Data Structures for Categorical Path Counting Queries

Meng He ✉
Faculty of Computer Science, Dalhousie University, Halifax, Canada

Serikzhan Kazi ✉
Faculty of Computer Science, Dalhousie University, Halifax, Canada

Abstract

Consider an ordinal tree T on n nodes, each of which is assigned a *category* from an alphabet $[\sigma] = \{1, 2, \dots, \sigma\}$. We preprocess the tree T in order to support *categorical path counting queries*, which ask for the number of distinct categories occurring on the path in T between two query nodes x and y. For this problem, we propose a linear-space data structure with query time $\mathcal{O}(\sqrt{n} \lg \frac{\lg \sigma}{\lg w})$, where $w = \Omega(\lg n)$ is the word size in the word-RAM. As shown in our proof, from the assumption that matrix multiplication cannot be solved in time faster than cubic (with only combinatorial methods), our result is optimal, save for polylogarithmic speed-ups. For a trade-off parameter $1 \le t \le n$, we propose an $\mathcal{O}(n + \frac{n^2}{t^2})$-word, $\mathcal{O}(t \lg \frac{\lg \sigma}{\lg w})$ query time data structure. We also consider c-approximate categorical path counting queries, which must return an approximation to the number of distinct categories occurring on the query path, by counting each such category at least once and at most c times. We describe a linear-space data structure that supports 2-approximate categorical path counting queries in $\mathcal{O}(\lg n / \lg \lg n)$ time.

Next, we generalize the categorical path counting queries to weighted trees. Here, a query specifies two nodes x, y and an orthogonal range Q. The answer to thus formed *categorical path range counting* query is the number of distinct categories occurring on the path from x to y, if only the nodes with weights falling inside Q are considered. We propose an $\mathcal{O}(n \lg \lg n + (n/t)^4)$-word data structure with $\mathcal{O}(t \lg \lg n)$ query time, or an $\mathcal{O}(n + (n/t)^4)$-word data structure with $\mathcal{O}(t \lg^\epsilon n)$ query time. For an appropriate choice of the trade-off parameter t, this implies a linear-space data structure with $\mathcal{O}(n^{3/4} \lg^\epsilon n)$ query time. We then extend the approach to the trees weighted with vectors from $[n]^d$, where d is a constant integer greater than or equal to 2. We present a data structure with $\mathcal{O}(n \lg^{d-1+\epsilon} n + (n/t)^{2d+2})$ words of space and $\mathcal{O}(t \frac{\lg^{d-1} n}{(\lg \lg n)^{d-2}})$ query time. For an $\mathcal{O}(n \cdot \operatorname{polylog} n)$-space solution, one thus has $\mathcal{O}(n^{\frac{2d+1}{2d+2}} \cdot \operatorname{polylog} n)$ query time.

The inherent difficulty revealed by the lower bound we proved motivated us to consider data structures based on *sketching*. In unweighted trees, we propose a sketching data structure to solve the approximate categorical path counting problem which asks for a $(1 \pm \epsilon)$-approximation (i.e. within $1 \pm \epsilon$ of the true answer) of the number of distinct categories on the given path, with probability $1 - \delta$, where $0 < \epsilon, \delta < 1$ are constants. The data structure occupies $\mathcal{O}(n + \frac{n}{t} \lg n)$ words of space, for the query time of $\mathcal{O}(t \lg n)$. For trees weighted with d-dimensional weight vectors ($d \ge 1$), we propose a data structure with $\mathcal{O}((n + \frac{n}{t} \lg n) \lg^d n)$ words of space and $\mathcal{O}(t \lg^{d+1} n)$ query time.

All these problems generalize the corresponding categorical range counting problems in Euclidean space $\mathbb{R}^{d+1}$, for respective d, by replacing one of the dimensions with a tree topology.

2012 ACM Subject Classification Theory of computation → Data structures design and analysis

Keywords and phrases data structures, weighted trees, path queries, categorical queries, coloured queries, categorical path counting, categorical path range counting

Digital Object Identifier 10.4230/LIPIcs.CPM.2021.15

1 Introduction

In orthogonal range searching, one preprocesses a given finite set $S \subset \mathbb{R}^d$ into a data structure so that the points inside an axis-aligned query (hyper-)rectangle can be efficiently searched. For example, orthogonal range counting asks for the number of points falling inside the query rectangle, whereas the orthogonal range reporting problem asks to enumerate all such points. We refer the reader to [28, 7, 1] and references therein for the state-of-the-art in the discipline.

In some applications, of the actual interest may be the number of distinct *types*, or *categories*, of points that fall within the query rectangle. Apart from uses in business intelligence (enshrined in SQL keywords `DISTINCT` and `GROUP BY`), these distinct values find uses in SQL query optimization [9], too. Categorical (also known as *coloured*) range searching is thus an area of active research in computer science [22, 2, 29, 30, 31, 21, 33, 19, 8, 6].

A few aspects render the categorical variants of range searching harder than their "plain" counterparts. First, there can be far fewer categories than points. Second, such problems are not easily decomposable – for two disjoint regions S_1 and S_2, knowing just the number of distinct categories in each of them is insufficient to infer the count for the union $S_1 \cup S_2$.

For all the progress in categorical *reporting* queries (where one enumerates the distinct categories in the query region) [8, 6], with results almost matching the state of the art in regular 2D reporting [7], efficient categorical *counting* remains elusive, with the currently best results of $\mathcal{O}(n^2 \lg^2 n)$ words and $\mathcal{O}(\lg^2 n)$ query time [22, 29], or $\mathcal{O}(n \lg^6 n)$ words and $\mathcal{O}(\sqrt{n} \lg^7 n)$ query time [29], versus the optimal linear-space and $\mathcal{O}(\frac{\lg n}{\lg \lg n})$-time data structure for 2D orthogonal range counting [28]. In the exact opposite to the "plain" case, the categorical version of range counting is deemed to be harder than its reporting counterpart [29], when $d \geq 2$.

Meanwhile, given the versatility of trees as a data organization tool, information retrieval from tree-structured hierarchies is set to gain in importance. Hence researchers considered the generalizations of orthogonal range searching, where one of the dimensions is replaced by a tree topology, whereas the remaining coordinates of the points become the *weights* of the nodes [23]. Such a weight can be either a scalar, which corresponds to generalizing a Euclidean 2D point-set to trees, or a *vector*, when extending from $\mathbb{R}^d$, $d \geq 3$. Therefore, a tree weighted with d-dimensional *weight vectors* generalizes a point-set from $\mathbb{R}^{d+1}$. (Note that when $d = 0$, an unweighted tree thus generalizes a 1D set.)

The generalization from point-sets to trees gives rise to *path queries*, which ask a question about the nodes on the query path, whose weights fall inside the query rectangle; for example, a path counting query asks only for the number of such nodes, whereas the reporting variant asks to enumerate them [34, 26]. Research on path queries has spawned a wide range of metrics, such as range quantiles [26], minimum/maximum [5], mode/minority [12], and (α-)majority/minority [17].

Analogously to the Euclidean scenario, the qualitative side of the relation between node-entities is best captured in categorical variants of path queries. For example, let us annotate a phylogenetic tree for a set of genomes by marking each divergence with a *type* of mutation. The number of distinct mutation event types between two given species then could serve as a proxy for evolutionary "distance" between them. A *categorical path counting query*, which asks for the number of distinct categories on a query path, provides an adequate model in this case.

Generalizing the 1D categorical reporting problem, Durocher et al. [12] solved the *top-k colour reporting problem* on unweighted trees. We believe that in trees, neither the counting problem in the categorical setting, nor the scenario of weighted nodes has been studied before. In this paper, we formalize these problems and propose solutions to them.

We consider an ordinal tree[1] T on n nodes, such that each node z of T is associated with a category $\mathtt{c}(z) \in [\sigma]$. [2] Specified at query time is a query path $P_{x,y}$ between two nodes x, y in T. We are to preprocess T into a data structure to compute in an efficient manner the number $n_{real} = |\{\mathtt{c}(z) \mid z \in P_{x,y}\}|$. This is the **categorical path counting problem** studied in this paper.

Next, for $d \geq 1$, we consider a tree T in which each node, along with a category, is associated with a certain weight vector $\mathbf{w}(z) \in [n]^d$. In addition to a query path $P_{x,y}$, at query time specified also is an axis-aligned (hyper-)rectangle Q from $[n]^d$. We are to preprocess T into a data structure to compute in an efficient manner the number $n_{real} = |\{\mathtt{c}(z) \mid z \in P_{x,y} \wedge \mathbf{w}(z) \in Q\}|$. This is the **categorical path range counting problem** studied in the present paper.

For both problems, a *c-approximate* (for $c > 1$) answer is a number n_{appr} such that $n_{real} \leq n_{appr} \leq c \cdot n_{real}$. A $(1 \pm \epsilon)$-approximate (for $0 < \epsilon < 1$) answer is a number n_{appr} such that $\frac{|n_{appr} - n_{real}|}{n_{real}} \leq \epsilon$.

1.1 Previous Work

For points on a line, Gagie and Kärkkäinen [18] have proposed an $\mathcal{O}(n)$-word solution to the 1D categorical counting problem, with query time $\mathcal{O}(\lg^{1+\epsilon} n)$, [3] for any $\epsilon > 0$. Nekrich [33] proposed another $\mathcal{O}(n)$-space solution with query time $\mathcal{O}(\frac{\lg \sigma}{\lg \lg n})$, where σ is the number of categories.

Grossi and Vind [21] solve the 2D categorical range counting problem in linear space and $o(n)$ time, and higher-dimensional variants in almost-linear space and $o(n)$ time. The core idea is to divide the universe of categories into chunks of size $\lg n$, and use bitwise-`OR` when querying the restriction of the input set to each such chunk. The best result with polylogarithmic time in the 2D categorical counting problem remains at $\mathcal{O}(n^2 \lg^2 n)$ words and $\mathcal{O}(\lg^2 n)$ query time [22, 29], with [29] also proposing an $\mathcal{O}(X \lg^7 n)$ query-time data structure with $\mathcal{O}((\frac{n}{X})^2 \lg^6 n + n \lg^4 n)$ storage space, for a trade-off parameter $1 \leq X \leq n$; for $X = \sqrt{n}$, the space is thus $\mathcal{O}(n \lg^6 n)$ and the query time is $\mathcal{O}(\sqrt{n} \lg^7 n)$. Whereas Gupta et al. [22] use persistence, Kaplan et al. [29] proceed by a disjoint decomposition of the region covering an individual category, with the subsequent reduction to rectangle-stabbing. In higher dimensions ($d > 2$), [29] proposed an $\mathcal{O}(n^d \lg^{2d-2} n)$-word data structure with $\mathcal{O}(\lg^{2d-2} n)$ query time. They also show that an algorithm for categorical range counting in $\mathbb{R}^2$ that answers m queries over the set of $\mathcal{O}(n)$ points in $o(\min\{n,m\}^{\omega/2})$ time would yield an algorithm for obtaining the matrix product $MM^{\intercal}$ in $o(k^{\omega/2})$ time, for any $k \times k$ matrix M over $\{0,1\}$, where ω is the best current exponent for Boolean matrix multiplication. Further, Kaplan et al. [29] proposed an $\mathcal{O}((\frac{n}{X})^{2d} + n \lg^{d-1} n)$-word data structure with $\mathcal{O}(X \lg^{d-1} n)$ query time, for a trade-off parameter $1 \leq X \leq n$. This implies an $\widetilde{\mathcal{O}}(n)$-space[4] data structure with $\widetilde{\mathcal{O}}(n^{\frac{2d-1}{2d}})$ query time.

Nekrich [33] proposed an $\mathcal{O}(n(\lg \lg n)^2)$-word data structure for $(4+\epsilon)$-approximate 2D categorical counting in $\mathcal{O}((\lg \lg n)^2)$ time; this translates to a linear-space data structure for an $n \times n$ grid, that returns in $\mathcal{O}(1)$ time a $(1+\epsilon)$-approximation for the number of points in a 3-sided 2D query range, for a constant $0 < \epsilon < 1$. El-Zein et al. [13] solved the approximate

[1] i.e. a tree in which the children of a node are ordered
[2] We set $[n] \triangleq \{1, 2, \ldots, n\}$ for any $n \in \mathbb{N}$.
[3] We use $\lg n \triangleq \log_2 n$, and explicitly specify the base otherwise.
[4] Notation $\widetilde{\mathcal{O}}$ leaves out polylogarithmic factors.

categorical range counting problem in 1D in succinct $\mathcal{O}(n)$ bits of space and $\mathcal{O}(1)$ time. The core technique is to sample the prefixes of an array with exponentially increasing number of distinct categories covered, and "sandwich" the query point between two sampled values using transdichotomous data structures [14].

Lai et al. [30] used sketching data structures [10] to solve the approximate categorical range counting problem in a probabilistic setting. In d dimensions, they proposed an $\mathcal{O}(dn \lg^{d-1} n)$-words-of-space data structure, to support queries in $\mathcal{O}(d \lg^{d+1} n)$ time, with probability $1 - \delta$, where $0 < \delta < 1$ is a given constant. Sketches approximate the number of distinct categories occurring in a collection; being small and additive, in the solution of Lai et al. they serve as summary structures.

To the best of our knowledge, the only categorical range searching problem considered so far for tree topologies is the *top-k colour reporting* problem. Therein, the categories have priorities, and the k highest-priority categories occurring on the given path are to be reported. Durocher et al. [11] introduce and solve this problem in (optimal) $\mathcal{O}(n)$ space and $\mathcal{O}(1 + k)$ time. They use heavy-path decomposition and chaining [32] to reduce the problem to 2D reporting in a narrow grid.

1.2 Our Contribution

For the categorical path counting problem, we propose a linear-space data structure with query time $\mathcal{O}(\sqrt{n} \lg \frac{\lg \sigma}{\lg w})$, where w is the word size on the word-RAM model. We show, by a reduction from Boolean matrix multiplication, that the query time is optimal within polylogarithmic factors, with current knowledge and when only combinatorial methods are allowed. This conditional lower bound is surprising, because the 1D counterpart in the Euclidean case admits a linear-space solution with a sub-logarithmic query time, and a similar conditional lower bound can only be proven in 2D. In other words, having a tree structure in the presence of categories is about as hard as having a second dimension, making the query time go up from polylogarithmic to polynomial, when the space usage is linear. This however is not the case in the previous work on path queries [12, 26, 23]. Specifically, for a trade-off parameter $1 \leq t \leq n$, we propose an $\mathcal{O}(n + \frac{n^2}{t^2})$-word, $\mathcal{O}(t \lg \frac{\lg \sigma}{\lg w})$ query time data structure (which corresponds to a linear-space data structure with $\mathcal{O}(\sqrt{n} \lg \frac{\lg \sigma}{\lg w})$ query time). We also describe a linear-space data structure that supports 2-approximate categorical path counting queries in $\mathcal{O}(\frac{\lg n}{\lg \lg n})$ time. These problems have not been considered in trees before.

We also generalize the categorical path counting queries to weighted trees. For $d = 1$, we propose an $\mathcal{O}(n \lg \lg n + (n/t)^4)$-word data structure with $\mathcal{O}(t \lg \lg n)$ query time, or an $\mathcal{O}(n + (n/t)^4)$-word data structure with $\mathcal{O}(t \lg^{\epsilon} n)$ query time. This implies a linear-space data structure with $\mathcal{O}(n^{3/4} \lg^{\epsilon} n)$ query time. The corresponding $\mathcal{O}(n \lg^6 n)$-word solution to categorical range counting in $\mathbb{R}^2$ by [29] achieves $\mathcal{O}(\sqrt{n} \lg^7 n)$ query time. Compared to the best result in the Euclidean counterpart, we thus sacrifice an $\widetilde{\mathcal{O}}(\sqrt[4]{n})$-factor in query time, to accommodate the tree structure.

We further extend the approach to the trees weighted with multidimensional vectors from $[n]^d$, where d is a constant integer greater than or equal to 2. We describe an $\mathcal{O}(n \lg^{d-1+\epsilon} n + (n/t)^{2d+2})$-word data structure with $\mathcal{O}(t \frac{\lg^{d-1} n}{(\lg \lg n)^{d-2}})$ query time. For an $\widetilde{\mathcal{O}}(n)$-space solution, this yields $\widetilde{\mathcal{O}}(n^{\frac{2d+1}{2d+2}})$ query time. When $d \geq 2$, this result matches the best corresponding result in $\mathbb{R}^{d+1}$ by Kaplan et al. [29], within polylogarithmic factors.

Our sketching data structure for unweighted trees solves the approximate categorical path counting problem, which asks for a $(1 \pm \epsilon)$-approximation for the number of distinct categories on the given path, with probability $1 - \delta$. The data structure occupies $\mathcal{O}(n + \frac{n}{t} \lg n)$

words of space, for the query time of $\mathcal{O}(t \lg n)$. For trees weighted with d-dimensional weight vectors ($d \geq 1$), we propose an $\mathcal{O}((n + \frac{n}{t} \lg n) \lg^d n)$-word data structure with $\mathcal{O}(t \lg^{d+1} n)$ query time. Here, $0 < \epsilon, \delta < 1$ are arbitrarily small constants.

2 Preliminaries

In this section we introduce the notation and give background on the concepts used in the paper.

2.1 Concepts and Notation

We denote by $|T|$ the size (i.e. the number of nodes) of the tree T, whose set of nodes is denoted as $V(T)$. For $x, y \in V(T)$, the path between x and y is denoted as $P_{x,y}$. For brevity, if no confusion ensues, we write $x \in T$ to denote $x \in V(T)$. We write $P_{x,y} \subseteq T$ to indicate that a path belongs to a tree. We denote the root of T by $\perp$; thus $P_{x,\perp}$ is the root-to-x path. In all our input trees, each $x \in T$ has a certain *category* $\mathtt{c}(x) \in [\sigma]$ associated with it. In addition, each $x \in T$ can be associated with a *weight* $\mathbf{w}(x) \in [n]$. In general, $\mathbf{w}(x)$ can be a *weight vector* drawn from $[n]^d$, for $d \geq 1$, and the i^{th} component of the weight vector is the i^{th} *weight.* In line with the current trends in orthogonal range searching, we assume the weights to be in the rank space [16]. For brevity, we shall also use *Iverson notation* [20]: For a Boolean predicate P, the symbol $[\![P]\!] \in \{0,1\}$ equals 1 *iff* $P = \mathtt{true}$. A sequence of objects $I_1, I_2, \ldots, I_k$ is denoted as $\{I_j\}_{j=1}^{k}$. Finally, unless otherwise indicated, w denotes the word size in the word-RAM machine; one typically has $w = \Omega(\lg n)$.

2.2 Compact Representation of Ordinal Labeled Trees

Fast navigation in compactly-represented ordinal labeled trees and *tree extractions* are central to our solutions. In this section we review the pertinent results.

▶ **Lemma 1** (He et al. [24]). *Let T be an ordinal tree on n nodes. Then, T can be represented in $2n + o(n)$ bits of space, to support the following operations in $\mathcal{O}(1)$ time, for any $x, y \in T$: (a) $\mathtt{depth}(x)$ the number of ancestors of x; and (b) $\mathtt{level\_anc}(x, i)$ the i^{th} nearest ancestor of x ($\mathtt{level\_anc}(x, 1)$ being x itself); and (c) $\mathtt{LCA}(x, y)$ the lowest common ancestor of x and y.*

When the tree T is labeled over $[\sigma]$, with $\mathtt{label}(z)$ denoting the label assigned to $z \in T$, the common operators can be sub-scripted. Indeed, let a node (resp. ancestor) labeled α be referred to as an *α-node* (resp. *α-ancestor*). The following result is our main tool in navigating labeled trees:

▶ **Lemma 2** (He et al. [25]). *Let T be an ordinal tree on n nodes, each of which is assigned a label over $[\sigma]$, $\sigma \leq n$. Then, under the word-RAM with word size $w = \Omega(\lg n)$, T can be represented using $\mathcal{O}(n)$ words of space to support the following operations in $\mathcal{O}(\lg \frac{\lg \sigma}{\lg w})$ time, for any $x, y \in T$ and any $\alpha \in [\sigma]$: (a) $\mathtt{depth}_\alpha(x)$ the number of α-nodes on $P_{x,\perp}$; and (b) $\mathtt{level\_anc}_\alpha(x, i)$ the i^{th} nearest α-ancestor of x ($\mathtt{level\_anc}(x, 1) = x$ if x is an α-node); and (c) $\mathtt{pre\_rank}_\alpha(x)$ the number of α-nodes preceding x in preorder; and (d) $\mathtt{pre\_select}_\alpha(j)$ the j^{th} α-node in preorder.*

Labeled versions of the common operators serve to restrict the queries to the given labels only. For example, the number $\mathtt{depth}_\alpha(x) + \mathtt{depth}_\alpha(y) - 2 \cdot \mathtt{depth}_\alpha(z) + [\![\mathtt{label}(z) = \alpha]\!]$, where $z = \mathtt{LCA}(x, y)$, equals the number of α-labeled nodes on the given path $P_{x,y}$.

2.3 Tree Extraction

Tree extraction [26] selects a subset of nodes while preserving the relative preorder ranks, as well as the hierarchical relations among the nodes. Precisely, given a subset $X \subseteq V$ of nodes (X is called the *extracted nodes*), the *extracted tree* T_X is constructed from T via the following *edit operations*. Fix an arbitrary node $y \notin X$, and let $p \in T$ be the parent of y. Let y be the i^{th} child of p, in preorder. Let us erase, from T, the node y together with its incident edges. This frees the i^{th} slot in the list of children of p, as well as the k children $y_1, y_2, \ldots, y_k$ of the node y. Then, y_1 becomes the i^{th} child of p, y_2 becomes its $(i+1)^{\text{st}}$ child, and so on, until y_k becomes p's $(i+k-1)^{\text{st}}$ child. The node that was the $(i+1)^{\text{st}}$ child of p prior to deletion becomes the $(i+k)^{\text{th}}$ child of p, i.e. all the initial children occurring after the i^{th} are shifted to k positions to the right. After erasing all the nodes $y \notin X$ in the described way, the resulting forest F_X is either a tree (in which case we do nothing), or a forest, in which case we create a dummy root r (with preorder rank and depth set to 0) that becomes the parent of all the roots of the trees in F_X, again preserving the relative preorder ranks of the roots.

2.4 Semigroup Path Sum Query Problem

Trees with nodes associated with semigroup elements give rise to *semigroup path sum* problems:

▶ **Definition 3.** *Let us be given a semigroup $(G, \oplus)$ with the sum operator denoted as $\oplus$, and the set of the semigroup's elements denoted as G. Furthermore, let T be an ordinal tree on n nodes, each node x of which is assigned a d-dimensional weight vector $\mathbf{w}(x)$, as well as a semigroup element $g(x)$. Then, in a d-*dimensional semigroup path sum problem*, one is given a query path $P_{x,y} \subseteq T$, a query range Q in d-dimensional space, and is asked to evaluate* $\bigoplus_{z \in P_{x,y} \wedge \mathbf{w}(z) \in Q} g(z)$.

The framework of [23] can be used to extend a solution to the multidimensional semigroup path sum query problem in the sense of Definition 3 from $(d-1)$ to d dimensions (here, the *size* of a problem refers to the corresponding $|T|$):

▶ **Lemma 4** (Lemma 5 in [23]). *Let d be a positive integer constant. Let $G^{(d-1)}$ be an $\mathtt{s}(n)$-word data structure for a $(d-1)$-dimensional semigroup path sum problem of size n. Then, there is an $\mathcal{O}(\mathtt{s}(n) \lg n + n)$-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.*

3 Categorical Path Counting

In this section, we consider the categorical path counting problem in the exact and approximate formulations. First, in Section 3.1 we prove a conditional lower bound on the categorical path counting problem in unweighted trees. Then, Section 3.2 offers some background on the techniques used to solve the categorical path counting problem. Further, we design a data structure that matches the lower bound within polylogarithmic factors when using only combinatorial approaches (Section 3.3). We conclude by designing a 2-approximate solution with a much faster query time (Section 3.4).

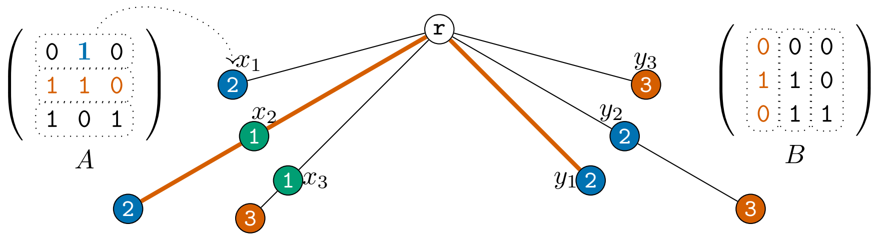

Figure 1 Two matrices A and B each of size $\sqrt{n} \times \sqrt{n}$ with $n = 9$ give rise to a tree over $\sqrt{n}+1$ categories. The dummy root is the node marked r, and the numbers inside circles, as well as the distinct colours, denote the category of the corresponding node. The path shown in thick coloured line corresponds to a path queried when computing the cell $(2,1)$ of the product $A \times B$. This entry corresponds to the product of the second row and the first column, respectively of the matrices A and B (which are also coloured).

3.1 Hardness of Categorical Path Counting

In this section we show a reduction from the Boolean matrix multiplication problem to the categorical path counting problem over unweighted trees. Namely, we prove

▶ **Theorem 5.** *Let $p(n)$ (for $n \in \mathbb{N}$) be the preprocessing time of a categorical path counting data structure and $q(n)$ its query time, over an ordinal tree T on n nodes, each of which is assigned a category over a finite alphabet. Then Boolean matrix multiplication on two $\sqrt{n} \times \sqrt{n}$ matrices can be solved in $\mathcal{O}(p(n) + nq(n) + n)$ time.*

Proof. Let A and B be two $\sqrt{n} \times \sqrt{n}$ Boolean matrices, and we are to compute the product $C = A \times B$. Let $a_{i,j}, b_{i,j}$ and $c_{i,j}$ be the elements in row i and column j of the matrices A, B and C, respectively. For the i^{th} row of A we construct the set $A_i = \{j \mid a_{i,j} = 1\}$, and for the j^{th} column of B we construct the set $B_j = \{i \mid b_{i,j} = 1\}$. We notice that $c_{i,j} = [\![A_i \cap B_j \neq \emptyset]\!]$.

As $|A_i \cap B_j| = |A_i| + |B_j| - |A_i \cup B_j|$, it is sufficient to focus on computing $|A_i \cup B_j|$, which in turn motivates the following construction of a tree T of size $\mathcal{O}(n)$:

1. We create a dummy root r with dummy category $\sqrt{n}+1$;
2. The root r has $2\sqrt{n}$ children $x_1, x_2, \ldots, x_{\sqrt{n}}$ and $y_1, y_2, \ldots, y_{\sqrt{n}}$;
3. The subtree rooted at each x_i, $1 \le i \le \sqrt{n}$, is a single path of length $m_i = |A_i|$, consisting of nodes $x_{i,1}, x_{i,2}, \ldots, x_{i,m_i}$, listed in preorder, i.e. with $x_i = x_{i,1}$ and x_{i,m_i} being the leaf;
4. The subtree rooted at each y_j, $1 \le j \le \sqrt{n}$, is a single path of length $n_j = |B_j|$, consisting of nodes $y_{j,1}, y_{j,2}, \ldots, y_{j,n_j}$, listed in preorder, i.e. with $y_j = y_{i,1}$ and y_{j,n_j} being the leaf;
5. $\forall\, 1 \le i \le \sqrt{n}$ and $\forall\, 1 \le j \le m_i$, the node $x_{i,j}$ is assigned a category – the rank-j entry of A_i;
6. $\forall\, 1 \le j \le \sqrt{n}$ and $\forall\, 1 \le i \le n_j$, the node $y_{j,i}$ is assigned a category – the rank-i entry of B_j.

Thus T is a tree of size $\mathcal{O}(n)$, in which each node is assigned a category from $[\sqrt{n}+1]$. (See Figure 1 for an example of the tree constructed for two matrices A and B.) Now, it is clear that computing $|A_i \cup B_j|$ is nothing but a categorical path query with query parameters x_{i,m_i} and y_{j,n_j} (subtracting 1 from the result, to correct for the root r). Processing $\sqrt{n} \times \sqrt{n}$ queries each in time $q(n)$, the claimed time bound follows. ◀

The best algebraic methods of multiplying two $t \times t$ Boolean matrices are known to have complexity $\mathcal{O}(t^{\omega})$ with $\omega < 2.3727$ (Williams [35]). Two $\sqrt{n} \times \sqrt{n}$ matrices can therefore be multiplied in $\mathcal{O}(n^{\omega/2})$ time. This means that, with current knowledge, one can not have preprocessing time $p(n)$ better than $\mathcal{O}(n^{\omega/2})$ and query time $q(n)$ better than $\mathcal{O}(n^{\omega/2-1})$ simultaneously, i.e. it must be either that $p(n)$ is $\Omega(n^{1.18635})$ or $q(n)$ is $\Omega(n^{0.18635})$.

The best known combinatorial algorithm for multiplying two $n \times n$ Boolean matrices is only polylogarithmically better than cubic [3, 4, 36]. Theorem 5 therefore implies that preprocessing time $p(n)$ can not be better than $n^{3/2}$ and query time $q(n)$ can not be better than $\sqrt{n}$ at the same time, by purely combinatorial methods with current knowledge, save for polylogarithmic speed-ups.

3.2 Uniform Partitioning of the Tree

Next, we review a tree mark-up technique that we use in our solutions in Sections 3.3 and 4.2.

▶ **Lemma 6** (Lemma 9 in [12]). *Given an ordinal tree T on n nodes and an integer $1 \leq t \leq n$ which is called the* blocking factor*, a subset $V' \subseteq V(T)$ of the nodes, called the* marked *nodes, can be selected in $\mathcal{O}(n)$ time so that: (i) $|V'| = \mathcal{O}(n/t)$; (ii) for any $x, y \in V'$ it follows that* $\texttt{LCA}(x, y) \in V'$; *and (iii) a path containing unmarked nodes only consists of less than t nodes and the edges between them.*

As Durocher et al. [12] only described which nodes should be marked without showing how to mark them in $\mathcal{O}(n)$ time, we propose a linear-time algorithm to mark these nodes, which is presented in the full version of the paper.

Path decomposition using the marked nodes (or, generally, nodes with certain labels) is encapsulated in a `decompose`-operator of Definition 7. Lemma 8 implements `decompose`, as a simple corollary to Lemma 2. In Definition 7 and Lemma 8, T is an ordinal tree on n nodes, each of which is assigned a label over an alphabet $[\sigma]$, where $\sigma \leq n$.

▶ **Definition 7.** *For any pair of nodes x, y of T, for any $\alpha \in [\sigma]$, consider the closest α-nodes $x', y' \in P_{x,y}$, to respectively x and y. Then, the operation* $\texttt{decompose}(x, y, \alpha)$ *returns the pair of nodes x' and y', or a special symbol* ***`undefined`*** *when no such x' and y' exist.*

▶ **Lemma 8.** *The tree T represented via Lemma 2 supports* $\texttt{decompose}(x, y, \alpha)$ *in $\mathcal{O}(\lg \frac{\lg \sigma}{\lg w})$ time.*

Proof. Let us preprocess the input tree T using Lemma 2. One then has the following cases and the corresponding courses of action:

Case 1 If $\texttt{LCA}(x, y) = y$, we have $x' = \texttt{level\_anc}_\alpha(x, 1)$. Node y' is set to be y itself if y is an α-node; otherwise, $y' = \texttt{level\_anc}_\alpha(x, a)$, where $a = \texttt{depth}_\alpha(x) - \texttt{depth}_\alpha(y)$. The result is *undefined* if y is not an α-node and $a = 0$. The case when x is the ancestor of y is symmetrical.

Case 2 If x and y are not ancestors of each other, we set $z = \texttt{LCA}(x, y)$. There are four sub-cases, depending on the values $a = \texttt{depth}_\alpha(x) - \texttt{depth}_\alpha(z)$ and $b = \texttt{depth}_\alpha(y) - \texttt{depth}_\alpha(z)$.

$\boldsymbol{a > 0,\ b > 0}$: One has $x' = \texttt{level\_anc}_\alpha(x, 1), y' = \texttt{level\_anc}_\alpha(y, 1)$;

$\boldsymbol{a > 0,\ b = 0}$: This case is reduced to **Case 1** by setting $y := z$;

$\boldsymbol{a = 0,\ b > 0}$: This case is reduced to **Case 1** by setting $x := z$;

$\boldsymbol{a = 0,\ b = 0}$: The result is *undefined* if z is not an α-node, and $x' = y' = z$, otherwise;

With $\mathcal{O}(1)$ amount of $\mathcal{O}(\lg \frac{\lg \sigma}{\lg w})$-time operations, the claimed running time follows. ◀

From the properties of tree extraction and Lemmas 2 and 8 it follows that

▶ **Proposition 9.** *In the conditions of Lemma 8, let $P_{x,y} \subseteq T$ be an arbitrary path and $\alpha \in [\sigma]$ an arbitrary label. Let T_α be a tree extraction from T of the node-set $X = \{z \in V(T) \mid \texttt{label}(z) = \alpha\}$. Let x' and y' be the nodes returned by* $\texttt{decompose}(x, y, \alpha)$. *Then, all the nodes in $P_{x,y} \cap X$ form a contiguous path π in T_α, with end-points $x_\alpha = 1 + \texttt{pre\_rank}_\alpha(x')$ and $y_\alpha = 1 + \texttt{pre\_rank}_\alpha(y')$. Furthermore,* $\texttt{decompose}(x, y, \alpha)$ *returns* ***`undefined`*** *iff $P_{x,y} \cap X = \emptyset$.*

3.3 Categorical Path Counting in Unweighted Trees

In this section, we solve the exact categorical path counting problem. We do so by precomputing certain information, with additional work at query time. Hence the storage space and the explicit query-time work are balanced by a trade-off parameter.

Namely, the tree T is subject to the following preprocessing:

Nodes marking For the parameter $t \leq n$ to be chosen later, we mark $\mathcal{O}(n/t)$ nodes in T using Lemma 6. Let K be a copy of T labeled over $\{0,1\}$ in such a way that a node $z \in V(K)$ has label 1 *iff* its copy in T is marked. We preprocess K via Lemma 2 thereby enabling the use of Lemma 8;

Path emptiness Let G be a copy of T labeled over $[\sigma]$ in such a way that the node $z \in V(G)$ has label α *iff* its copy in T has category α. We preprocess G via Lemma 2 thereby enabling the use of Lemma 8;

Tabulation We store a table M such that, for the x^{th} and y^{th} (in preorder) marked node of T, one has $M[x,y] \triangleq |\{\mathtt{c}(z) \,|\, z \in P_{x',y'}\}|$ (i.e. $M[x,y]$ is the number of distinct categories occurring on the path – the *span* – $P_{x',y'} \subseteq T$). Here, x' and y' are found via Proposition 9 as the corresponding nodes to respectively x and y.

The data structures built in Section 3.3 result in the following

▶ **Theorem 10.** *Let T be an ordinal tree on n nodes, each of which is assigned a category over an alphabet $[\sigma]$, where $\sigma \leq n$. Then, T can be preprocessed into a data structure of size $\mathcal{O}(n + \frac{n^2}{t^2})$, for a given $1 \leq t \leq n$, so that a categorical path counting query is answered in $\mathcal{O}(t \lg \frac{\lg \sigma}{\lg w})$ time. The preprocessing time is $\mathcal{O}(\frac{n^2}{t} \lg \frac{\lg \sigma}{\lg w})$. In particular, setting $t = \sqrt{n}$ yields a linear-space data structure with $\mathcal{O}(\sqrt{n} \lg \frac{\lg \sigma}{\lg w})$ query time, and $\mathcal{O}(n^{3/2} \lg \frac{\lg \sigma}{\lg w})$ preprocessing time.*

Proof. We preprocess the input tree T as described in Section 3.3. The structures K, G and M contribute respectively $\mathcal{O}(n), \mathcal{O}(n)$ and $\mathcal{O}(n^2/t^2)$ words, and hence the claimed space.

We thus turn to answering queries and analyzing the query time. As answering the query when $|P_{x,y}| \leq t$ is subsumed in our analysis, we let $|P_{x,y}| > t$.

A call to $\mathtt{decompose}(x, y, 1)$ on K returns two nodes x' and y' such that $|P_{x,x'}|, |P_{y,y'}| \leq t$, and x', y' are marked.

Let x_M and y_M respectively be the relative preorder ranks of x' and y' among the marked nodes of T; one computes x_M and y_M using Proposition 9. We use x_M and y_M to address the table M.

The answer to our query is contained in the following sets of nodes: **Group 0 :** $P_{x',y'}$ (the *span*); **Group 1 :** $P_{x,x'} \setminus \{x'\}$; and **Group 2 :** $P_{y,y'} \setminus \{y'\}$. We note that Groups 1-2 are each of size at most t.

The strategy is to process each group sequentially, so that a category contributes to the answer as long as it appears neither in the groups one has so far traversed, nor in the portion of the path preceding the current node, in the current group.

Namely, the processing of Group 0 reduces to initializing the result counter `res` with $M[x_M, y_M]$. Next, one traverses Group 1 in the direction towards x. Let q be the current node, and p be the node immediately preceding q, on the current path $P_{x,x'}$ consistent with the direction of traversal. We check whether $\mathtt{c}(q)$ occurs in $P_{p,y'}$ using the data structure G and Proposition 9; if not, we increment `res`. Finally, we traverse Group 2 in the direction towards y. Let q be the current node, and p be the node immediately preceding q, on the current path $P_{y,y'}$ and in the direction of traversal. We check whether $\mathtt{c}(q)$ occurs in $P_{x,p}$ using the data structure G and Proposition 9; if it does not, we increment `res`.

We call the operations in Lemmas 2 and 8 $\mathcal{O}(t)$ times; the claimed query time bound follows.

To analyze the preprocessing time, consider the list $\mathcal{L}$ of all the pairs (u, v), $u < v$ of marked nodes, ordered non-decreasingly by $|P_{u,v}|$. The list $\mathcal{L}$ has length $\mathcal{O}(\frac{n^2}{t^2})$ and can be ordered in $\mathcal{O}(\frac{n^2}{t^2})$ time using e.g. counting sort (because the values $|P_{u,v}|$ are non-negative integers that are at most n). We compute the entries of the table M traversing $\mathcal{L}$ from left to right. For the given pair $(u, v) \in \mathcal{L}$, it is either (i) $|P_{u,v}| \leq t$, or (ii) the operation $\texttt{decompose}(u'', v'')$, called on the nodes $u'' \in P_{u,v}$ and $v'' \in P_{u,v}$ respectively closest to u and v, returns two marked nodes u' and v' with the properties claimed in Definition 7. If (i) holds, one explicitly traverses the path $P_{u,v}$ in time $\mathcal{O}(t \lg \frac{\lg \sigma}{\lg w})$, as previously described. In case (ii), one has $|P_{u',v'}| < |P_{u,v}|$ and the answer for the span $P_{u',v'}$ is available; hence, one again uses the algorithm described in the beginning of this proof. ◀

Under the assumption that matrix multiplication cannot be solved faster than cubic time [4, 36], the bounds given in Theorem 10 are optimal, save for polylogarithmic speed-ups.

3.4 2-Approximate Categorical Path Counting

We provide a 2-approximation for the number of distinct categories on $P_{x,y}$ by decomposing the path $P_{x,y}$ as $P_{x,z}$ followed by $P_{y,z}$, with $z = \texttt{LCA}(x, y)$, and computing the answers in $P_{x,z}$ and $P_{y,z}$ separately. It turns out that in contrast to general paths, a query path in which one end is an ancestor of the other lends itself to an efficient categorical counting.

We apply the *chaining* approach [32], by assigning weights to the nodes of T as follows. If for $q \in T$ one has $\texttt{c}(q) = \gamma$, then we identify q's lowest proper γ-ancestor p and set $\mathbf{w}(q) = \texttt{depth}(p)$. We set $\mathbf{w}(q) = -1$, if there is no such p. It can be seen that counting the number of distinct categories on $P_{p,q}$ is equivalent to counting the number of nodes on $P_{p,q}$ with weights in the range $(-\infty, \texttt{depth}(p))$. We use the result of He et al. [26] to encode, in a structure C, the weighted tree and support *path counting queries* (for a path $P_{x,y}$ and a range Q, a path counting query returns the number $|\{z \in P_{x,y} \mid \mathbf{w}(z) \in Q\}|$):

▶ **Lemma 11** (He et al. [26]). *Let T be an ordinal tree on n nodes, each having a weight drawn from $[m]$. Under the word-RAM model, T can be encoded in $\mathcal{O}(n)$ words to support path counting queries in $\mathcal{O}(\frac{\lg m}{\lg \lg n} + 1)$ time.*

For the data structures built in Section 3.4, one thus has

▶ **Theorem 12.** *An ordinal tree T on n nodes, each of which assigned a category, can be preprocessed into an $\mathcal{O}(n)$-word data structure to solve the 2-approximate categorical path counting problem in $\mathcal{O}(\frac{\lg n}{\lg \lg n})$ time. When one query node is an ancestor another, the answer is exact.*

Proof. The input tree T is preprocessed as described in Section 3.4.

The dominant-size data structure C is linear in size (Lemma 11), hence the claimed space.

We focus on answering the query on the path $P_{x,z}$, where $z = \texttt{LCA}(x, y)$, for the query nodes x and y. Given the query $P_{x,z}$, we execute a path counting query in C with arguments $P_{x,z}$ and $(-\infty, \texttt{depth}(z))$. After the verbatim procedure for $P_{y,z}$, we return the sum of (the answers to) the two queries as the sought 2-approximation. We note that when y is an ancestor of x, the answer is exact.

The total running time is dominated by at most two path counting queries; the claimed query time bound follows. ◀

4 Categorical Path Range Counting

In this section, we solve the categorical path counting problem in the case of weighted trees, including those weighted with multidimensional weight vectors. We assume that the number, d, of dimensions is a constant.

In solving the categorical path range counting problem, we still apply the marking technique of Lemma 6. The core idea remains, but we guard against over-counting using somewhat more complex data structures. Namely, in Section 4.1 we extend the repertoire of useful tree operations (Section 2.2) by a *path range emptiness* query, which, in the case of unweighted trees (Section 3.3), was simulated using labeled ancestors and labeled depths (Lemma 2).

4.1 Path Range Emptiness Queries

First, let us formally introduce path range emptiness queries:

▶ **Definition 13.** *For a constant $d \in \mathbb{N}$, let T be an ordinal tree on n nodes, each node z of which is assigned a weight vector $\mathbf{w}(z) \in [n]^d$. For any two nodes $x, y \in T$ and any axis-aligned hyper-rectangle Q from $[n]^d$, a* path range emptiness *query is a path query that returns* `false` *if the set $\{z \in T \mid z \in P_{x,y} \wedge \mathbf{w}(z) \in Q\}$ is empty, and* `true`*, otherwise.*

It follows from the solutions to the path reporting problem of [5] [5] and Lemma 4 that

▶ **Lemma 14** ([5, 23])**.** *Let T be an ordinal tree on n nodes, each of which is assigned a weight vector from $[n]^d$. Then, T can be preprocessed into a data structure so that a path emptiness query is answered in*

$d = 1$: *either (a) $\mathcal{O}(\lg\lg n)$; or (b) in $\mathcal{O}(\lg^{\epsilon} n)$ time. The data structures occupy respectively (a) $\mathcal{O}(n \lg\lg n)$; and (b) $\mathcal{O}(n)$ words of space.*

$d \geq 2$: *$\mathcal{O}(\frac{\lg^{d-1} n}{(\lg\lg n)^{d-2}})$ time, for an $\mathcal{O}(n \lg^{d-1+\epsilon} n)$-word data structure.*

Lemma 14 presents different trade-offs to be used in our solutions for different values of d. For brevity, we shall refer to the query time as $\tau_d(n)$ and to the space cost as $\mathtt{s}_d(n)$.

4.2 Categorical Path Range Counting in d Dimensions

As in Section 3, here, too, we trade off explicit traversals for the storage for precomputed information. There are a few notable differences to accommodate weights. Precisely, the tree T is preprocessed as follows:

Nodes marking. We mark the nodes of T using Lemma 6, with blocking factor t;

Weights partitioning. Along each of the d dimensions, we partition the space $[n]^d$ into $\lceil n/t \rceil$ slabs, using axis-aligned hyper-planes, in such a way that each slab contains exactly t (except, possibly, for the last slab, which may contain less than t) nodes of the tree T (this is always possible, as the weights are in rank space). Precisely, we maintain a list λ_i of slabs per weight component: $\lambda_{i,j} \triangleq \{z \in T \mid (j-1)t < w_i(z) \leq \min\{jt, n\}\}$, for $1 \leq i \leq d$ and $1 \leq j \leq \lceil n/t \rceil$. Somewhat abusing notation, we use "slab $\lambda_{i,j}$" to denote both the orthogonal range and the corresponding set of nodes defined above;

[5] The original works state the results for path reporting, but these results imply the results on path emptiness as stated in Lemma 14.

Path emptiness. For each category $\gamma \in [\sigma]$, we build the tree extraction T_γ of all the nodes with category γ. The nodes of T_γ inherit the weights of the original nodes in T. Each T_γ, in turn, is associated with the following data structures:

- The path emptiness data structure C_γ of Lemma 14;
- y-fast tries [15] $\{Y_{\gamma,i}\}_{i=1}^{d}$ such that $Y_{\gamma,j}$ maps the j^{th} weights of T_γ into rank space $[|T_\gamma|]$;

Mapping structures. Maintained using Lemma 2 are also trees K and G with the topology of T:

- K is labeled over $\{0,1\}$ such that a node $z \in K$ is labeled with 1 *iff* its copy in T is marked;
- G is labeled over $[\sigma]$ such that a node $z \in G$ is given a label γ *iff* its copy in T has category γ;

Tabulation. For each of the $\Theta((n/t)^{2d+2})$ *spans* we store, in a table M, the number of distinct categories occurring in the span. Precisely, let the indices $i_1, i_2, \ldots, i_d, j_1, j_2, \ldots, j_d$ be such that $\forall k 1 \leq i_k \leq j_k \leq \lceil n/t \rceil$ and two nodes x' and y' be marked. Then, the *span* corresponding to these indices is the set $\{z \in P_{x',y'} \mid z \in \cap_{k=1}^{d}(\cup_{l=i_k}^{j_k} \lambda_{k,l})\}$ (i.e. the set of nodes on the path $P_{x',y'}$ such that their weights fall into the relevant rectangle in $[n]^d$). To save space, the nodes x' and y' are referred to by their relative preorder ranks x_M and y_M among the marked nodes. Now, M is a table whose entry $M[x_M, y_M, i_1, j_1, i_2, j_2, \ldots, i_d, j_d]$ stores the number of distinct categories in the span corresponding to the given indices.

▶ **Lemma 15.** *The data structures built in Section 4.2 occupy $\mathcal{O}(\mathtt{s}_d(n) + (n/t)^{2d+2})$ words of space.*

Due to space considerations, we consign the proof of Lemma 15 to the full version of the paper.

We next describe how to resolve queries and analyze the query time:

▶ **Lemma 16.** *The data structures built in Section 4.2 answer a categorical path range counting query in $\mathcal{O}(t \cdot \tau_d(n))$ time.*

Proof. Let $P_{x,y}$ and $Q = \prod_{k=1}^{d}[a_k, b_k]$ be the query arguments. If $|P_{x,y}| \leq t$, we explicitly traverse the path $P_{x,y}$ and count the number of unique categories encountered; the exact procedure is subsumed in the discussion that follows. We therefore assume $|P_{x,y}| > t$, and split the path $P_{x,y}$ into $P_{x,x'}$, $P_{x',y'}$, and $P_{y,y'}$, as described in the proof of Theorem 10. One has that $|P_{x,x'}|, |P_{y,y'}| \leq t$.

The grid of the marked nodes and the slabs induce a decomposition of the query region into the span and the "rim" – the parts of the query region abutting the span. Of these, only the rim is meant to be explicitly traversed. The details follow.

First, we initialize the indices $i_1, i_2, \ldots, i_d$ and $j_1, j_2, \ldots, j_d$ as $i_k := \lceil a_k/t \rceil$ and $j_k := \lceil b_k/t \rceil$, for all $1 \leq k \leq d$. That is, i_k^{th} range contains a_k, and j_k^{th} range contains b_k. Furthermore, let x' and y' be respectively the x_M^{th} and y_M^{th} marked node, in preorder; one computes x_M and y_M using Proposition 9. Now the tuple $(x_M, y_M, i_1{+}1, j_1{-}1, i_2{+}1, j_2{-}1, \ldots, i_d{+}1, j_d{-}1)$ determines a span, for which the answer – the number of distinct categories occurring therein – is already precomputed. We initialize the counter variable `res` holding the answer to the query with the table entry $M[x_M, y_M, i_1+1, j_1-1, i_2+1, j_2-1, \ldots, i_d+1, j_d-1]$. With this span, we also associate the pair of query arguments $P^{(span)} = P_{x',y'}$ and $Q^{(span)} = \prod_{k=1}^{d}[i_k t + 1, \min\{(j_k-1)t, n\}]$.

Next, our goal is to traverse the rim systematically, scanning for categories, while being careful not to double-count nor miss them. A description of such a traversal follows.

We split the query path $P_{x,y}$ into the "prefix" $P_{x,x'} \setminus \{x'\}$, the middle $P_{x',y'}$, and the "suffix" $P_{y,y'} \setminus \{y'\}$, using marked nodes x' and y' found earlier as in the proof of Theorem 10. Consider all the nodes $z \in P_{x',y'}$ whose weight vector $\mathbf{w}(z)$ pushes z outside of the span. The loci of such vectors in $[n]^d$ clearly can be covered with $\mathcal{O}(d) = \mathcal{O}(1)$ disjoint axis-aligned rectangles – henceforth *canonical rectangles* – in such a way that each canonical rectangle $r \in \mathcal{D}$ lies entirely within some $\lambda_{i,j}$. For each dimension k, there are at most two canonical rectangles within slabs λ_{k,i_k} and λ_{k,j_k}. We assume the availability of such a cover $\mathcal{D}$. From each canonical rectangle $r \in \mathcal{D}$, a *canonical set* $s(r) \triangleq \{z \in P_{x',y'} \mid \mathbf{w}(z) \in r\}$ is constructed. As each r lies inside a slab, one has $|s(r)| \leq t$.

We enumerate the nodes in the rim in (say) the following order: the nodes of the prefix, the nodes of the suffix, and the nodes in each canonical set. Within each set, the nodes are conceptually ordered in the direction from x to y (the traversal order is ascertained via Lemma 1). We walk through these sets, while referring as *previously seen* to the union of the span and the processed sets.

Let z, with $\gamma = \mathtt{c}(z)$, be the current node in our traversal of the rim. The category γ contributes towards `res` *iff* each query from the following list E of path range emptiness queries comes back as `false`. All queries in E are launched on C_γ, the query parameters being restrictions of previously seen sets to the tree T_γ. Namely, we (i) adjust the weights to the rank space of T_γ using the y-fast tries $\{Y_{\gamma,k}\}_{k=1}^{d}$; and (ii) map the associated path to T_γ using Proposition 9 (the path component of a canonical set is $P_{x',y'}$, whereas for the span we use the previously defined $P^{(span)}$ and $Q^{(span)}$). Finally, we also check the part of the current set (be it the prefix, the suffix, or a canonical set) that precedes z in our conventional x-to-y ordering. If the current set is the prefix, we launch a path range emptiness query for the path $P_{x,z} \setminus \{z\}$ and the range Q on C_γ. (For the suffix and canonical sets, this last step is analogous.)

Mapping the query path takes $\mathcal{O}(\lg \frac{\lg \sigma}{\lg w})$ time (Lemma 8); the mapping of the weight-ranges using the y-fast tries is an additive $\mathcal{O}(\lg \lg n)$ time. We note that both time bounds do not exceed $\tau_d(n)$ (the query time stated in Lemma 14). The rim consisting of $\mathcal{O}(d) = \mathcal{O}(1)$ sets of $\mathcal{O}(t)$ nodes, with $\mathcal{O}(1)$ time per fetching an entry, the claim for the query time follows. ◀

As the procedure of zooming into T_γs can be of independent interest, we formalize it in the full version of the paper.

Combining Lemmas 15 and 16 one has

▶ **Theorem 17.** *Let $d \in \mathbb{N}$ be a constant. Let T be an ordinal tree on n nodes, each node $z \in T$ of which is assigned a category $\mathtt{c}(z) \in [\sigma]$, as well as a d-dimensional weight vector $\boldsymbol{w}(z)$, in rank space. Let, furthermore, $1 \leq t \leq n$ be a parameter set prior to construction. Then, for the categorical path range counting problem there exists a data structure such that it uses*

$\boldsymbol{d = 1}$*: either*

- $\mathcal{O}(n \lg \lg n + (n/t)^4)$ *words of space for the query time of* $\mathcal{O}(t \lg \lg n)$*; or*
- $\mathcal{O}(n + (n/t)^4)$ *words of space for the query time of* $\mathcal{O}(t \lg^{\epsilon} n)$*;*

In particular, a linear-space data structure has the query time $\mathcal{O}(n^{3/4} \lg^{\epsilon} n)$*;*

$\boldsymbol{d \geq 2}$*:* $\mathcal{O}(n \lg^{d-1+\epsilon} n + (n/t)^{2d+2})$ *words of space for the query time of* $\mathcal{O}(t \frac{\lg^{d-1} n}{(\lg \lg n)^{d-2}})$*. In particular, a linear-space data structure has the query time* $\widetilde{\mathcal{O}}(n^{\frac{2d+1}{2d+2}})$*.*

5 Sketching Data Structures for Approximate Categorical Path Range Counting

In this section we consider a probabilistic approach to the approximate categorical path range counting problem. Section 5.1 reviews *sketches* [10, 30] that we use to approximate the number of distinct categories. Then, in Section 5.2, we solve the $(1 \pm \epsilon)$-approximate categorical counting problem proper, with probability $1 - \delta$, for arbitrarily small constants $0 < \epsilon, \delta < 1$.

5.1 Sketches

For an arbitrary vector $\mathbf{a} = (a_1, a_2, \ldots, a_\sigma) \in \mathbb{R}^\sigma$, Cormode et al. [10] introduce *Hamming norm* $|\mathbf{a}|_H$ of $\mathbf{a}$, defined as $|\mathbf{a}|_H \triangleq \sum_{i=1}^{\sigma} |a_i|^0$, with $|0|^0 \triangleq 0$. It is clear that $|\mathbf{a}|_H = |\{a_i \mid a_i \neq 0\}|$, i.e. the Hamming norm equals the number of non-zero components in $\mathbf{a}$. For our purposes, this original vector $\mathbf{a}$ is the *frequency array* $(a_1, a_2, \ldots, a_\sigma)$, with a_i standing for the number of the occurrences of the category i. While referring the reader to [10] and references therein for discussion in depth, we state the main result we build on:

► **Lemma 18** ([10, 30]). *Let $0 < \epsilon, \delta < 1$ be constants. Given a vector $\mathbf{a}$, there exists a sketch, $h(\mathbf{a})$, that requires $m = \mathcal{O}(\frac{1}{\epsilon^2} \cdot \lg \frac{1}{\delta})$ words and allows approximation of $|\mathbf{a}|_H$ within a factor of $1 \pm \epsilon$ of the true answer with probability $1 - \delta$. Updating the sketch and computing $|\mathbf{a}|_H$ both take $\mathcal{O}(m)$ time. Furthermore, if $\mathbf{a}$ and $\mathbf{b}$ are two vectors, then $h(\mathbf{a} \pm \mathbf{b}) = h(\mathbf{a}) \pm h(\mathbf{b})$.*

A clarification is in order regarding the *update* operation referred to in Lemma 18. The scenario of Cormode et al. [10] is that of observing a stream while maintaining an approximation to the number of the distinct values seen so far. Update refers to updating the approximation upon observing the next value in the stream. The gist of our solution is in treating certain paths $P_{x,\perp}$ each as a stream of its own and maintaining *several* sketch-summaries thereof. Our adaptation comprises (i) using the same transformation matrix [10, 30] throughout the computations; and (ii) building the sketches using $\delta' = \frac{\delta}{n^{2d+2}}$. Ensured by (i) is the "compatibility" of any two arbitrarily chosen summaries – sketches are obtained by a linear transformation [10] of (in our case) the frequency array, with linearity implying additivity. With (ii), the value of m in Lemma 18 works out to be $m = \mathcal{O}(\frac{1}{\epsilon^2} \lg \frac{1}{\delta/n^{2d+2}}) = \mathcal{O}(\lg n)$.

5.2 $(1 \pm \epsilon)$-Approximate Categorical Path Range Counting

We first solve the $(1 \pm \epsilon)$-approximate categorical path counting problem for unweighted trees; then we use Lemma 4 to extend the data structures to trees weighted with d-dimensional weight vectors.

First, we apply Lemma 19 (whose proof easily follows from the Pigeonhole Principle) with parameter t to mark $\mathcal{O}(n/t)$ nodes in the tree:

► **Lemma 19** ([27]). *Let $1 \leq t \leq n$ be an integer parameter. There exists a level l' no deeper than t such that, when one* marks *the nodes on every t^{th} level of the tree T, starting from l', then there are $\mathcal{O}(n/t)$ marked nodes in total.*

Next, at each marked node $z \in T$, one stores the sketch $h(z)$ as a summary of the categories occurring on the path $P_{z,\perp}$. Indeed, let $\mathbf{a}(z)$ be the (conceptual) frequency vector for the categories on the path $P_{z,\perp}$. Then we associate with z a length-m vector $h(z)$ – the sketch of $\mathbf{a}(z)$. One thus obtains

▶ **Lemma 20.** *The data structures built in Section 5.2 occupy* $\mathcal{O}(n+(n/t)\lg n)$ *words and answer a* $(1\pm\epsilon)$*-approximate categorical path counting query in* $\mathcal{O}(t\lg n)$ *time, with probability* $1-\delta$.

Proof. There are $\mathcal{O}(n/t)$ marked nodes, each storing $m=\mathcal{O}(\lg n)$ words, hence the claimed space.

By the additivity stated in Lemma 18, the answer to a query with arbitrary query nodes x and y is simply $h(x)+h(y)-2\cdot h(\mathtt{LCA}(x,y))$, corrected for $\mathtt{c}(\mathtt{LCA}(x,y))$ using Lemma 18. Therefore, it is sufficient to show how to compute $h(x)$ for an arbitrary node x.

The path $P_{x,\perp}$ can be represented as $P_{x,x'}\cup P_{x',\perp}$, where x' is the closest marked ancestor of x. If there is no such x', then, by construction, the depth of x is no greater than t; this case is solved by an explicit traversal, as shown below. We therefore assume the existence of such x'. The case $x=x'$ is trivial, as we use $h(x')$ directly. If $x\neq x'$, then by construction $|P_{x,x'}|\leq t$. We initialize a zero-vector s of length m and the *current node* to x. We then start ascending the path $P_{x,x'}$ in the direction of x' until the current node equals x'. (Informally, the path $P_{x,x'}$ in the direction towards x' is our "stream", and the "next value" is the category of the next node encountered on this path.) For the category of the node currently being observed, the current sketch s is updated using Lemma 18. This increment thus being an $\mathcal{O}(m)=\mathcal{O}(\lg n)$-time operation, the traversal's time cost is $\mathcal{O}(tm)=\mathcal{O}(t\lg n)$. At the node x', we return the sum of s and of $h(x')$ (which is precomputed), as the sketch for $P_{x,\perp}$. ◀

When the marked nodes in Lemma 20 are assigned the sketches, they are assigned semigroup elements in the sense of Definition 3, the regular component-wise addition in vectors being the corresponding semigroup sum operator. Unmarked nodes are assigned conceptual zero-vectors in view of formal compliance with Definition 3; as the sketches stored at unmarked nodes are never consulted, this has no effect on our algorithm.

The combination of Lemma 20 and Lemma 4 thus yields the following

▶ **Theorem 21.** *Let* $0<\epsilon,\delta<1$ *be arbitrarily small constants, and* $d\geq 1$ *be an integer constant. Let, furthermore,* T *be an ordinal tree on* n *nodes, each node* z *of which is assigned a weight* $\mathbf{w}(z)\in[n]^d$, *as well as a category* $\mathtt{c}(z)\in[\sigma]$. *Then, there exists a data structure of* $\mathcal{O}((n+\frac{n}{t}\lg n)\lg^d n)$ *words that solves a* $(1\pm\epsilon)$*-approximate categorical path range counting query in* $\mathcal{O}(t\lg^{d+1} n)$ *time, with success probability no less than* $1-\delta$.

Proof. We iteratively apply Lemma 4 to Lemma 20. Since we start with G^0 (supplied by Lemma 20) and apply Lemma 4 exactly d times, the space cost of $\mathcal{O}((n+\frac{n}{t}\lg n)\lg^d n)$ words and the query time of $\mathcal{O}(t\lg^{d+1} n)$ follows.

Our data structure fails *iff* at least one of the $\Theta(n^{2d+2})$ possible queries fails. The total probability of failure therefore is at most the sum of the failure probabilities of each of these $\Theta(n^{2d+2})$ queries. When building the data structure, we thus use a stronger guarantee of $\delta'=\frac{\delta}{n^{2d+2}}$, which also means that the length m of the vectors storing sketches is $\mathcal{O}(\lg\frac{1}{\delta'})=\mathcal{O}(\lg n)$. ◀

References

1 Peyman Afshani and Konstantinos Tsakalidis. Optimal Deterministic Shallow Cuttings for 3-d Dominance Ranges. *Algorithmica*, 80(11):3192–3206, 2018.

2 Pankaj K. Agarwal, Sathish Govindarajan, and S. Muthukrishnan. Range Searching in Categorical Data: Colored Range Searching on Grid. In *ESA*, pages 17–28, 2002.

3 Nikhil Bansal and Ryan Williams. Regularity Lemmas and Combinatorial Algorithms. *Theory Comput.*, 8(1):69–94, 2012.

4 Timothy M. Chan. Speeding up the Four Russians Algorithm by About One More Logarithmic Factor. In *SODA*, pages 212–217, 2015.

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

6 Timothy M. Chan, Qizheng He, and Yakov Nekrich. Further Results on Colored Range Searching. In *SoCG*, volume 164, pages 28:1–28:15, 2020.

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

8 Timothy M. Chan and Yakov Nekrich. Better Data Structures for Colored Orthogonal Range Reporting. In *SODA*, pages 627–636, 2020.

9 Moses Charikar, Surajit Chaudhuri, Rajeev Motwani, and Vivek R. Narasayya. Towards Estimation Error Guarantees for Distinct Values. In *PODS*, pages 268–279, 2000.

10 Graham Cormode, Mayur Datar, Piotr Indyk, and S. Muthukrishnan. Comparing Data Streams Using Hamming Norms (How to Zero In). In *VLDB*, pages 335–345, 2002.

11 Stephane Durocher, Rahul Shah, Matthew Skala, and Sharma V. Thankachan. Top-k color queries on tree paths. In *SPIRE*, pages 109–115, 2013.

12 Stephane Durocher, Rahul Shah, Matthew Skala, and Sharma V. Thankachan. Linear-Space Data Structures for Range Frequency Queries on Arrays and trees. *Algorithmica*, 74(1), 2016.

13 Hicham El-Zein, J. Ian Munro, and Yakov Nekrich. Succinct Color Searching in One Dimension. In *ISAAC*, pages 30:1–30:11, 2017.

14 Michael L. Fredman and Dan E. Willard. Surpassing the Information Theoretic Bound with Fusion Trees. *J. Comput. Syst. Sci.*, 47(3):424–436, 1993.

15 Michael L. Fredman and Dan E. Willard. Trans-Dichotomous Algorithms for Minimum Spanning Trees and Shortest Paths. *J. Comput. Syst. Sci.*, 48(3):533–551, 1994.

16 Harold N. Gabow, Jon Louis Bentley, and Robert E. Tarjan. Scaling and related techniques for geometry problems. In *STOC*, pages 135–143, 1984.

17 Travis Gagie, Meng He, and Gonzalo Navarro. Tree Path Majority Data Structures. In *ISAAC*, volume 123, pages 68:1–68:12, 2018.

18 Travis Gagie and Juha Kärkkäinen. Counting Colours in Compressed Strings. In *CPM*, pages 197–207, 2011.

19 Arnab Ganguly, J. Ian Munro, Yakov Nekrich, Rahul Shah, and Sharma V. Thankachan. Categorical Range Reporting with Frequencies. In *ICDT*, pages 9:1–9:19, 2019.

20 Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete Mathematics: A Foundation for Computer Science, 2nd Ed.* Addison-Wesley, 1994.

21 Roberto Grossi and Søren Vind. Colored Range Searching in Linear Space. In *SWAT*, volume 8503, pages 229–240, 2014.

22 Prosenjit Gupta, Ravi Janardan, and Michiel Smid. Further results on generalized intersection searching problems: Counting, reporting, and dynamization. *J. Algorithms*, 19(2):282–317, 1995.

23 Meng He and Serikzhan Kazi. Path and Ancestor Queries over Trees with Multidimensional Weight Vectors. In *ISAAC*, volume 149, pages 45:1–45:17, 2019.

24 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.

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

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

27 David A. Hutchinson, Anil Maheshwari, and Norbert Zeh. An external memory data structure for shortest path queries. *Discret. Appl. Math.*, 126(1):55–82, 2003.

28 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.

29 Haim Kaplan, Natan Rubin, Micha Sharir, and Elad Verbin. Efficient Colored Orthogonal Range Counting. *SIAM J. Comput.*, 38(3):982–1011, 2008.

30 Ying Kit Lai, Chung Keung Poon, and Benyun Shi. Approximate colored range and point enclosure queries. *J. Discrete Algorithms*, 6(3):420–432, 2008.

31 Kasper Green Larsen and Freek van Walderveen. Near-Optimal Range Reporting Structures for Categorical Data. In *SODA*, pages 265–276, 2013.

32 S. Muthukrishnan. Efficient algorithms for document retrieval problems. In *SODA*, pages 657–666, 2002.

33 Yakov Nekrich. Efficient range searching for categorical and plain data. *ACM Trans. Database Syst.*, 39(1):9:1–9:21, 2014.

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

35 Virginia Vassilevska Williams. Multiplying matrices faster than Coppersmith-Winograd. In *STOC*, pages 887–898, 2012.

36 Huacheng Yu. An improved combinatorial algorithm for Boolean matrix multiplication. *Inf. Comput.*, 261:240–247, 2018.