u$ 's range

In turn, in the  $\mathcal{T}_u \rightarrow T_l$ -transition, we are looking for the identity of  $x_u$  in  $T_l$ . For this second transformation, we recall from Section 2.4 that  $\mathcal{T}_u$  is embedded within  $T_l$ , and the nodes of  $\mathcal{T}_u$  must lie contiguously in the preorder sequence of  $T_l$ .

Lemma 14 shows how to perform node mapping. Our annotation arrays  $I_u$  then use this observation to perform the transformation.

► **Lemma 14.** *The data structures built in Section 5.1 enable determining  $x_{u,l} \in T_l$  corresponding to the  $\mathcal{T}_u$ -view of  $x \in T$ , in  $\mathcal{O}(\log^{\epsilon'} n)$  time, where  $\epsilon'$  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  $l$ .*

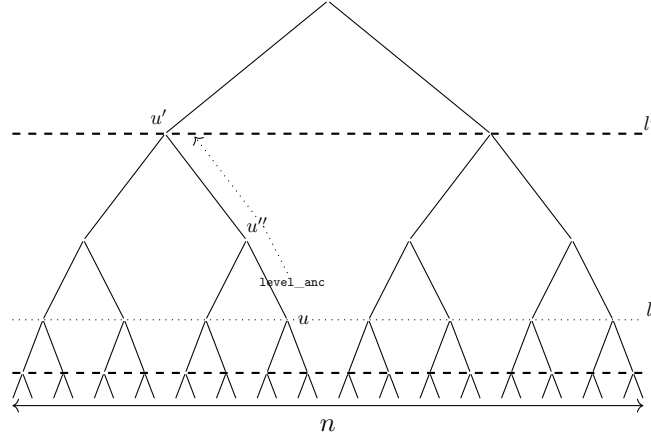
**Proof.** Let us show how to answer queries using the data structures built in Section 5.1. Resolving a query falls into two distinct cases. The first is when the level  $l$ , at which the query node  $u$  resides, is marked; the second is when it is not.

When the level  $l$  is marked, we use the structures  $S_u$  stored therein, directly. We adopt the strategy of He et al. [18] to find  $x_{u,l}$ . First, for an arbitrary index  $i$  to  $A_u$ , we observe that the node  $A_u[i] \in T$  corresponds to the 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  $\mathcal{O}(\lg \lg n)$ -time query and  $\mathcal{O}(1)$  calls to the  $\mathcal{B}$ -structure. This results in  $\mathcal{O}(\lg^{\epsilon'} n)$  time in total. We then determine the lowest common ancestor  $s \in T$  of  $x$  and  $p$ , in  $\mathcal{O}(1)$  time. Further, Figure 8 illustrates the two possible cases for the weight of the node  $s$ : it either belongs to the range  $[a_u, b_u]$ , or it does not.

If the weight of  $s$  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  $s$  in  $A_u$ , and  $(s_u + j - 1)$  is the sought  $x_{u,l}$ . This case subsumes all the corner cases, too. Indeed, if the node  $x$ 's weight was from  $[a_u, b_u]$  to begin with, the predecessor query would duly return  $p = x$ ; if  $p \neq x$  and  $p \in \mathcal{A}(x)$ , then  $p = s$ .

If the weight of  $s$  is not in  $[a_u, b_u]$ , a final query to  $S_u$  returns the successor  $t$  in  $A_u$  of  $s$ ; the node  $t$  must exist because of  $p$ . Let  $\kappa$  be the position of  $t$  in  $A_u$ . Then the parent of the node  $(s_u + \kappa - 1)$  in  $T_l$  is the sought node  $x_{u,l}$ . Indeed, it is clear that the sought node  $x_{u,l}$  should correspond to a proper ancestor of  $s$ . A valid choice to reach such an ancestor would be via the preorder successor  $t$  of  $s$  such that  $t$ 's weight is in  $[a_u, b_u]$ ; no other nodes between  $t$  and  $s$  have weights in  $[a_u, b_u]$ .

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  $\mathcal{O}(\lg^{\epsilon'} n)$ .



■ **Figure 9** Illustration to the proof of Lemma 14. Marked levels are shown in dashed lines. Level  $l$  corresponds to the query node  $u$ , and is represented as a dotted line. The node  $u'$  is the ancestor of the node  $u$  on the marked level  $l'$ . One ascends from the level  $l$  to the marked level  $l'$  via `level_anc` available through the encoding of the binary tree  $R$ . The node  $u''$  represents any node on the path from  $u$  to  $u'$ , at a level  $l''$  such that  $l' \leq l'' \leq l$ .

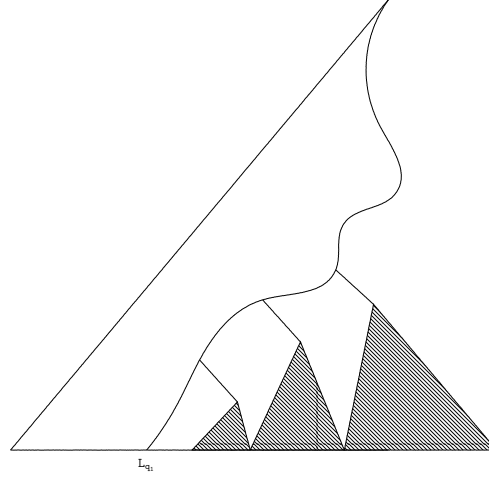
When the level  $l$  is *not* marked, we ascend to the lowest ancestor  $u'$  of  $u$  residing on a marked level  $l'$ , and reduce the problem to the previous case. More precisely, via the navigation operations (`level_anc` to move to a parent, and `depth` to determine whether the level is marked) 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 then find the node  $x'_{u',l'}$  in  $T_{l'}$  that corresponds to the  $\mathcal{T}_{u'}$ -view of the node  $x$  in  $T$ . Afterwards, we initialize a variable  $s$  to be  $x_{u',l'}$ . At each level  $l'' \leq l' \leq l$  this variable  $s$  represents the node  $x''_{u'',l''} \in T_{l''}$  that corresponds to the  $\mathcal{T}_{u''}$ -view of the node  $x \in T$ ; here  $u''$  is the node of the range tree such that it is an ancestor of  $u$  and a descendant of  $u'$ . We descend down to the original level  $l$ , back to the original query node  $u$ , all the while adjusting the node  $s$  as we move down a level, analogously to the proof of Lemma 6. As we arrive, in time  $\mathcal{O}(\lg \lg n)$ , at node  $u$ , the variable  $s$  stores the answer,  $x_{u,l}$ . Figure 9 facilitates these discussions.

In the second case, too, the term  $\mathcal{O}(\lg^{\epsilon'} n)$  dominates the time complexity, as climbing from the current level up to the closest marked level<sup>7</sup> and back is an additive term of  $\mathcal{O}(\lg \lg n)$ . Therefore, the query is answered in  $\mathcal{O}(\lg^{\epsilon'} n)$  time. ◀

To conclude the section, Lemmas 13 and 14 can be combined to produce

► **Lemma 15.** *Let  $R$  be a binary range tree with topology encoded using Lemma 2, and augmented with the ball-inheritance data structure  $\mathcal{B}$  from Lemma 4(a). With an additional space of  $\mathcal{O}(n)$  words, the node  $x_{u,l}$  in  $T_l$  corresponding to the  $\mathcal{T}_u$ -view of  $x$  can be found in  $\mathcal{O}(\log^{\epsilon'} n)$  time, where  $\epsilon'$  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  $l$ .*

<sup>7</sup> One could reach the marked level in one leap, in  $\mathcal{O}(1)$  time, using `level_anc`. This however has no bearing on the asymptotical time bound, because we later descend to the original level one level at a time, anyway.



■ **Figure 10** Illustration to the proof of Lemma 16. Finding the path successor amounts to locating the deepest node such that its right child contains the answer to the query. For the path successor query with parameters  $[q_1, \infty)$  and  $P_{x,y}$ , one would like to efficiently “intersect” the path  $P_{x,y}$  with a 1D range corresponding to a shaded triangle. Shaded triangles represent the right children of the nodes on a root-to-leaf path leading to  $L_{q_1}$

## 5.2 Answering path successor queries

Having dealt in Section 5.1 with the subroutine for node mapping, we now proceed to the actual solution.

Here, in addition to the data structures built in Section 5.1, each  $T_l$  is further augmented with succinct indices  $m_l$  (resp.  $M_l$ ) from Lemma 5(b), for path minimum (resp. path maximum) queries; here, the weights of the nodes of  $T_l$  are the weights of their corresponding nodes in  $T$ . With this, we are now ready to describe, in Lemma 16, the algorithm for answering queries. The idea is to conduct binary search in the path from the root of the range tree  $R$  to the leaf corresponding to the query weight, to look for the lowest ancestor,  $\pi_f$ , of this leaf such that its extracted tree contains the answer. At each iteration, we use the data structures built in Section 5.1 for fast adjustments of the query path. Then, the extracted tree corresponding to the right child of  $\pi_f$  must contain the answer. And, since all the nodes in this extracted tree have weights greater than the query weight, a path minimum query computes the final result.

► **Lemma 16.** *A 1D-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  $\epsilon$  is an arbitrary constant in  $(0, 1)$ .*

**Proof.** Let  $x, y$  and  $Q = [q_1, q'_1]$  be respectively the nodes and the orthogonal range given as query parameters. As in the proof of Lemma 6, we focus only on the path  $A_{x,z}$ , where  $z$  is  $\text{LCA}(T, x, y)$ . 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  $\Pi$  be the root-to-leaf path to  $L_{q_1}$  in  $R$ , and let  $\pi_l$  be the node at level  $l$  of  $\Pi$ ; Figure 10 shows such a path schematically. We binary search in  $\Pi$  for the deepest node  $\pi_f \in \Pi$  whose associated extraction  $\mathcal{T}_{\pi_f}$  contains the node corresponding to the answer to the given query, as follows.

We initialize two variables: *high* as 1 so that  $\pi_{\text{high}}$  is the root of  $R$ , and *low* as the height of  $R$  so that  $\pi_{\text{low}}$  is the leaf  $L_{q_1}$ . We first check whether  $\mathcal{T}_{\pi_{\text{low}}}$  already contains the answer, by fetching the node  $x'$  in  $T_{\text{low}}$  corresponding to the  $\mathcal{T}_{\pi_{\text{low}}}$ -view of  $x$ , using Lemma 15. If  $x'$  exists, we examine its corresponding node  $x''$  in  $T$  (fetched via  $\mathcal{B}$ ). We check whether

$x''$  is on  $A_{x,z}$ , using `depth` and `level_anc` operations. If  $x'' \in A_{x,z}$ , then  $x''$  is the final answer. If not, this establishes the invariant of the ensuing search:  $\mathcal{T}_{\pi_{high}}$  contains a node corresponding to the answer, whereas  $\mathcal{T}_{\pi_{low}}$  does not.

At each iteration, therefore, we set (via `level_anc` in  $R$ )  $\pi_{mid}$  to be the node mid-way from  $\pi_{low}$  to  $\pi_{high}$ . We then fetch the nodes  $x', z'$  in  $T_{mid}$  corresponding to the  $\mathcal{T}_{\pi_{mid}}$ -views of respectively  $x$  and  $z$ , using Lemma 15. 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 1<sup>st</sup> weight,  $\mu$ , 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 by setting  $high$  to  $mid$  if  $\mu \geq q_1$ , and by setting  $low$  to  $mid$ , otherwise.

Once  $\pi_f$  is located, it must hold for  $\pi_f$  that (i) it is its left child that is on  $\Pi$  [26]; 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  $\pi_f$ , we also found the nodes in  $T_f$  corresponding to the  $\mathcal{T}_{\pi_f}$ -views of  $x$  and  $z$ ; they can be further used to find the nodes in  $T_{f+1}$ ,  $x^*$  and  $z^*$ , corresponding to the  $\mathcal{T}_v$ -views of  $x$  and  $z$ . We then use  $m_{f+1}$  to find the node in  $A_{x^*,z^*}$  with the minimum 1<sup>st</sup> weight, whose corresponding node in  $T$  is the answer.

The total query time is determined by that needed for binary search. Each iteration of the binary 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 2, uses only  $2n + \mathcal{O}(n)$  bits. As stated in Section 2.4, all the structures  $T_l$  occupy  $\mathcal{O}(n)$  words in total. The space cost of the structure from Lemma 15 built for  $R$  is  $\mathcal{O}(n)$  words. The  $\mathcal{B}$ -structure occupies another  $\mathcal{O}(n)$  words. The  $m_l$ - and  $M_l$ -structures 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. ◀

Combining Lemmas 6 and 16, we arrive at the following

► **Theorem 17.** *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  $\epsilon$ .*

**Proof.** We instantiate Section 3 with  $g(x) = x$  and the semigroup sum operator  $\oplus$  as  $x \oplus y = \arg \min_{t \in \{x,y\}} \{w_1(t)\}$ . Lemma 6 applied to Lemma 16 yields the space bound of  $\mathcal{O}(n \lg^{d-1} n)$  words and the query time complexity of  $\mathcal{O}(\lg^{d-1+\epsilon} n)$ . ◀

## 6 Path counting

In this section, we use a tree decomposition technique called *tree covering*. This is different from other sections, where we use tree extraction.

Section 6.1 introduces tree covering. Based on the exposition, we shall summarize the main idea of solving the path counting problem for the base case of trees weighted with  $(0, d, \epsilon)$ -dimensional weight vectors. Then, Section 6.2 lays out the details of solving the path counting problem in the base case, which is further generalized to trees weighted with  $d$ -dimensional weight vectors, via the space-reduction approach described in Section 3.2.

## 6.1 High-level overview

Tree covering first appeared in [13, 17, 11] as a method of succinct representation of ordinal trees. The original tree is split into *mini-trees*, given a certain parameter  $L$ :

► **Lemma 18** ([11]). *A tree with  $n$  nodes can be decomposed into  $\Theta(n/L)$  subtrees of size at most  $2L$ . These are pairwise disjoint aside from the subtree roots. Furthermore, aside from the edges incident to the subtree roots, there is at most one edge per subtree leaving a node of a subtree to its child in another subtree.*

Figure 11 (a) gives an example of tree covering with parameter  $L = 3$ .

An interesting feature of tree covering is that each of the mini-trees, in turn can, be recursively decomposed into *micro-trees*, with another parameter  $L' < L$ . For example, in Figure 11 (a), we further decompose the mini-trees into micro-trees, using  $L' = 2$ .

It turns out that when the weight vector of a node can be packed into  $\mathcal{O}(\lg n)$  bits, counting queries can be executed in constant time. The key machinery used is tree covering. Namely, we decompose the input tree  $T$  into mini- and micro-trees using respectively parameters  $L$  and  $L'$ . At the root of each mini-tree, we precompute for each possible query weight range, how many nodes between the mini-tree root and the root of the entire tree  $T$  have weight vectors in this range. Similarly for micro-trees: For each possible  $(0, d, \epsilon)$ -dimensional hyper-rectangle, we precompute the same information, now from the root of the micro-tree to the root of the encompassing mini-tree. The key is to choose the parameter  $L'$  such that *intra*-micro-tree queries are executed in constant-time via a lookup into precomputed table. Then, the answer to a query is computed in three steps – the query node to micro-tree root, micro-tree root to the mini-tree root, and mini-tree root to the root of  $T$ .

## 6.2 Data structures

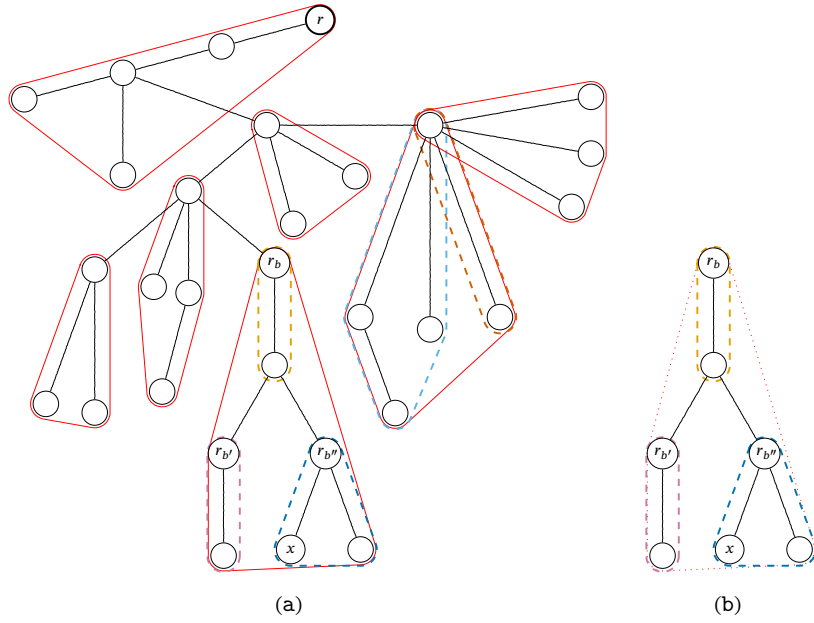
Let, indeed,  $T$  be a given ordinal tree on  $n$  nodes, each node of which is assigned a weight vector in  $(0, d, \epsilon)$  dimensions. Let  $r$  be the root of  $T$ . In our solution to the  $(0, d, \epsilon)$ -dimensional path counting problem for  $T$ , we set  $c = \lceil \lg^\epsilon n \rceil$ , and use Lemma 18 to decompose  $T$  into mini-trees with parameter  $L = c^{2d} \lg n$ . Each of the mini-trees is further subject to decomposition into micro-trees with parameter  $L' = c^{2d}$ . We chose the parameter  $L'$  so that *intra*-micro-tree queries are executed in constant-time by virtue of a precomputed table  $\mathcal{T}$  of size  $\mathcal{O}(n)$ , indexed by micro-trees. For any given node  $x \in T$ , the solutions of [13, 17, 11] provide constant-time access to the mini-tree  $\tau$  and micro-tree  $\tau'$  containing the node  $x$ , as well as the address of the micro-tree  $\tau'$  in the table  $\mathcal{T}$ , using  $\mathcal{O}(n)$  bits of space.

Furthermore, let us denote by  $r_b$  the root of a mini- or micro-tree  $b$ , for any  $b$  that shall be clear from the context. Each mini- or micro-tree  $b$  stores an array  $b.cnt$ , indexed by a tuple from  $([c] \times [c])^d$ , with the following contents:

- for a mini-tree  $b$ , an  $\mathcal{O}(\lg n)$ -bit number  $b.cnt[Q]$  stores the number of the nodes with weight vectors falling within the range  $Q$  on the path  $A_{r_b, r}$  in  $T$ , where  $r$  is the root of  $T$ ;
- for a micro-tree  $b'$  inside a mini-tree  $b$ , an  $\mathcal{O}(\lg \lg n)$ -bit number  $b'.cnt[Q]$  stores the number of the nodes with weight vectors falling within the range  $Q$  on the path  $A_{r_{b'}, r_b}$ .

We also precompute a look-up table  $D$  that is indexed by a quadruple from the following Cartesian product:

- all possible micro-tree topologies  $\tau$ , *times*
- all possible assignments  $\lambda$  of weight vectors to the nodes of  $\tau$ , *times*
- all nodes in  $\tau$ , *times*



■ **Figure 11** Tree covering of Farzan and Munro [11] is shown in Figure (a), for parameter  $L = 3$ . Mini-trees are outlined via red solid lines. The mini-trees can share roots only, and there is at most one arc leading from a non-root node of a mini-tree to the root of another mini-tree. Figures (a)-(b) illustrate the proof of Lemma 19, when  $L = 3$  and  $L' = 2$ . Mini- and micro-trees are represented by solid and dashed lines, respectively. For simplicity, decompositions to micro-trees are shown for two mini-trees only. For a query node  $x$ , we show the root  $r_b$  of the mini-tree  $b$  and the root  $r_{b'}$  of a micro-tree  $b'$  containing  $x$ . In Figure (b), a mini-tree from (a) is further decomposed into micro-trees, represented by dashed lines. The roots  $r_{b'}$  and  $r_{b''}$  of the micro-trees store precomputed information only up until the root  $r_b$  of the encompassing mini-tree

- all possible query orthogonal ranges  $Q$ .

Let  $\lambda$  be a labeling of a micro-tree  $\tau$  with  $(0, d, \epsilon)$ -dimensional weight vectors. Then the entry  $D[\tau, \lambda, x, Q]$  stores the number of nodes on the path  $A_{x, r_\tau}$  in  $\tau$ , such that their weight vectors belong to the range  $Q$ .

We now show how to use these data structure to answer queries.

► **Lemma 19.** *The data structures in this section can answer a  $(0, d, \epsilon)$ -dimensional path counting query in  $\mathcal{O}(1)$  time, for any constant integer  $d \geq 1$ .*

**Proof.** Let  $P_{x,y}$  and  $Q$  be, respectively, the path and the orthogonal range given as the parameters to the query. For the reasons given in Lemma 6, we describe only how to answer the query over  $A_{x,z}$ , where  $z = \text{LCA}(T, x, y)$ . We assume the encoding of  $T$  as in Lemma 2, so the LCA operator is available; the space overhead is only  $\mathcal{O}(n)$  bits, i.e. negligible with respect to the space bound we are ultimately aiming at.

We further notice that answering the path counting query over  $A_{x,z}$  is equivalent to answering two path counting queries, one over  $A_{x,r}$  and another over  $A_{z,r}$ , and taking their arithmetic difference. It is thus sufficient to describe the procedure of answering the query over  $A_{x,r}$ , and analyze its running time, as the query over  $A_{z,r}$  can be handled similarly.

The key observation is that overall we perform a constant number of constant-time operations. Indeed, using data structures of [11], we first identify, in  $\mathcal{O}(1)$  time, the mini-tree  $b$  and the micro-tree  $b'$  containing the node  $x$ , as well as the encoding of  $b'$  (see Figure 11 for an illustration). From the (disjoint) decomposition of the path  $A_{x,r} = A_{x,r_{b'}} \cup A_{r_{b'},r_b} \cup A_{r_b,r}$ , it is now immediate that the answer to the query over  $A_{x,r}$  is  $\text{res}_x = D[b', \text{lab}(b'), x', Q] + b.\text{cnt}[Q] + b'.\text{cnt}[Q]$ , where  $\text{lab}(b')$  is the labeling of the micro-tree  $b'$ , and  $x'$  is the preorder number of the node  $x \in T$  in  $b'$ , provided by the standard encodings [13, 17, 11]. One finds the answer  $\text{res}_z$  to the path counting query over  $A_{z,r}$  analogously. Then the final answer is  $\text{res}_x - \text{res}_z$ . Therefore, our algorithm runs in  $\mathcal{O}(1)$  time. ◀

We now analyze the space cost of our data structures.

► **Lemma 20.** *The data structures in this section occupy  $\mathcal{O}(n \lg \lg n)$  bits when  $\epsilon \in (0, \frac{1}{4d})$ .*

**Proof.** To analyze the space cost, we tally up the costs of the main constituents of our data structure: the *cnt*-arrays stored at the roots of each mini- and micro-tree, and the  $D$ -table.

There being  $\Theta(\frac{n}{c^{2d} \lg n})$  mini-trees, each of which contains an array of  $c^{2d}$  elements,  $\mathcal{O}(\lg n)$  bits each, the associated *cnt*-arrays contribute  $\mathcal{O}(\frac{n}{c^{2d} \lg n} \times c^{2d} \times \lg n) = \mathcal{O}(n)$  bits.

Analogously, for the  $\Theta(n/c^{2d})$  micro-trees, the net contribution of the associated *cnt*-arrays is  $\mathcal{O}(n/c^{2d} \times c^{2d} \times \lg(c^{2d} \lg n)) = \mathcal{O}(n \lg \lg n)$  bits, which is the space claimed in the statement.

It suffices to show, therefore, that the space occupied by the  $D$ -structure can not exceed  $\mathcal{O}(n \lg \lg n)$  bits; as demonstrated below, it is much less. Indeed, as mentioned in Lemma 18, each micro-tree can have up to  $2L' = 2c^{2d}$  nodes, which gives us less than  $2^{2 \cdot 2L'} = 2^{2 \cdot 2c^{2d}}$  possible topologies  $\tau$ . In turn, each of the  $2c^{2d}$  nodes can independently be assigned  $c^d$  possible weight vectors; hence the number of possible configurations  $\lambda$  is at most  $(c^d)^{2c^{2d}}$  (the number of strings of length  $2c^{2d}$  over alphabet  $[c^d]$ ). Furthermore, the  $c^{2d}$  query orthogonal ranges  $Q$  make for  $2c^{2d} \times c^{2d} = 2c^{4d}$  distinct queries, i.e. the number of nodes times the number of ranges. The number of entries in the table  $D$  is thus  $\mathcal{O}(2^{4c^{2d}} \cdot (c^d)^{2c^{2d}} \cdot 2c^{4d}) = \mathcal{O}((4c^d)^{2c^{2d}} c^{4d})$ . The term  $c^{4d}$  is  $\mathcal{O}(\lg n)$ . To upper-bound the term  $(4c^d)^{2c^{2d}}$ , we notice that  $c^d = \lceil \lg^\epsilon n \rceil^d < \sqrt[4]{\lg n}/4$  for sufficiently large  $n$ . Therefore, we have the following chain of inequalities for sufficiently



large  $n$ :

$$\begin{aligned}
 (4c^d)^{2c^{2d}} &< \left(\sqrt[d]{\lg n}\right)^{2c^{2d}} < \left(\sqrt[d]{\lg n}\right)^{2\sqrt{\lg n}} = \left(\sqrt{\lg n}\right)^{\sqrt{\lg n}} \\
 &= \left(2^{\lg \sqrt{\lg n}}\right)^{\sqrt{\lg n}} = 2^{\sqrt{\lg n} \cdot \lg \sqrt{\lg n}} < 2^{\sqrt{\lg n} \cdot \frac{\sqrt{\lg n}}{2}} = \sqrt{n}
 \end{aligned}$$

Thus, the number of entries in  $D$  is at most  $\mathcal{O}(\sqrt{n} \lg^{4d\epsilon} n)$ . Each entry holding a value of  $\mathcal{O}(\lg c^{2d}) = \mathcal{O}(\lg \lg n)$  bits, the table  $D$  occupies  $\mathcal{O}(\sqrt{n} \lg \lg n \lg^{4d\epsilon} n) = o(n)$  bits, in total.

Finally, as shown above, the number of ways to assign weight vectors to nodes of a micro-tree is a  $\mathcal{O}(L' \lg \lg n)$ -bit number. Thus, the storage space for the labelings of each of the  $\Theta(n/L')$  micro-trees amounts to  $\mathcal{O}(n \lg \lg n)$  bits. ◀

With Lemmas 19 and 20, we have the following

► **Lemma 21.** *Let  $d \geq 0$ ,  $\epsilon \in (0, \frac{1}{4d})$  be constants. A tree  $T$  on  $n$  nodes, in which each node is assigned a  $(0, d, \epsilon)$ -dimensional weight vector, can be represented in  $\mathcal{O}(n \lg \lg n)$  bits of space such that a path counting query is answered in  $\mathcal{O}(1)$  time.*

Finally, instantiating Section 3.2 with  $g(x) \equiv 1$  and  $\oplus$  as the regular arithmetic addition operation  $+$  in  $\mathbb{R}$ , we can apply Lemma 7 to Lemma 21 iteratively to obtain the following

► **Theorem 22.** *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  $\mathcal{O}(n(\lg n / \lg \lg n)^{d-1})$  words such that a path counting query can be answered in  $\mathcal{O}((\lg n / \lg \lg n)^d)$  time.*

## 7 Path reporting

In this section, we solve the path reporting problem. The solution immediately follows from combining the ideas in Section 4.1, Section 3.2. and the result of Chan et al. [4].

More specifically, we first design a solution for solving the path reporting problem for trees weighted with  $(1, d, \epsilon)$ -dimensional weight vectors. This solution is then generalized to trees weighted with  $d$ -dimensional weight vectors, via the space-reduction approach described in Section 3.2.

Analogously to Section 4.1, a  $d$ -dimensional version of the problem is reduced to solving a  $(1, d, \epsilon)$ -dimensional problem: For each possible  $(d-1)$ -dimensional hyper-rectangle, we precompute the data structure for solving the relevant 1-dimensional problem. In turn, for solving the 1-dimensional version of the problem, we use the following result of Chan et al. [4]:

► **Lemma 23 ([4]).** *An ordinal tree on  $n$  nodes whose weights are drawn from  $[n]$  can be represented using  $\mathcal{O}(n \lg^\epsilon n)$  words of space, such that path reporting queries can be supported in  $\mathcal{O}(\lg \lg n + k)$  time, where  $k$  is the number of reported nodes and  $\epsilon$  is an arbitrary positive constant.*

Lemma 23 implies the following

► **Lemma 24.** *Let  $d \geq 1$  and  $0 < \epsilon < \frac{1}{2(d-1)}$  be constants, and let  $T$  be an ordinal tree on  $n$  nodes, in which each node is assigned a  $(1, d, \epsilon)$ -dimensional weight vector. Then,  $T$  can be represented in  $\mathcal{O}(n \lg^{\epsilon'} n)$  words of space, for any constant  $\epsilon' \in (2(d-1)\epsilon, 1)$ , so that a path reporting query can be answered in  $\mathcal{O}(\lg \lg n + k)$  time, where  $k$  is the number of reported nodes.*

**Proof.** In brief, we build a path reporting data structure from Lemma 23 for each possible orthogonal range over the last  $(d - 1)$  dimensions. When presented with a query, we directly proceed to the appropriately-tagged (by the last  $(d - 1)$  weights) reporting structure, and launch the query therein. A detailed exposition follows.

We assume the encoding of  $T$  as in Lemma 2; the space incurred is only  $\mathcal{O}(n)$  bits, i.e. negligible with respect to the terms derived below.

For any  $(0, d - 1, \epsilon)$ -dimensional orthogonal range  $G$ , we build an explicit weighted tree  $E_G$  as the extraction of the node set  $\{v \mid v \in T \text{ and } \mathbf{w}_{2,d}(v) \in G\}$  from  $T$ . The weight of a node in  $E_G$  is the 1<sup>st</sup> weight of its  $T$ -source. If  $E_G$  has a dummy root, then its weight is  $-\infty$ .

$E_G$  is represented using Lemma 23, in space that is at most  $\mathcal{O}(n \lg^\delta n)$  words, for an arbitrarily small positive  $\delta$ . In order to adjust the nodes between  $T$  and  $E_G$ , indicator tree  $T_G$  of  $(T, E_G)$  is maintained.

Accounting for all possible ranges  $G$ , we therefore have  $\mathcal{O}(n \lg^{\delta+2(d-1)\epsilon} n)$  words of space, in total. Setting  $\delta$  to be sufficiently small and assigning  $(\delta + 2(d - 1)\epsilon)$  to  $\epsilon'$  justifies the space claimed. All the  $T_G$ -structures collectively occupy  $\mathcal{O}(n \lg^{2(d-1)\epsilon} n)$  bits of space, which is  $\mathcal{O}(n)$  words.

Let  $P_{x,y}$  and  $Q$  be, respectively, the path and the orthogonal range given as the query parameters. Using the notation of Lemma 6, and for the same reasons as given therein, we concern ourselves only with answering the query over the path  $A_{x,z}$ , where  $z = \text{LCA}(T, x, y)$ . To answer the query, we first locate the relevant tree  $E_{Q'}$ , where  $Q' = Q_{2,d}$ , and launch a path reporting query in  $E_{Q'}$ , having adjusted the nodes  $x$  and  $z$  to their  $T_{Q'}$ -views as described in Proposition 1. Finally, the one-dimensional query in  $E_{Q'}$  executes in  $\mathcal{O}(\lg \lg n + k)$  time, by Lemma 23, thereby establishing the claimed time bound. Each returned node  $x$ 's original identifier is recovered as in Proposition 1. ◀

Instantiating Section 3.2 with  $g(x) = \{x\}$  and the semigroup sum operator  $\oplus$  as the set-theoretic union operator  $\cup$ , Lemma 7 and Lemma 24 combined imply the following

► **Theorem 25.** *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  $\mathcal{O}(n \lg^{d-1+\epsilon} n)$  words such that a path 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 reported nodes, for an arbitrarily small positive constant  $\epsilon$ .*

## 8 Conclusion

This article proposes solutions to the ancestor dominance reporting, path successor, path counting and path reporting problems, over ordinal trees weighted with multidimensional weight vectors. These problems generalize the classical orthogonal range searching problems, to the case of one dimension being replaced by a tree topology.

In solving these problems, we combine diverse data-structuring ideas and propose a few novel techniques that could be of interest in their own right.

We propose a linear-space,  $\mathcal{O}(\lg n + k)$  query time data structure for the 2D *ancestor dominance reporting problem*. This data structure matches the corresponding space bound for 3D dominance reporting of [1, 3]. When extended to  $d \geq 3$ , our data structures are still inferior to the combination of existing techniques in tree pre-processing with the latest results in dominance reporting [24]. Improving upon the latter is an open problem.

We solve the *path successor problem* in  $\mathcal{O}(n \lg^{d-1} n)$  words and time  $\mathcal{O}(\lg^{d-1+\epsilon} n)$  for  $d \geq 1$  and an arbitrary constant  $\epsilon > 0$ . We propose a solution to the *path counting problem*, with  $\mathcal{O}(n(\lg n / \lg \lg n)^{d-1})$  words of space and  $\mathcal{O}((\frac{\lg n}{\lg \lg n})^d)$  query time, for  $d \geq 1$ . These

results match or nearly match the best trade-offs of the respective range queries. We are also the first to solve path successor even for  $d = 1$ .

Finally, we solve the *path reporting problem* in  $\mathcal{O}(n \lg^{1+\epsilon} n)$  words of space and  $\mathcal{O}(\lg n + k)$  query time, for  $d = 2$ . This is a good result for  $d = 2$ , because the space remains the same as in the state-of-the-art solution for the Euclidean counterpart, whereas the slowdown in the additive term is sub-logarithmic. For  $d \geq 3$ , we propose an  $\mathcal{O}(n \lg^{d-1+\epsilon} n)$  words-of-space, and  $\mathcal{O}(\frac{\lg^{d-1} n}{(\lg \lg n)^{d-2}} + k)$  query-time solution. Thus for  $d \geq 3$ , we independently achieve what can be achieved by the state-of-the-art [25] result in the corresponding problem in  $\mathbb{R}^{d+1}$ , in conjunction with heavy-path decomposition [28]. The result of Nekrich [25], however, was published after the preliminary version of the present article, and is more complex.

By the dint of any two nodes uniquely determining a path, tree topologies generalize one-dimensional arrays. And when tree degenerates into a single path, the path query problems and their Euclidean counterparts become identical. The big question is how far this similarity stretches, and for which classes of problems can one hope to close the gap between the best results on the either side?

---

## References

- 1 Peyman Afshani. On dominance reporting in 3d. In *ESA*, pages 41–51, 2008. doi:10.1007/978-3-540-87744-8\4.
- 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. *ACM Trans. Algorithms*, 9(3):22:1–22:22, 2013. doi:10.1145/2483699.2483702.
- 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. doi:10.1007/s00453-016-0170-7.
- 5 Timothy M. Chan, Kasper Green Larsen, and Mihai Pătraşcu. Orthogonal range searching on the RAM, revisited. In *SoCG*, pages 1–10, 2011. doi:10.1145/1998196.1998198.
- 6 Bernard Chazelle. Computing on a free tree via complexity-preserving mappings. *Algorithmica*, 2:337–361, 1987. doi:10.1007/BF01840366.
- 7 Bernard Chazelle and Herbert Edelsbrunner. Linear space data structures for two types of range search. *Discrete & Computational Geometry*, 2:113–126, 1987. doi:10.1007/BF02187875.
- 8 Mark de Berg, Otfried Cheong, Marc J. van Kreveld, and Mark H. Overmars. *Computational geometry: algorithms and applications, 3rd Edition*. Springer, 2008.
- 9 Erik D. Demaine, Gad M. Landau, and Oren Weimann. On cartesian trees and range minimum queries. *Algorithmica*, 68(3):610–625, 2014. doi:10.1007/s00453-012-9683-x.
- 10 Martin Farach and S. Muthukrishnan. Perfect hashing for strings: Formalization and algorithms. In *CPM*, pages 130–140, 1996. doi:10.1007/3-540-61258-0\11.
- 11 Arash Farzan and J. Ian Munro. A uniform paradigm to succinctly encode various families of trees. *Algorithmica*, 68(1):16–40, 2014. doi:10.1007/s00453-012-9664-0.
- 12 Harold N. Gabow, Jon Louis Bentley, and Robert Endre Tarjan. Scaling and related techniques for geometry problems. In *STOC*, pages 135–143, 1984. doi:10.1145/800057.808675.
- 13 Richard F. Geary, Rajeev Raman, and Venkatesh Raman. Succinct ordinal trees with level-ancestor queries. *ACM Trans. Algorithms*, 2(4):510–534, 2006. doi:10.1145/1198513.1198516.
- 14 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. doi:10.4230/LIPIcs.STACS.2009.1847.

- 15 Torben Hagerup. Parallel preprocessing for path queries without concurrent reading. *Inf. Comput.*, 158(1):18–28, 2000. doi:10.1006/inco.1999.2814.
- 16 Meng He and Serikzhan Kazi. Path and Ancestor Queries over Trees with Multidimensional Weight Vectors. In *ISAAC*, pages 45:1–45:17, 2019.
- 17 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. doi:10.1145/2344422.2344432.
- 18 Meng He, J. Ian Munro, and Gelin Zhou. A framework for succinct labeled ordinal trees over large alphabets. *Algorithmica*, 70(4):696–717, 2014. doi:10.1007/s00453-014-9894-4.
- 19 Meng He, J. Ian Munro, and Gelin Zhou. Data structures for path queries. *ACM Trans. Algorithms*, 12(4):53:1–53:32, 2016. doi:10.1145/2905368.
- 20 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. doi:10.1007/978-3-540-30551-4\_49.
- 21 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.
- 22 Christos Makris and Athanasios K. Tsakalidis. Algorithms for three-dimensional dominance searching in linear space. *Inf. Process. Lett.*, 66(6):277–283, 1998. doi:10.1016/S0020-0190(98)00075-1.
- 23 Yakov Nekrich. A data structure for multi-dimensional range reporting. In *SoCG*, pages 344–353, 2007. doi:10.1145/1247069.1247130.
- 24 Yakov Nekrich. New Data Structures for Orthogonal Range Reporting and Range Minima Queries, 2020. URL: <https://arxiv.org/abs/2007.11094>, doi:10.48550/ARXIV.2007.11094.
- 25 Yakov Nekrich. New Data Structures for Orthogonal Range Reporting and Range Minima Queries. In *SODA*, pages 1191–1205, 2021. doi:10.1137/1.9781611976465.73.
- 26 Yakov Nekrich and Gonzalo Navarro. Sorted range reporting. In *SWAT*, pages 271–282, 2012. doi:10.1007/978-3-642-31155-0\_24.
- 27 Manish Patil, Rahul Shah, and Sharma V. Thankachan. Succinct representations of weighted trees supporting path queries. *J. Discrete Algorithms*, 17:103–108, 2012. doi:10.1016/j.jda.2012.08.003.
- 28 Daniel Dominic Sleator and Robert Endre Tarjan. A data structure for dynamic trees. *J. Comput. Syst. Sci.*, 26(3):362–391, 1983. doi:10.1016/0022-0000(83)90006-5.
- 29 Dan E. Willard. Log-logarithmic worst-case range queries are possible in space  $\theta(n)$ . *Inf. Process. Lett.*, 17(2):81–84, 1983. doi:10.1016/0020-0190(83)90075-3.
- 30 Gelin Zhou. Two-dimensional range successor in optimal time and almost linear space. *Inf. Process. Lett.*, 116(2):171–174, 2016. doi:10.1016/j.ipl.2015.09.002.