On Approximate Colored Path Counting[⋆]

Younan Gao[1][0000−0003−4984−2551] and Meng He[2][0000−0003−0358−7102]

[1] Département d'informatique et d'ingénierie, Université du Québec en Outaouais, Gatineau, Canada
[2] Faculty of Computer Science, Dalhousie University, Halifax, Canada
gaoy03@uqo.ca, mhe@cs.dal.ca

Abstract. Given an ordinal tree T on n nodes in which each node is assigned a color from $\{0, 1, \ldots, C-1\}$, an approximate colored path counting query asks for an approximation of the number, `occ`, of distinct colors assigned to nodes in a query path. We first present data structures that can compute a 2-approximate answer, i.e., a number in $[\texttt{occ}, 2\texttt{occ}]$, and achieve three different time/space trade-offs: i) an $O(n)$-word structure with $O(\lg^{\lambda} n)$ query time for any constant $0 < \lambda < 1$, ii) an $O(n \lg\lg n)$-word structure with $O(\lg\lg n)$ query time and iii) an $O(n \lg^{\lambda} n)$-word structure with $O(1)$ query time. The first trade-off beats the $O(\lg n/\lg\lg n)$ query time of the linear-word 2-approximate structure in previous work. We then design an $O(n)$-word structure which can compute in $O(\epsilon^{-2} \lg n)$ time a $(1 \pm \epsilon)$-approximate answer, i.e, a number in $[(1-\epsilon)\texttt{occ}, (1+\epsilon)\texttt{occ}]$, for any $\epsilon \in (0, 1)$. Previously, when the space cost is $O(n)$ words, the only known solution computes a $(1\pm\epsilon)$-approximate answer in $O(\epsilon^{-4} \lg^2 n)$ time with success probability no less than $1-\delta$, where δ is an arbitrary constant in $(0, 1)$; our solution not only has faster query time but also always returns a $(1 \pm \epsilon)$-approximation. When designing $(1 \pm \epsilon)$-approximate solutions, our techniques also yield an $O(n)$-word structure that can answer a colored type-2 path counting query in $O(\texttt{occ})$ time; this query reports the number of occurrences of each distinct color in a query path. This result improves the best previous linear-word solution in which the query time is $O(\texttt{occ} \lg\lg n)$.

Keywords: Path queries · Colored path counting · Colored path reporting · Approximate colored path counting

1 Introduction

In tree-structured data, information such as categories can be viewed as colors assigned to tree nodes. One query which can retrieve such information is the *colored path counting query*. It is defined over an ordinal tree[3] T on n nodes, each assigned a color from $\{0, 1, \ldots, C-1\}$, where $C \leq n$, and it computes the number, `occ`, of distinct colors assigned to the nodes in any query path in T.

[⋆] This work was supported by NSERC of Canada.

[3] This query can be defined over free trees. Following [12], we assume that the input tree is ordinal, so that we can use data structures for ordinal trees directly [13, 14].

When the tree structure is a single path, this query becomes the well-known 1D colored range counting query, for which Nekrich [19] designed an $O(n)$-word solution with $O(\lg \texttt{occ}/\lg\lg n + 1)$ query time. However, a conditional lower bound gives evidence that the colored path counting query problem is much harder: He and Kazi [12] showed how to multiply two $\sqrt{n} \times \sqrt{n}$ boolean matrices by answering n colored path counting queries over a tree of $O(n)$ nodes. This reduction means, with current knowledge, the total running time of answering n colored path counting queries, including preprocessing, cannot be faster than $n^{\omega/2}$, save for polylogarithmic speedups, where $\omega < 2.37286$ denotes the exponent of matrix multiplication [1]. Furthermore, since the best known combinatorial approach of multiplying two $n \times n$ Boolean matrices under the word RAM model requires $\Theta(n^3/\texttt{polylog}(n))$ time [23], the total time of answering n of these queries cannot be faster than $n^{1.5}$, save for polylogarithmic speedups, using pure combinatorial methods with current knowledge. He and Kazi designed an $O(n)$-word structure with $O(\sqrt{n}\lg\lg C)$ query time and $O(n^{3/2}\lg\lg C)$ preprocessing time under the word RAM model. More recently, Gao and He [10] considered the batched version of this problem and showed how to answer n queries, including preprocessing, in $O(n^{1.40704})$ time.

To achieve faster queries, approximate colored path counting problems have been studied. Two different ways of bounding approximate ratios have been considered [12]: a *c-approximate colored path counting query* computes a number in $[\texttt{occ}, c \cdot \texttt{occ}]$, while a $(1 \pm \epsilon)$*-approximate* query returns a number in $[(1 - \epsilon)\texttt{occ}, (1 + \epsilon)\texttt{occ}]$ for any $\epsilon \in (0, 1)$. In this paper, we study this problem and aim at improving previous results under both approximate measures.

We also note that 1D colored range counting is sometimes called *1D colored type-1 range counting* in the literature [11, 4], while *1D colored type-2 range counting* reports the number of occurrences of each distinct color in a query range. We can also generalize the latter to consider tree topology by defining *colored type-2 path counting* over a colored tree, which reports the number of occurrences of each distinct color in a query path. The $O(n)$-word data structure of Durocher et al. [7] can be used to answer a colored type-2 path counting query in $O(\texttt{occ}\lg\lg n)$ time. This is slower than the $O(\texttt{occ} + 1)$-time support for 1D colored type-1 range counting over points in rank space [11, 9]. Thus, another goal is to close this gap.

Previous work. By reducing colored path counting to path counting over weighted trees [15] using the chaining approach [11], He and Kazi [12] designed a linear space data structure that supports 2-approximate colored path counting in $O(\lg n/\lg\lg n)$ time. For $(1 \pm \epsilon)$-approximate colored path counting queries, they showed a sketching data structure that occupies $O(n + \frac{n}{\epsilon^2 t}\lg n)$ words and answers a query in $O(\epsilon^{-2} t \lg n)$ time[4] with success probability no less than $1 - \delta$, where t is an arbitrary integer in $[1, n]$ and δ is an arbitrary constant in $(0, 1)$. Setting $t = \lceil \epsilon^{-2}\lg n \rceil$ makes the space cost linear and the query time $O(\epsilon^{-4}\lg^2 n)$.

[4] He and Kazi [12] originally stated their result for constant ϵ, but it is easy to generalize their bounds when $\epsilon = o(1)$.

Similar approximate problems can also be defined for colored 1D range counting and colored 2D orthogonal range counting which generalizes the former by preprocessing colored points in 2D to efficiently compute the number of distinct colors assigned to points in an axis-aligned query rectangle. The same conditional lower bound for colored path counting also applies to the latter [17]. In 1D, given n colored points in the rank space, El-Zein et al. [8] designed an encoding data structure that uses $O(n)$-bits of space and supports c-approximate colored counting for any constant $c > 1$ in $O(1)$ time without accessing the given point set. In higher dimensions, Rahul [21] showed that $(1 \pm \epsilon)$-approximate colored range counting can be answered by combining a colored range reporting structure and a c-approximate colored range counting structure. With it, he designed an $O(n \lg n)$-word structure to support $(1 \pm \epsilon)$-approximate colored 2D range counting in $O(\epsilon^{-2} \lg n)$ time.

Regarding 1D colored type-2 range counting, Gupta et al. [11] designed an $O(n)$-word structure with $O(\lg n + \mathtt{occ})$ query time, over a set of n colored points on a real line. Ganguly et al. [9] stated that, by combining the approach of Gupta et al. and some other results [18, 22], the query time can be further improved to $O(1 + \mathtt{occ})$ if all points are in rank space. This query problem can also be generalized to point sets on the plane, and we refer to [6, 4] for recent work on 2D colored type-2 range counting. For colored trees, the linear word structures designed by Durocher et al. [7] can answer a colored type-2 path counting query in $O(\mathtt{occ} \lg \lg n)$ time. They did not state this result explicitly, but it is implied by the algorithmic steps stated in the proof of Theorem 6 in their article.

Our results. Under the word RAM model, we first design 2-approximate colored path counting structures with i) $O(n)$ words of space and $O(\lg^{\lambda} n)$ query time for any constant $0 < \lambda < 1$, ii) $O(n \lg \lg n)$ words of space and $O(\lg \lg n)$ query time and iii) $O(n \lg^{\lambda} n)$ words of space and $O(1)$ query time. In all three cases, the preprocessing time is $O(n \lg n)$. Hence the first trade-off beats the $O(\lg n / \lg \lg n)$ query time of the linear-word 2-approximate structure of He and Kazi [12]. We then design an $O(n)$-word $(1 \pm \epsilon)$-approximate colored path counting structure with $O(\epsilon^{-2} \lg n)$ deterministic query time and $O(n^2 \lg C \lg \lg C)$ expected preprocessing time. Compared to the sketching structure by He and Kazi [12] with $O(n)$-word space and $O(\epsilon^{-4} \lg^2 n)$ query time, we not only achieve improvement for query time but also guarantee that the query algorithm always returns a $(1 \pm \epsilon)$-approximation, though the cost of preprocessing is higher. In the new $(1 \pm \epsilon)$-approximate solution, our techniques also lead to a linear-word data structure supporting colored type-2 path counting in $O(\mathtt{occ})$ time. This result improves the solution of Durocher et al. [7] which has $O(\mathtt{occ} \lg \lg n)$ query time. See Table 1 for a comparison of our results to all previous results.

To achieve these results, we develop new techniques. For 2-approximate colored path counting, note that no further improvement can be made using the strategy of He and Kazi for this problem, due to the lower bound on (uncolored) 2D orthogonal range counting [20] (which is a special case of path counting over weighted trees). Instead, we adopt the strategy of Gao and He [10] for batched

Table 1: A summary of our results on approximate colored path counting and colored type-2 path counting, in which space costs are measured in words, ϵ is an arbitrary parameter in $(0, 1)$, λ is an arbitrary constant in $(0, 1)$, † marks an expected bound, and ‡ marks a solution that returns a $(1 \pm \epsilon)$-approximation with probability no less than $1 - \delta$ for any constant $\delta \in (0, 1)$.

	Space	Query	Preprocess	Ref
2-approx	$O(n)$	$O(\frac{\lg n}{\lg\lg n})$	$O(n\frac{\lg n}{\lg\lg n})$	[12]
	$O(n)$	$O(\lg^\lambda n)$	$O(n\lg n)$	Thm 1a)
	$O(n\lg\lg n)$	$O(\lg\lg n)$		Thm 1b)
	$O(n\lg^\lambda n)$	$O(1)$		Thm 1c)
$(1 \pm \epsilon)$-approx	$O(n + \frac{n\lg n}{\epsilon^2 t})$	$O(\epsilon^{-2} t\lg n)$	$O(\epsilon^{-2} n\lg n)$	[12]‡
	$O(n)$	$O(\epsilon^{-2}\lg n)$	$O(n^2\lg C\lg\lg C)^\dagger$	Thm 2
Type-2	$O(n)$	$O(\mathsf{occ}\lg\lg n)$	$O(n\lg n)$	[7]
		$O(\mathsf{occ})$		Lemma 6

colored path counting which applies centroid decomposition to decompose the tree into a hierarchy of components. A query is answered by locating and querying the component that satisfies these two conditions: This component contains the entire query path, and its centroid is in the path. This means, within each component, we need only support queries for the paths that pass through a fixed node, e.g, the selected centroid, given during the construction. This strategy however incurs $O(n \lg n)$ words of space cost in [10]. To address this, we design a data structure of $O(n)$ bits that answers 2-approximate queries in constant time, provided that a query path must contain a fixed node. To speed up the mapping of the endpoints of a query path to nodes in a specific component in a space-efficient manner, we borrow ideas from the solutions to the *ball inheritance* problem [5] and design data structures with different time-space trade-offs.

With regard to $(1 \pm \epsilon)$-approximate colored path counting queries, we adapt the reduction by Rahul [21] and make it work for trees. When doing so, we design a solution to colored type-2 path counting with optimal query time.

2 Preliminaries

This section introduces the notation and the previous results used in this paper.

Notation. Given an ordinal tree T, we use $|T|$ to represent the number of nodes in T and $\perp$ to represent its root. Each node of T is assigned a color encoded by an integer in $\{0, 1, \cdots, C - 1\}$, where $C \le n$. We identify each node by its preorder rank, i.e., node x is the x-th node in a preorder traversal (x starts from 0), and $c(x)$ denotes the color of node x. Furthermore, $P_{x,y}$ denotes the path between nodes x and y, and $C(P_{x,y})$ denotes the set of colors that appear in it.

Navigation in colored ordinal trees. To support navigational operations over the input tree, we apply the succinct representations of ordinal trees [13] and labeled trees [14]. The operations and their complexity are summarized in Lemma 1. As in previous work, we refer to a node (resp. ancestor) colored in α as an α-node (resp. α-ancestor).

Lemma 1 ([13, 14]). *Let T denote a labeled ordinal tree on n nodes, each of which is assigned a color from $\{0, 1, \ldots, C-1\}$, where $C \leq n$. A data structure occupying $n \lg C + 2n + o(n \lg C)$ bits can be built over T in $O(n)$ time[5] to support:*

– *counting the number,* $\texttt{depth}(x)$*, of ancestors of x in $O(1)$ time,*
– *counting the number,* $\texttt{depth}_\alpha(x)$*, of α-nodes in $P_{x,\perp}$ in $O(\lg\lg C)$ time,*
– *finding the lowest common ancestor,* $\texttt{lca}(x, y)$*, of x and y in $O(1)$ time,*
– *finding the parent node,* $\texttt{parent}(x)$*, of non-root node x in $O(1)$ time,*
– *finding x's lowest proper α-ancestor,* $\texttt{parent}_\alpha(x)$*, in $O(\lg\lg C)$ time and*
– *finding x's ancestor,* $\texttt{level\_anc}(x, d)$*, at depth* $\texttt{depth}(x) - d$ *in $O(1)$ time.*

Given a query path $P_{x,y}$ in a tree and a color α, a *colored path emptiness* query determines whether color α appears in $P_{x,y}$. He and Kazi [12] showed how to use $\texttt{depth}_\alpha$ and $\texttt{lca}$ to compute the number of appearances of α in $P_{x,y}$ in $O(\lg\lg C)$ time. Via counting the number of appearances of color α in $P_{x,y}$, one can figure out whether or not α appears in $P_{x,y}$. Hence, Lemma 1 can support colored path emptiness query in $O(\lg\lg C)$ time.

Partial rank. Let $A[0..n-1]$ be a sequence of symbols over alphabet $\{0, 1, \ldots, \sigma-1\}$. The *partial rank* operation [2], $\texttt{rank}'(A, i)$, counts the number of elements equal to $A[i]$ in $A[0..i]$.

Lemma 2 ([2]). *Given a sequence $A[0..n-1]$ over alphabet $\{0, 1, \ldots, \sigma-1\}$, where $\sigma \leq n$, a structure of $O(n \lg \sigma)$ bits can be constructed in $O(n)$ time to support* $\texttt{rank}'$ *in $O(1)$ time.*

Tree extraction [15] Given a subset, X, of nodes of an ordinal tree T, the extracted tree, T_X, can be constructed by deleting each node $v \notin X$ using the following approach: If v is not the root, let $u = \texttt{parent}(v)$. We remove v and its incident edges from T and insert its children into the list of children of u, replacing v in this list while preserving these children's original left-to-right order. This means that v's left and right siblings before the deletion will respectively become the left and right siblings of its children after the deletion. If v is the root, then, before we apply the same procedure to delete v, we add a dummy root to T and make it the parent of v, so that T_X will remain a tree.

To map a path $P_{x,y}$ in T to a path in T_X, we use the operation, $\texttt{decompose}(x, y)$, defined in [12]: If $P_{x,y} \cap X = \emptyset$, it returns $\texttt{null}$. Otherwise, let $\hat{x}$ and $\hat{y}$ denote the nodes in $P_{x,y} \cap X$ that are closest to x and y, respectively (this can be the

[5] As mentioned by Gao and He [10], the string representation of Belazzougui *et al.* [2] needs to be used in the framework of He *et al.* [14] to achieve $O(n)$ deterministic preprocessing time, at the cost of slowing down labeled operations from $O(\lg \frac{\lg C}{\lg w})$ in [14] to $O(\lg\lg C)$ in this lemma.

node x or y itself if it is in X). Then $\texttt{decompose}(x, y)$ returns the nodes x' and y' in T_X that correspond to (i.e., whose original copies are) nodes $\hat{x}$ and $\hat{y}$ in T, respectively.

Lemma 3 ([12, Proposition 9]). *Given a tree T on n nodes and a tree extraction T_X, an $O(n)$-bit structure on top of T and T_X can be constructed in $O(n)$ time to support* `decompose` *in $O(1)$ time.*

3 2-Approximate Colored Path Counting

For 2-approximate colored path counting, we first consider a special case in which query paths must contain a fixed tree node specified in the preprocessing step (Section 3.1). Then we generalize it for arbitrary paths (Sections 3.2-3.3).

3.1 Counting over a Path that Contains a Fixed Node

Fix a node v of T, and we design an $O(n)$-bit encoding data structure that supports 2-approximate colored path counting over any query path containing v. To answer a query, our encoding data structure does not need to access T after preprocessing, provided that the preorder ranks of the endpoints of the query path are known.

Lemma 4. *Let T be a colored tree on n nodes and fix any node v in T. A data structure of $O(n)$ bits can be constructed in $O(n)$ time to support 2-approximate colored path counting over any query path containing v in $O(1)$ time.*

Proof. We associate a binary label $B(u)$ to each node u of T as follows: If $u = v$, then $B(u) = 1$. Otherwise, locate the node, t, in $P_{u,v}$ that is adjacent to u. If color $c(u)$ appears in $P_{t,v}$, then set $B(u) = 0$. If not, set $B(u) = 1$. We discard the original colors of T, treat these labels as node colors and represent T with these labels using Lemma 1. Since there are only two possible labels, this uses $3n + o(n)$ bits.

Let $P_{x,y}$ be a query path containing v. Consider the nodes in the subpath $P_{x,v}$ one by one in the direction from v to x. Observe that, each time we see a node labeled by 1, we encounter a color that has not been seen previously. Therefore, the number of 1-bits assigned to nodes in $P_{x,v}$ is equal to $|C(P_{x,v})|$. Similarly, the number of 1-bits assigned to nodes in $P_{y,v}$ is equal to $|C(P_{y,v})|$. Therefore, the number of 1-bits assigned to nodes in $P_{x,y}$ is a 2-approximation of the precise answer. Following the discussion after Lemma 1, this number can be computed in $O(\lg\lg C) = O(1)$ time using operations `lca` and $\texttt{depth}_1$, as $C = 2$.

To prove the bound on construction time, it suffices to show that these binary labels can be assigned in $O(n)$ time. This can be done by performing a depth-first traversal of T using v as the starting node. During this traversal, we also update an array $A[0..C-1]$, and the invariant that we maintain is that, each time we visit a node u, $A[i]$ stores the number of nodes in $P_{v,u}$ that are assigned color i in the original tree T. The following are the steps: We start the traversal from vertex v, set $A[c(v)] = 1$ and initialize all other entries of A to 0. During the traversal, each time we follow an edge (x, y) with $x \in P_{v,y}$, there are two cases. In the first case, we follow this edge to visit node y. Then this is the first time we visit y. We check if $A[c[y]] = 0$. If it is, then the color of y does not appear in $P_{v,x}$, so we set $B(y) = 1$. Afterwards, we increment $A[c[y]]$ to maintain the invariant. Otherwise, we set $B(y) = 0$ and also increment $A[c[y]]$. In the second case, we follow this edge to visit x. Since x is closer to v than y is, we have visited x before and we will not traverse any paths containing y in the future. In this case, we decrement $A[c[y]]$ to maintain the invariant. □

3.2 Counting over Arbitrary Paths

To support 2-approximate colored path counting queries over arbitrary paths, we transform the given ordinal tree T on n colored nodes into a binary tree $\tilde{T}$. Our transformation, similar to that used by Chan et al. [3], works as follows: For each node v with degree d, where $d > 2$, we remove the edges between v and its children, $v_1, v_2, \ldots, v_d$, to detach the subtrees rooted at these children from T. For the convenience of the description, let v_0 denote the node v. We then create $d - 2$ dummy nodes, $\tilde{v}_1, \tilde{v}_2, \ldots, \tilde{v}_{d-2}$, each assigned the color of v_0. Next we add edges to reconnect v_0 and its d children with the newly created dummy nodes as follows: For $t = 1, 2, \ldots, d-2$, make v_t and $\tilde{v}_t$ the left and right children of v_{t-1}, respectively. Afterwards, make v_{d-1} and v_d the left and right children of $\tilde{v}_{d-2}$, respectively. The resulting tree is $\tilde{T}$ which has at most $2n$ nodes. This transformation preserves preorder among the original nodes of T. More importantly, any path $P_{x,y}$ in T and its corresponding path, $\tilde{P}$, in $\tilde{T}$ share the same set of colors. To see this, observe that the lowest common ancestor, z, of nodes x and y in T is the only original node that appears in $P_{x,y}$ but may not necessarily appear in $\tilde{P}$. If z does not appear in $\tilde{P}$, then $\tilde{P}$ must contain a dummy node created for z which is also colored in $c(z)$. On the other hand, any original node in $\tilde{P}$ must also be in $P_{x,y}$ in T, while any dummy node that appears in $\tilde{P}$ must satisfy the condition that the original node it is created for must be in $P_{x,y}$ in T.

After this transformation, for each node in T, we store the preorder rank of its corresponding node in $\tilde{T}$. Then we follow the strategy in [10] to decompose $\tilde{T}$ recursively using *centroid decomposition* [16], but different data structures will be constructed this time. Here a *centroid* of an m-node tree is a node whose removal splits the tree into connected components, each containing at most $m/2$ nodes. It is known that a centroid can be found in $O(m)$ time.

We now give the details of the recursion. At level 0 of the recursion, we call tree $\tilde{T}$ the level-0 component. We find a centroid, u, of $\tilde{T}$ and construct the data structure in Lemma 4 supporting 2-approximate queries over paths containing u. Since $\tilde{T}$ is a binary tree, after removing u, we are left with at most three connected components, and we add u back into the smallest component. In this way, tree $\tilde{T}$ is partitioned into at most three pairwise-disjoint connected components in the level-0 recursion, each of which is a tree on no more than $|\tilde{T}|/2$

nodes. We call each of these three components a *level*-1 *component* and build the data structure recursively upon each of them. In general, at level i of the recursion, we compute a centroid, v, of each level-i component γ. We then use Lemma 4 to construct a data structure $D(\gamma)$ supporting 2-approximate colored path counting over paths that are entirely contained within γ and also contain v. This component can be partitioned into up to three level-$(i+1)$ components using the approach described above. We call component γ the *parent* of these up to three level-$(i+1)$ components, and each of these up to three level-$(i+1)$ components is a *child* of γ. A component that has a single node is called a *base component* and is not partitioned further. Hence, the recursion has $O(\lg n)$ levels.

Suppose that query path $P_{x,y}$ is contained entirely within a level-t component γ but not in any level-$(t+1)$ components. This means that $P_{x,y}$ contains the centroid of γ. Therefore, we can find a 2-approximate of $|C(P_{x,y})|$ using the data structure $D(\gamma)$, provided that the preorder ranks of x and y in γ are known. To locate component γ and then to compute the preorder rank of x and y in it, we define a *component tree* $\mathcal{CT}$. A component tree is a 3-ary tree in which each node represents a component and the edges represent the parent-child relationship between components. More specifically, a node v at level l of $\mathcal{CT}$ represents a level-l component $\mathcal{C}_v$, where l starts from 0. A node v of $\mathcal{CT}$ is the parent of another node u iff component $\mathcal{C}_v$ is the parent of component $\mathcal{C}_u$. Among the nodes that share the same parent in $\mathcal{CT}$, the relative order between them does not matter, so we order them arbitrarily. The height of $\mathcal{CT}$ is bounded by $O(\lg n)$, and each leaf in it represents a base component. Since each internal node has at least two children, $\mathcal{CT}$ has $O(n)$ nodes in total.

At each internal node v of $\mathcal{CT}$, we build an array $\texttt{SP}(v)$ of length $|\mathcal{C}_v|$, in which $\texttt{SP}(v)[i]$ is set to be d if the i-th node (in preorder) in $\mathcal{C}_v$ is stored in the d-th child component of v in the next level. Then we represent $\texttt{SP}(v)$ using Lemma 2 to support $\texttt{rank}'$. Since v can have at most 3 children, the alphabet size of $\texttt{SP}(v)$ is constant. Therefore, $\texttt{SP}(v)$ is represented in $O(|\texttt{SP}(v)|)$ bits. With these data structures, we can support queries over arbitrary paths and achieve Lemma 5.

Lemma 5. *Let T be an ordinal tree on n nodes in which each node is assigned a color. A data structure of $O(n)$ words can be constructed in $O(n \lg n)$ time to support 2-approximate colored path counting over T in $O(\lg n)$ time.*

Proof. Given a node $i \in \tilde{T}$, let π denote the path from the root of $\mathcal{CT}$ to the leaf of $\mathcal{CT}$ representing the base component that contains i, and let π_l denote the node in π whose depth is l. First, we show how to locate π_l and to compute the preorder rank of i in component $\mathcal{C}_{\pi_l}$ for $l = 0, 1, 2, \ldots$. The procedure proceeds as follows: We start at the root π_0 of $\mathcal{CT}$. The preorder rank of i in $\mathcal{C}_{\pi_0}$ is i, and π_1 is the $\texttt{SP}(\pi_0)[i]$-th child of π_0, following the definition of array $\texttt{SP}(\pi_0)$. In general, given that the preorder rank of i in $\mathcal{C}_{\pi_l}$ is j, one can find node π_{l+1}, which is the $\texttt{SP}(\pi_l)[j]$-th child of π_l. Since tree extraction preserves preorder, the preorder rank of i in $\mathcal{C}_{\pi_{l+1}}$ is $\texttt{rank}'(\texttt{SP}(\pi_l), j) - 1$. Each $\texttt{rank}'$ query takes constant time, so this procedure uses constant time per level of $\mathcal{CT}$.

Since each node of T stores the preorder rank of its corresponding node in $\tilde{T}$, to answer a query, it is sufficient to compute a 2-approximation of $|C(P_{x,y})|$ for a query path $P_{x,y}$ in $\tilde{T}$. This can be done by performing the top-down traversals of $\mathcal{CT}$ described in the previous paragraph for x and for y simultaneously until we reach the lowest level, l, of $\mathcal{CT}$ such that x and y are contained in the same level-l component γ. This process also gives us the preorder ranks of x and y in γ, which allows us to query $D(\gamma)$ to find a 2-approximate answer. Since $\mathcal{CT}$ has $O(\lg n)$ levels, the query algorithm uses $O(\lg n)$ time.

To analyze the space cost, observe that the total number of nodes in the components at the same level is at most the number of nodes of $\tilde{T}$, which is $2n$. The component tree contains $O(\lg n)$ levels, and all $D(\gamma)$'s and $\mathtt{SP}(v)$'s at the same level use $O(n)$ bits, for a total of $O(n \lg n)$ bits, or $O(n)$ words. The $O(n)$-node component tree $\mathcal{CT}$ itself occupies another $O(n)$ words of space. Therefore, the total space cost is $O(n)$ words. The data structure at each level can be constructed in linear time, so the overall construction time is $O(n \lg n)$. $\square$

3.3 Speeding up the Query

To further improve the query efficiency in Lemma 5, observe that two procedures introduced before require $O(\lg n)$ time for a query $P_{x,y}$ in $\tilde{T}$: The first locates the lowest component γ in $\mathcal{CT}$ that contains both nodes x and y, and the second computes the preorder ranks of x and y in γ. Previously, both procedures proceed in the same top-down traversal of $\mathcal{CT}$. Now, we perform them separately. For the first procedure, observe that the node representing component γ in $\mathcal{CT}$ must be the lowest common ancestor of the two leaves of $\mathcal{CT}$ representing the base components that contain nodes x and y, respectively. To locate γ in constant time, we can represent $\mathcal{CT}$ using Lemma 1 to support $\mathtt{lca}$ in $O(1)$ time and store with each node x of $\tilde{T}$ a pointer to the base component that contains x. This incurs $O(n)$ words of space and $O(n)$ preprocessing time. To improve the second procedure, we model it by defining an operation, $\mathtt{locate}(v, x)$; given a node v of $\mathcal{CT}$ and a tree node x of $\tilde{T}$ that appears in component $\mathcal{C}_v$, $\mathtt{locate}(v, x)$ returns the preorder rank of node x in $\mathcal{C}_v$.

To support $\mathtt{locate}(v, x)$, we borrow ideas from the solution to the ball inheritance problem [5] and achieve various trade-offs. Let π denote the path between the root of $\mathcal{CT}$ and the leaf representing the base component that contains x, and let π_l denote the node in π whose depth is l. Then each component $\mathcal{C}_{\pi_l}$ contains a copy of node x. Array $\mathtt{SP}$'s in Section 3.2 work as pointers between these copies in components at consecutive levels. The algorithm in the proof of Lemma 5 follows these pointers one by one till we locate x in $\mathcal{C}_v$, which requires $O(\lg n)$ time. To speed it up, we construct *skipping pointers* which allow us to jump over many levels at one time: Suppose that we have computed the preorder rank, i, of node x in component $\mathcal{C}_{\pi_l}$, and we need to locate x in $\mathcal{C}_{\pi_{l+\Delta}}$ for some positive integer Δ. What we can do is to build an array $\mathtt{SP}_\Delta(\pi_l)$ with length $|\mathcal{C}_{\pi_l}|$, in which $\mathtt{SP}_\Delta(\pi_l)[k]$ is set to d if the k-th node in preorder in $\mathcal{C}_{\pi_l}$ appears in the component represented by the d-th descendant of π_l at depth $l + \Delta$ of $\mathcal{CT}$. If this node is not stored in any level-$(l + \Delta)$ descendant component of π_l (this may happen when $\mathcal{CT}$ is not a complete tree), then $\mathtt{SP}_\Delta(\pi_l)[k] = -1$. Since π_l has up to 3^Δ descendants at level $l + \Delta$, we can represent $\mathtt{SP}_\Delta(\pi_l)$ in $O(|\mathcal{C}_{\pi_l}|\Delta)$

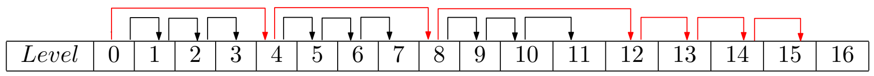

Fig. 1: An example of the first traversal strategy. In this example, the component tree $\mathcal{CT}$ has 16 levels, and $B = 4$. The arrows represent the skipping pointer. By following 6 skipping pointers, one can reach level-15 from the root level.

bits by Lemma 2 to support $\texttt{rank}'$. Then $\texttt{rank}'(\texttt{SP}_{\Delta}(\pi_l), i) - 1$ is the preorder rank of node x in $\mathcal{C}_{\pi_{l+\Delta}}$ and can be computed in $O(1)$ time. We regard $\texttt{SP}_{\Delta}(\pi_l)$ as an array of skipping pointers that connect the nodes in component $\mathcal{C}_{\pi_l}$ to the nodes in level-$(l + \Delta)$ descendant components of π_l.

If an array of skipping pointers map nodes in a level-l component to nodes in level-$(l+\Delta)$ descendants of this component, then we say that the length of each of these skipping pointers is Δ. Furthermore, based on previous discussions, storing a skipping pointer of length Δ incurs a space cost of $O(\Delta)$ bits. To achieve good time/space trade-offs, we design two strategies to decide what skipping pointers to construct for each level. Henceforth, let $h = O(\lg n)$ denote the height of $\mathcal{CT}$. Let $B \in [2, h]$ be an integer parameter to be chosen later, and let $\tau = \log_B h$; for simplicity, assume that τ is an integer.

In the first strategy, consider level l of the component tree $\mathcal{CT}$. For each integer $i \in [0, \tau - 1]$ such that l is a multiple of B^i but $l + B^i$ is not a multiple of B^{i+1}, we build an array of length B^i skipping pointers for each level-l component. See Figure 1 for an example. Since at most $\frac{h}{B^i}$ levels of $\mathcal{CT}$ have skipping pointers of length B^i, the total space cost of all the skipping pointers in this strategy is $\sum_{i=0}^{\tau-1}(\frac{h}{B^i}) \cdot O(nB^i) = O(n \lg n \log_B \lg n)$ bits, which is $O(n \log_B \lg n)$ words.

To use these skipping pointers to compute $\texttt{locate}(v, x)$, let $b_{\tau-1}b_{\tau-2}\cdots b_0$ denote the base-B expression of the depth[6], l_v, of node v in $\mathcal{CT}$. That is, each b_i is in $[0, B-1]$ and $l_v = \sum_{i=0}^{\tau-1} b_i B^i$. We then compute $\texttt{locate}(v, x)$ in τ phases. In phase-1, we start from the root of $\mathcal{CT}$ and follow length $B^{\tau-1}$ skipping pointers $b_{\tau-1}$ times. Each time after we follow a skipping pointer to reach a level of $\mathcal{CT}$, we use $\texttt{level\_anc}$ to locate the ancestor, u, of v at that level. We then follow the skipping pointers in $\texttt{SP}_{B^{\tau-1}}(u)$ to continue this phase. At the end of phase-1, we have located the ancestor of v at level $b_{\tau-1}B^{\tau-1}$ of $\mathcal{CT}$ and computed the preorder rank of node x in the component that this ancestor represents. In phase-2, we start from this ancestor and follow length $B^{\tau-2}$ skipping pointers $b_{\tau-2}$ times, and so on. In general, in phase-p, we follow length $B^{\tau-p}$ skipping pointers $b_{\tau-p}$ times, reach the ancestor of v at level $\sum_{j=1}^{p} b_{\tau-p}B^{\tau-p}$ of $\mathcal{CT}$ and compute the preorder rank of node x in the component represented by this ancestor. Thus, we reach v and compute the answer after τ phases. Since we follow at most $B - 1$ skipping pointers in each phase, the total running time is

[6] Note that a component tree has $O(\lg n)$ depths and the base-B expression of any depth can be encoded in $O(\log B \times \log_B \lg n) = O(\lg n)$ bits. Storing the base-B expressions of all $O(\lg n)$ depths uses $O(\lg n)$ words of space overall.

$O(B\tau) = O(B \log_B \lg n)$. Setting $B = \lg^{\lambda} n$ for an arbitrary constant $\lambda \in (0, 1)$ yields a solution with $O(n)$ space and $O(\lg^{\lambda} n)$-time support for `locate`.

The second strategy improves the running time of the above process by constructing a different set of skipping pointers so that each phase can be completed by following exactly one skipping pointer. Let l be an arbitrary level of $\mathcal{CT}$. For each integer $i \in [0, \tau - 1]$ such that l is a multiple of B^i, we construct for level-l clusters of skipping pointers of length $B^{i-1}, 2B^{i-1}, \ldots, (B-1)B^{i-1}$. With these skipping pointers, in phase-p, for each $p \in [\tau]$, of the above process, we only need one hop by following a skipping pointer of length $b_{\tau-p}B^{\tau-p}$, decreasing the total query time to $O(\tau) = O(\log_B \lg n)$. The total space cost of these skipping pointers is then at most $\sum_{i=0}^{\tau-1} \frac{h}{B^i} \cdot (B-1)O(nB^i) = O(nB \lg n \log_B \lg n)$ bits, which is $O(nB \log_B \lg n)$ words. Setting $B = 2$ bounds the space cost by $O(n \lg \lg n)$ and query time by $O(\lg \lg n)$, while setting $B = \lg^{\lambda} n$ bounds the space cost by $O(n \lg^{\lambda} n)$ and query time by $O(\lambda^{-1}) = O(1)$ for any constant $0 < \lambda < 1$.

With these three trade-offs for `locate`, we have the following theorem:

Theorem 1. *Let T be an ordinal tree on n nodes in which each node is assigned a color. A data structure of $s(n)$ words can be constructed in $O(n \lg n)$ time to support 2-approximate colored path counting over T in $q(n)$ time, where a) $s(n) = O(n)$ and $q(n) = O(\lg^{\lambda} n)$; b) $s(n) = O(n \lg \lg n)$ and $q(n) = O(\lg \lg n)$; or c) $s(n) = O(n \lg^{\lambda} n)$ and $q(n) = O(1)$ for any constant $0 < \lambda < 1$.*

4 $(1 \pm \epsilon)$-Approximate Colored Path Counting

We first present in Section 4.1 a data structure for optimal colored type-2 path counting, which implies the support for colored path reporting. This data structure is further used in our solution to $(1 \pm \epsilon)$-approximate colored path counting. In Section 4.2, we perform random sampling of node colors and construct an extracted tree accordingly. We also determine a condition under which this extracted tree can be used to compute a $(1 \pm \epsilon)$-approximate answer with high probability. Then, in Section 4.3, we show how to combine the techniques in the previous two subsections to design a solution for the case in which the number of distinct colors in a query path is in $[\kappa/2, 2\kappa]$, where κ is an integer parameter specified in the preprocessing stage. Finally, in Section 4.4, we construct a set of data structures from Section 4.3, each for a different parameter κ. We then use 2-approximate colored path counting to determine a range that the exact answer to the query must be in, so that we can use an appropriate building block to compute a $(1 \pm \epsilon)$-approximate answer.

4.1 A New Solution to Colored Type-2 Path Counting

Let $P_{x,x'}$ denote a query path such that x' is an ancestor of x. Consider colored type-2 path counting for $P_{x,x'}$. Our strategy can be described as follows: For each color $c \in C(P_{x,x'})$, we locate the lowest node, ℓ_c, in $P_{x,x'}$ whose color is c, as well as the highest node, h_c, in $P_{x,x'}$ colored in c. Then the frequency of color c in $P_{x,x'}$ is $\texttt{depth}_c(\ell_c) - \texttt{depth}_c(h_c) + 1$. If we precompute the value of $\texttt{depth}_{c(v)}(v)$ for each node v, then, after locating ℓ_c and h_c, we can compute the frequency of color c in $P_{x,x'}$ immediately. The details of finding h_c's and ℓ_c's and extending this method to general query paths are deferred to the full version of this paper. The result is summarized as Lemma 6.

Lemma 6. *Let T be an ordinal tree on n nodes with each node assigned a color from $\{0, 1, \ldots, C-1\}$, where $C \leq n$. An $O(n)$-word data structure can be constructed over T in $O(n \lg n)$ time to support colored type-2 path counting in $O(\texttt{occ})$ time, where* occ *denotes the number of distinct colors in a query path.*

4.2 Random Sampling

We now perform random sampling of node colors and apply tree extraction accordingly. Set $\theta = \frac{6(c_1+3)\lg n}{\epsilon^2 \lg e}$, where e denotes Euler's number and $c_1 \geq 1$ is an arbitrary positive constant. Let $\kappa \in (\theta, n]$ be an integer parameter to be chosen later, and define $\texttt{M} = \theta/\kappa$. We create a random color set C' by choosing each color that appears in T independently at random with probability M. Then we construct a tree extraction T' from T by removing nodes whose colors are not in C' using the approach described in Section 2. All the nodes in T' are assigned their original color in T except for the dummy root; if a dummy root is added, it is uncolored. For each color $c \in \{0, \cdots, C-1\}$, let X_c denote a random variable indicating whether color c is sampled: X_c is set to 1 if c has been sampled and 0 otherwise. Thus, $\texttt{Pr}[X_c = 1] = \texttt{M}$. Furthermore, for an arbitrary path $P_{x,y}$ in T, we define a random variable $X_{x,y} = \sum_{c \in C(P_{x,y})} X_c$. Lemma 7 states the conditions under which $X_{x,y}/\texttt{M}$ is a $(1 \pm \epsilon)$-approximation of $|C(P_{x,y})|$ with high probability; the proof is deferred to the full version of this paper.

Lemma 7. *Consider an arbitrary path $P_{x,y}$ in T. If* $\texttt{occ} \geq \kappa/2$*, where* occ *denotes $|C(P_{x,y})|$, then* $\texttt{Pr}[(1-\epsilon)\texttt{occ} \leq \frac{X_{x,y}}{\texttt{M}} \leq (1+\epsilon)\texttt{occ}] > 1 - \frac{2}{n^{c_1+3}}$.

4.3 Approximate Colored Path Counting over Canonical Paths

To use Lemma 7 and also due to other considerations, we call a path in T *canonical* if the number of colors that appear in the path is in $[\kappa/2, 2\kappa]$, for an integer $\lceil\theta\rceil \leq \kappa \leq C/2$ to be decided later. We first solve the $(1 \pm \epsilon)$-approximate problem for canonical paths:

Lemma 8. *Let T be an ordinal tree on n nodes represented by Lemma 1. With success probability more than $1 - \frac{1}{n^{c_1+1}}$, one can construct a data structure in $O(n \lg n)$ worst-case time to answer $(1 \pm \epsilon)$-approximate colored path counting queries over canonical paths in $O(\frac{1}{\epsilon^2} \lg n)$ worst-case time. The space cost is $O(n \cdot \texttt{M} + n/\lg n)$ words in the expected case (and $O(n)$ words in the worst case).*

Proof. First, we present the data structures. As described in Section 4.2, we choose a random color set and construct a tree extraction T' consisting of nodes whose colors are sampled. Tree T' has $O(n \cdot \mathtt{M})$ expected number of nodes but $O(n)$ nodes in the worst case. We represent T' by Lemma 1 in $O(|T'| \lg C)$ bits to support fast navigation. We also construct the data structure of Lemma 3 over T and T' to support `decompose` in constant time. The data structure uses $O(n)$ bits which is $O(n/\lg n)$ words. Finally, we construct in $O(n \lg n)$ time in the worst case the linear space data structure for colored path reporting queries over T' by applying Lemma 6. The overall space cost is $O(n \cdot \mathtt{M} + n/\lg n)$ words in the expected case, and $O(n)$ words in the worst case. The construction time is bounded by $O(n \lg n)$.

To describe the query algorithm, let $P_{x,y}$ be a canonical query path. Since $|C(P_{x,y})| \geq \kappa/2$, $X_{x,y}/\mathtt{M}$ is a $(1 \pm \epsilon)$-approximate of $|C(P_{x,y})|$ with probability greater than $1 - \frac{2}{n^{c_1+3}}$ by Lemma 7. Since $\mathtt{M}$ is given in preprocessing, we need only compute $X_{x,y}$. Let x' and y' be the nodes of T' returned by $\mathtt{decompose}(x, y)$ in $O(1)$ time. If x' and y' are `null`, no color in $P_{x,y}$ has been sampled, so we set $X_{x,y}$ to be 0. Otherwise, $X_{x,y}$ is either $|C(P_{x',y'})|$ or $|C(P_{x',y'})| - 1$, and we can determine which case it is by performing these steps: If the color of $z = \mathtt{lca}(x, y)$ in T is sampled, then its corresponding node, z', in T' belongs to $P_{x',y'}$. In this case, there is a one-to-one correspondence between the nodes in $P_{x',y'}$ and the nodes in $P_{x,y}$ whose colors are sampled, so $X_{x,y} = |C(P_{x',y'})|$. If the color of z is not sampled, then the node $z'' = \mathtt{lca}(x', y')$ in T' does not correspond to z, but all other nodes in $P_{x',y'}$ correspond to nodes with sampled colors in the query path. Then there are two sub-cases to be considered, depending on whether the color of z'' also happens to appear in $P_{x',y'} \setminus \{z''\}$. If it does, then we have $X_{x,y} = |C(P_{x',y'})|$, and otherwise, $X_{x,y} = |C(P_{x',y'})| - 1$. Navigational operations such as `lca` can be performed over T and T' in constant time, and whether $c(z'')$ appears in $P_{x',y'} \setminus \{z''\}$ can be tested by two path emptiness queries in $O(\lg \lg C)$ time. Therefore, if we know the value of $|C(P_{x',y'})|$, we can compute $X_{x,y}$ in $O(\lg \lg C)$ extra time.

It remains to show how to compute $|C(P_{x',y'})|$. To do it, observe that if $X_{x,y}/\mathtt{M}$ is a $(1 \pm \epsilon)$-approximation, then $X_{x,y} \leq (1+\epsilon) \cdot \mathtt{M} \cdot |C(P_{x,y})| \leq (1+\epsilon) \cdot \mathtt{M} \cdot 2\kappa$. Since $X_{x,y}$ is at least $|C(P_{x',y'})| - 1$, we have $|C(P_{x',y'})| \leq (1+\epsilon) \cdot \mathtt{M} \cdot 2\kappa + 1$. With this, we can apply Lemma 6 to report the distinct colors in $C(P'_{x',y'})$, and instead of reporting all these colors, we stop when the number of reported colors reaches $(1+\epsilon) \cdot \mathtt{M} \cdot 2\kappa + 2$. If this happens, we terminate our query algorithm with failure. Otherwise, the number of colors reported is $|C(P_{x',y'})|$. Since $(1+\epsilon) \cdot \mathtt{M} \cdot 2\kappa + 2 = O(\epsilon^{-2} \lg n)$, this process uses $O(\epsilon^{-2} \lg n)$ time.

By Lemma 7, for an arbitrary query path, our data structure fails to return a $(1 \pm \epsilon)$-approximation with probability $\mathtt{Pr}[|\frac{X}{\mathtt{M}} - \mathtt{occ}| > \epsilon \cdot \mathtt{occ}] < \frac{2}{n^{c_1+3}}$. Since there are $\binom{n}{2}$ different query paths, the probability of constructing a data structure that answers all queries correctly is more than $1 - \binom{n}{2} \cdot \frac{2}{n^{c_1+3}} > 1 - \frac{1}{n^{c_1+1}}$. $\square$

Next, we keep resampling colors and building the structure of Lemma 8 for the sample, until we find a data structure that occupies $O(n \cdot \mathtt{M} + \frac{n}{\lg n})$ words in the worst case and can always return $(1 \pm \epsilon)$-approximations for canonical paths. This process requires $O(n^2 \lg \lg C)$ expected preprocessing time (the analysis is deferred to the full version of this paper). Therefore, we achieve:

Lemma 9. *Let T be an ordinal tree on n nodes represented by Lemma 1. A data structure occupying $O(n \cdot \mathtt{M} + n/\lg n)$ extra words in the worst case can be constructed in $O(n^2 \lg\lg C)$ expected time to support $(1 \pm \epsilon)$-approximate colored path counting over canonical paths in $O(\epsilon^{-2} \lg n)$ worst-case time.*

4.4 Approximate Colored Path Counting over Arbitrary Paths

To solve queries over arbitrary paths, we first represent T by Lemma 1 to support navigational operations. We also construct the data structures of part a) of Theorem 1 to support 2-approximate colored path counting over T. In addition, we build the data structures of Lemma 6 to support colored path reporting. These data structures use $O(n)$ words and can be built in $O(n \lg n)$ time.

Then, for each $i \in [\lceil \lg \theta \rceil, \lceil \lg C \rceil)$, let κ_i be 2^i. We refer to a query path as a *tier-i canonical path* if the number of distinct colors that appear in it is in $[\kappa_i/2, 2\kappa_i]$. For each possible value of i, we apply Lemma 9 to construct a data structure $\mathbb{DS}_i$ to support $(1 \pm \epsilon)$-approximate colored path counting over tier-i canonical paths. Data structure $\mathbb{DS}_i$ uses $O(n\theta/\kappa_i + n/\lg n) = O(n\theta/2^i + n/\lg n)$ words in the worst case and can be constructed in $O(n^2 \lg\lg C)$ expected time. Summing up over all $i \in [\lceil \lg \theta \rceil, \lceil \lg C \rceil)$, the overall space cost of these data structures is $O(n)$ words in the worst case, and they can be constructed in $O(n^2 \lg C \lg\lg C)$ expected time. Theorem 2 summarizes our final result.

Theorem 2. *Let T be an ordinal tree on n nodes with each node assigned a color from $\{0, 1, \ldots, C-1\}$, where $C \le n$. A data structure of $O(n)$ words of space in the worst case can be constructed in $O(n^2 \lg C \lg\lg C)$ expected time to support $(1 \pm \epsilon)$-approximate colored path counting in $O(\epsilon^{-2} \lg n)$ worst-case time.*

Proof. It remains to show the query algorithm. Let $P_{x,y}$ denote the query path. We first use the colored path reporting structure to report up to θ distinct colors in $C(P_{x,y})$. If less than θ colors are reported, then we return the exact number of colors, taking $O(\epsilon^{-2} \lg n)$ time. Otherwise, $\mathtt{occ} > \theta$. In this case, we compute a 2-approximate result, $\mathtt{occ}_a$, in $O(\lg^{\lambda} n)$ time. Then, $\mathtt{occ} \le \mathtt{occ}_a \le 2\mathtt{occ}$. Observe that, for any $i \in [\lceil \lg \theta \rceil, \lceil \lg C \rceil)$, if $\kappa_i \le \mathtt{occ}_a \le 2\kappa_i$, then $\kappa_i/2 \le \mathtt{occ} \le 2\kappa_i$. This allows us to perform a binary search in $O(\lg\lg n)$ time to find the value of i such that $P_{x,y}$ is a tier-i canonical path. Finally, by querying $\mathbb{DS}_i$, we can find a $(1 \pm \epsilon)$-approximation of $\mathtt{occ}$ in $O(\epsilon^{-2} \lg n)$ worst-case time. $\square$

References

1. Alman, J., Williams, V.V.: A Refined Laser Method and Faster Matrix Multiplication. In: 32nd Annual ACM-SIAM Symposium on Discrete Algorithms (2021), 10.1137/1.9781611976465.32
2. Belazzougui, D., Cunial, F., Kärkkäinen, J., Mäkinen, V.: Linear-time String Indexing and Analysis in Small Space. ACM Transactions on Algorithms (2020), 10.1145/3381417
3. Chan, T.M., He, M., Munro, J.I., Zhou, G.: Succinct Indices for Path Minimum, with Applications. Algorithmica (2016), 10.1007/s00453-016-0170-7

4. Chan, T.M., He, Q., Nekrich, Y.: Further Results on Colored Range Searching. In: 36th Annual Symposium on Computational Geometry (2020), 10.4230/LIPICS.SOCG.2020.28
5. Chan, T.M., Larsen, K.G., Pătraşcu, M.: Orthogonal Range Searching on the RAM, Revisited. In: 27th Annual Symposium on Computational Geometry (2011), 10.1145/1998196.1998198
6. Chan, T.M., Nekrich, Y.: Better Data structures for Colored Orthogonal Range Reporting. In: 31st Annual ACM-SIAM Symposium on Discrete Algorithms (2020), 10.1137/1.9781611975994.38
7. Durocher, S., Shah, R., Skala, M., Thankachan, S.V.: Linear-Space data Structures for Range Frequency Queries on Arrays and Trees. Algorithmica (2016), 10.1007/s00453-014-9947-8
8. El-Zein, H., Munro, J.I., Nekrich, Y.: Succinct Color Searching in One Dimension. In: 28th International Symposium on Algorithms and Computation (2017), 10.4230/LIPICS.ISAAC.2017.30
9. Ganguly, A., Munro, J.I., Nekrich, Y., Shah, R., Thankachan, S.V.: Categorical Range Reporting with Frequencies. In: 22nd International Conference on Database Theory (2019), 10.4230/LIPICS.ICDT.2019.9
10. Gao, Y., He, M.: Faster Path Queries in Colored Trees via Sparse Matrix Multiplication and Min-Plus Product. In: 30th Annual European Symposium on Algorithms (2022), 10.4230/LIPICS.ESA.2022.59
11. Gupta, P., Janardan, R., Smid, M.: Further Results on Generalized Intersection Searching Problems: Counting, Reporting, and Dynamization. Journal of Algorithms (1995), 10.1006/jagm.1995.1038
12. He, M., Kazi, S.: Data Structures for Categorical Path Counting Queries. Theoretical Computer Science (2022), 10.1016/j.tcs.2022.10.011
13. He, M., Munro, J.I., Rao, S.S.: Succinct Ordinal Trees Based on Tree Covering. ACM Transactions on Algorithms (2007), 10.1007/978-3-540-73420-8-45
14. He, M., Munro, J.I., Zhou, G.: A Framework for Succinct Labeled Ordinal Trees over Large Alphabets. Algorithmica (2014), 10.1007/s00453-014-9894-4
15. He, M., Munro, J.I., Zhou, G.: Data Structures for Path Queries. ACM Transactions on Algorithms (2016), 10.1145/2905368
16. Jordan, C.: Sur les assemblages de lignes. Journal für die reine und angewandte Mathematik (Crelles Journal) (1869), 10.1515/crll.1869.70.185
17. Kaplan, H., Rubin, N., Sharir, M., Verbin, E.: Efficient Colored Orthogonal Range Counting. SIAM Journal on Computing (2008), 10.1137/070684483
18. Muthukrishnan, S.: Efficient Algorithms for Document Retrieval Problems. In: 13th Annual ACM-SIAM Symposium on Discrete Algorithms (2002), 10.5555/545381.545469
19. Nekrich, Y.: Efficient Range Searching for Categorical and Plain Data. ACM Transactions on Database Systems (2014), 10.1145/2543924
20. Pătraşcu, M.: Lower Bounds for 2-Dimensional Range Counting. In: 39th Annual ACM Symposium on Theory of Computing (2007), 10.1145/1250790.1250797
21. Rahul, S.: Approximate Range Counting Revisited. Journal of Computational Geometry (2021), 10.20382/JOCG.V12I1A3
22. Sadakane, K.: Succinct Data Structures for Flexible Text Retrieval Systems. Journal of Discrete Algorithms (2007), 10.1016/j.jda.2006.03.011
23. Yu, H.: An Improved Combinatorial Algorithm for Boolean Matrix Multiplication. Information and Computation (2018), 10.1016/j.ic.2018.02.006