Succinct data structures for nearest colored node in a tree

Dekel Tsur*

Abstract

We give succinct data structures that store a tree with colors on the nodes. Given a node x and a color α, the structures find the nearest node to x with color α. Our results improve the $O(n \log n)$-bits structure of Gawrychowski et al. [CPM 2016].

Keywords succinct data structures, labeled trees, colored trees.

1 Introduction

In the *nearest colored node* problem the goal is to store a tree with colors on the nodes such that given a node x and a color α, the nearest node to x with color α can be found efficiently. Gawrychowski et al. [12] gave a data structure for this problem that uses $O(n \log n)$ bits and answers queries in $O(\log \log n)$ time, where n is the number of nodes in the tree.

In this paper we give succinct structures for the nearest colored node problem. Our results are given in the following theorem.

Theorem 1. *Let T be a colored tree with n nodes and colors from $[1, \sigma]$, and let P_T be a string containing the colors of the nodes in preorder.*

1. *For $\sigma = o(\log n/(\log \log n)^2)$, for any $k = o(\log n/\log^2 \sigma)$, there is a representation of T that uses $nH_k(P_T) + 2n + o(n)$ bits and answers nearest colored node queries in $O(1)$ time, where $H_k(P_T)$ is the k-th order entropy of P_T.*
2. *For $\sigma = w^{O(1)}$ (where w is the word size), for any function $f(n) = \omega(1)$, there is a representation of T that uses $nH_0(P_T) + 2n + o(n)$ bits and answers nearest colored node queries in $O(f(n))$ time.*
3. *For $\sigma \leq n$, there is a representation of T that uses $nH_0(P_T)+2n+o(nH_0(P_T))+o(n)$ bits and answers nearest colored node queries in $O(\log \frac{\log \sigma}{\log w})$ time.*

Theorem 1 improves both the space complexity and the query time complexity of the structure of Gawrychowski et al. [12].

1.1 Related work

Gawrychowski et al. [12] also considered a dynamic version of the nearest colored node problem in which the colors of the nodes can be changed. For this problem they gave an $O(n \log n)$ bits structure that supports updates and queries in $O(\log n)$ time. They also gave a structure with $O(n \log^{2+\epsilon} n)$ space, optimal $O(\log n/\log \log n)$ query time, and $O(\log^{1+\epsilon} n)$ update time.

*Department of Computer Science, Ben-Gurion University of the Negev. Email: dekelts@cs.bgu.ac.il

Several papers studied data structures for storing colored trees with support for various queries [4, 6, 9, 13, 14, 19, 20]. In particular, the problem of finding the nearest ancestor with color α was considered in [6, 13, 14, 19, 20]. In order to solve the nearest colored node problem, we combine techniques from the papers above and from Gawrychowski et al. [12].

Another related problem is to find an approximate nearest node with color α. This problem has been studied in general graphs [7, 15, 16] and planar graphs [1, 17, 18].

2 Preliminaries

Throughout the paper we assume the tree T is an ordinal tree (for a non-ordinal tree an ordering can be chosen arbitrarily). When we write that a node w is a descendant of v it means that either $w = v$ or w is a proper descendant of v. The same holds for other tree terminology, e.g. ancestor.

A node with color α will be called *α-node*. We also use other α- terms with the appropriate meaning, e.g. an α-descendant of a node v is a descendant of v with color α.

2.1 Tree decomposition

We use the following tree decomposition [2, 3, 8].

Lemma 2. *For a tree T with n nodes and an integer L, there is a collection $\mathcal{D}_{T,L}$ of subtrees of T with the following properties.*

1. *Every edge of T appears in exactly one tree of $\mathcal{D}_{T,L}$.*
2. *The size of every tree in $\mathcal{D}_{T,L}$ is at most L and at least 2.*
3. *The number of trees in $\mathcal{D}_{T,L}$ is $O(n/L)$.*
4. *For every $T' \in \mathcal{D}_{T,L}$, at most two nodes of T' can appear in other trees of $\mathcal{D}_{T,L}$. These nodes are called the* boundary nodes *of T'.*
5. *A boundary node of a tree $T' \in \mathcal{D}_{T,L}$ can be either a root of T' or a leaf of T'. In the latter case the node will be called the* boundary leaf *of T'.*
6. *For every $T' \in \mathcal{D}_{T,L}$, there are two intervals I_1 and I_2 such that a node $x \in T$ is a non-root node of T' if and only if the preorder rank of x is in $I_1 \cup I_2$.*

For a tree T and an integer L we define a tree T^L as follows. Construct a tree decomposition $\mathcal{D}_{T,L}$ according to Lemma 2. If the root r of T appears in several trees of $\mathcal{D}_{T,L}$, add to $\mathcal{D}_{T,L}$ a tree that consists of r. The tree T^L has a node v_S for every tree $S \in \mathcal{D}_{T,L}$. For two trees $S_1, S_2 \in \mathcal{D}_{T,L}$, v_{S_1} is the parent of v_{S_2} in T^L if and only if the root of S_2 is equal to the boundary leaf of S_1.

For a node v_S in T^L let $V(v_S)$ be the set of the nodes of S excluding the root. For a node $v \in T$, we denote by $\mathrm{map}(u)$ the node of T^L for which $u \in V(\mathrm{map}(u))$.

2.2 Data structures

In this section we describe the data structure we use in this paper.

Ferragina and Venturini [10] gave a data structure for storing a string S of length n using $nH_k(S) + O(nk \log \sigma / \log_\sigma n) + O(n \log \log n / \log_\sigma n)$ bits (for $k = o(\log_\sigma n)$) that supports accessing a substring of S of length l in $O(1 + l/\log_\sigma n)$ time.

A rank-select structure on a string S is a structure that supports the following queries. $\mathrm{rank}_\alpha(S,i)$ returns the number of times the character α appears in the substring $S[1..i]$, and $\mathrm{select}_\alpha(S,i)$ returns the index of the i-th occurrence of α in S. We use the following results of Belazzougui and Navarro [5]. For $\sigma = w^{O(1)}$, there is a rank-select structure that uses $nH_0(S) + o(n)$ bits and answers queries in $O(1)$ time. For $\sigma \leq n$, there is a rank-select structure that uses $nH_0(S) + o(nH_0(S)) + o(n)$ bits. The time for rank queries is $O(\log \frac{\log \sigma}{\log w})$ and the time for select queries is $O(1)$.

An RMQ structure on an array A supports the query $\mathrm{RMQ}(A,i,j)$ which returns the index of the minimum element in the sub-array $A[i..j]$. Fischer and Heun [11] gave an RMQ structure that uses $O(n/c(n))$ bits and answers RMQ queries in $O(c(n))$ time, for every $c(n) = O(n^\epsilon)$ for an arbitrary constant $0 < \epsilon < 1$. The structure requires access to A for answering RMQ queries.

There are several succinct structures for storing ordinal trees. Here we use the structure of Farzan and Munro [8]. The structure stores the balanced parenthesis string of the tree (uncompressed) using $2n$ bits, and additional information that takes $o(n)$ bits. The structure supports various tree queries. The queries we use in this paper are as follows (we assume that the nodes are identified by their preorder numbers). $\mathrm{lca}(u,v)$ returns the lowest common ancestor of u and v, $\mathrm{rightmost\_leaf}(v)$ returns the rightmost descendant leaf of v, and $\mathrm{depth}(v)$ returns the depth of v.

Suppose T is a tree with n nodes and some of its nodes are marked. Let $v_1, \ldots, v_m$ be the marked nodes, in preorder. Apply the tree decomposition of Section 2.1 on T with parameter L and obtain the tree T^L. In [20] we showed a data structure that stores the tree T^L using $O((n/L)\log L)$ bits and supports the following queries in constant time. (1) Given i, compute $\mathrm{map}(v_i)$ (2) Given a node $v_S \in T^L$, compute the intervals I_1, I_2 such that $v_i \in V(v_S)$ if and only if $i \in I_1 \cup I_2$ (these intervals exists due to Property 6 of Lemma 2) (3) Given nodes $u, v \in T^L$, compute $\mathrm{lca}(u,v)$.

3 Proof of part 1 of Theorem 1

Our structure is similar to the labeled tree structure of He et al. [14]. As in [14], the data structure stores P_T in the compressed structure of Ferragina and Venturini [10], the tree T without the colors in the data structure of Farzan and Munro [8], and additional structures described below. Recall that the structure of T keeps the balanced parenthesis string of T.

Using the tree decomposition of Lemma 2, the tree T is partitioned into *mini-trees* of size at most $L' = \lceil \log^2 n \rceil$, and every mini-tree is decomposed into *micro-trees* of size at most $L = \lceil \frac{1}{2} \log_\sigma n \rceil$. From Property 4 of the tree decomposition we have that if y is the nearest α-node to a node x and y is not in the mini-tree that contains x, then the path from x to y passes through a boundary node of the mini-tree that contains x. Similar property holds for micro-trees. Based on the observation above, the data structure stores the following additional information. (1) A lookup table that contains for every colored tree S of size at most L, every node x in S, and every color α, the α-node in S that is nearest to x (if such nodes exist). (2) For every mini-tree S' and every color α, the α-nodes in T that are nearest to the root of S' and to the boundary leaf of S'. (3) For every micro-tree S and every color α, the α-nodes in the mini-tree that contains S that are nearest to the root of S and to the boundary leaf of S. The space for storing P_T is $nH_k(P_T) + o(n)$ bits, and the space for storing T is $2n + o(n)$ bits. The space for the lookup table is $O(2^{2L}\sigma^L L\sigma \log L) = o(n)$, the space for the mini-tree information is $O((n/L')\sigma \log n) = o(n)$, and the space for the micro-tree information is

$O((n/L)\sigma \log L') = O(\sigma \log \sigma \cdot n \log\log n / \log n) = o(n)$.

Given a query x, α, we obtain up to five candidates for the nearest α-node to x: the α-node in the micro-tree that contains x that is nearest to x, and the four α-nodes stored for the micro-tree and mini-tree that contain x. Note that in order to use the lookup-table, we need to generate the balanced parenthesis string of the micro-tree that contains x, and a sequence containing the colors of the nodes in this tree in preorder. This can be done in $O(1)$ time due to Property 6 of Lemma 2. The distance between x and every candidate y can be computed in $O(1)$ time (the distance is $\text{depth}(x) + \text{depth}(y) - 2 \cdot \text{depth}(\text{lca}(x, y))$). Therefore, the query is answered in $O(1)$ time.

4 Proof of parts 2 and 3 of Theorem 1

Our data structure stores the rank-select structure of Belazzougui and Navarro [5] on P_T, the tree structure of Farzan and Munro [8] on the tree T without the colors, and additional information that will be described below.

Our structure is similar to the structure of Gawrychowski et al. [12]. We next give a short description of the structure of [12]. For a color α, let Z_α be the set of all α-nodes and their ancestors, and let Y_α be the set of all nodes $x \in Z_\alpha$ such that either x has color α, or x has at least two children in Z_α. We define a tree T_α whose nodes are Y_α, and x is the parent of y in T_α if and only if x is the lowest proper ancestor of y that is in Y_α.

Let x, α be a query. If x is an α-node, the answer to the query is x itself, so assume for the rest of the section that x is not an α-node (checking whether x is an α-node is done using rank and select queries on P_T). We define nodes in the tree that will be used for answering the query: z is the lowest ancestor of x which has an α-descendant that is not a descendant of x. Moreover, y (resp., y_2) is the nearest descendant (resp., ancestor) of z which is in Y_α. Note that $y = y_2 = z$ if $z \in Y_\alpha$.

The nearest α-node to x is either (1) the nearest α-descendant of x, (2) the nearest α-node to y, or (3) the nearest α-node to y_2. Based on this observation, the structure of Gawrychowski et al. [12] finds these three candidate nodes, and returns the one that is closest to x. Our structure is based on a slightly different observation: The nearest α-node to x is either (1) the nearest α-descendant of x, (2) the nearest α-descendant of y, or (3) the nearest α-non-descendant of y.

We next describe the approach we use for finding the node y, which is different than the one used in [12]. We assume that the nodes of T are $1, \ldots, n$, according to preorder. For a node v, let $\text{succ}_\alpha(v)$ (resp., $\text{pred}_\alpha(v)$) be the minimum (resp., maximum) w such that $w \geq v$ (resp., $w \leq v$) and w is an α-node. We also define $\text{succ-nd}_\alpha(v)$ to be the minimum w such that $w > v$, w is an α-node, and w is not a descendant of v. The following lemma shows how to efficiently find z.

Lemma 3. *Let x_* be the node from $\{\text{pred}_\alpha(x), \text{succ-nd}_\alpha(x)\}$ that minimizes the distance of $\text{lca}(x_*, x)$ to x (if $\text{pred}_\alpha(x)$ does not exist then $x_* = \text{succ-nd}_\alpha(x)$ and vice versa). Then, $z = \text{lca}(x_*, x)$.*

Proof. First, $\text{lca}(x_*, x)$ is an ancestor of x and an ancestor of an α-node that is not a descendant of x (the node x_*). Now, consider some ancestor x' of x that is closer to x than $\text{lca}(x_*, x)$. In other words, $\text{lca}(x_*, x)$ is a proper ancestor of x'. By definition $\text{lca}(\text{pred}_\alpha(x), x)$ is an ancestor of x_*, and therefore $\text{lca}(\text{pred}_\alpha(x), x)$ is a proper ancestor of x'. Since $\text{pred}_\alpha(x) < x$, it follows that $\text{pred}_\alpha(x) < x'$. Therefore, $\text{pred}_\alpha(x) < x''$ for every descendant x'' of x'. Similarly, $\text{succ-nd}_\alpha(x) > x''$ for every descendant x'' of x'. From the

definition of $\text{pred}_\alpha(x)$ and $\text{succ-nd}_\alpha(x)$, x' does not have an α-descendant that is not a descendant of x. Since this is true for every x', the lemma follows. ■

Finding $\text{pred}_\alpha(x)$ and $\text{succ-nd}_\alpha(x)$ can be done using rank and select queries on P_T. Moreover, computing $\text{lca}(x', x)$ and its distance to x can be done using the tree data structure of T.

The next lemma shows how to find y.

Lemma 4. *Let v be a node in Z_α, and let w be the nearest descendant of v that is in Y_α. Then, $w = \text{lca}(\text{succ}_\alpha(v), \text{pred}_\alpha(\text{rightmost\_leaf}(v)))$.*

Proof. Suppose first that w does not have color α. Since $w \in Y_\alpha$, w has at least two children that are in Z_α. Let w', w'' be the first and last children of w that are in Z_α, respectively. Every α-descendant of v is also a descendant of w. It follows that $\text{succ}_\alpha(v)$ is a descendant of w' and $\text{pred}_\alpha(\text{rightmost\_leaf}(v))$ is a descendant of w''. Thus, $\text{lca}(\text{succ}_\alpha(v), \text{pred}_\alpha(\text{rightmost\_leaf}(v))) = \text{lca}(w', w'') = w$.

If w has color α then $\text{succ}_\alpha(v) = w$ and $\text{pred}_\alpha(\text{rightmost\_leaf}(v))$ is a descendant of w. Therefore, $\text{lca}(\text{succ}_\alpha(v), \text{pred}_\alpha(\text{rightmost\_leaf}(v))) = w$. ■

We now show how to find the nearest α-non-descendant of y. We use the approach of [20]. For the case $\sigma = w^{O(1)}$ let $L = f(n)$ where f is an integer function that satisfies $f(n) = \omega(1)$ and $f(n) = O(\log n)$, and for larger σ let $L = \lceil \log \frac{\log \sigma}{\log w} \rceil$. We say that a color α is *frequent* if the number of α-nodes is at least L. Given a query x, α, finding whether α is frequent can be done by performing a rank query on P_T. If α is non-frequent, the query can be answered in $O(L)$ time by enumerating all α-nodes (by computing $\text{select}_\alpha(P_T, k)$ for all k) and computing the distance between x and each enumerated node. For the rest of the section, we describe how to handle queries in which the color is frequent.

We apply the tree decomposition of Section 2.1 on T_α with parameter L and obtain the tree T_α^L. For a node v_S in T_α^L let $V_\alpha(v_S)$ be the set of α-nodes in $V(v_S)$. Due to the properties of the tree decomposition we have that for two nodes $u, v \in Y_\alpha$, $\text{lca}(u, v)$ is a node in the tree S in the decomposition for which $v_S = \text{lca}(\text{map}(u), \text{map}(v))$ (if $\text{lca}(u, v)$ is not the root of S then $\text{map}(\text{lca}(u, v)) = v_S$ and otherwise $\text{map}(\text{lca}(u, v)) = \text{parent}(v_S)$).

We assign weights to each node v_S of T_α^L as follows.

- $w_1(v_S)$ is the distance between the boundary nodes of S.
- $w_2(v_S)$ (resp., $w_3(v_S)$) is the shortest distance between the root (resp., boundary leaf) of S and a node in $V_\alpha(v_S)$. If $V_\alpha(v_S) = \emptyset$ then $w_2(v_S) = w_3(v_S) = \infty$.

Let v and v' be two nodes of T_α^L, and let P be the path from v to v'. The *weighted distance from v to v'* is the sum of the following values.

1. $w_1(u)$ for every node $u \neq v, v'$ which is on P and the parent of u is also on P.
2. $w_2(v')$ if v' is not an ancestor of v.
3. $w_3(v')$ if v' is an ancestor of v.

The descendant (resp., non-descendant) of v with minimum weighted distance of v (with ties broken arbitrarily) will be denoted $\text{wnearest-d}(v)$ (resp., $\text{wnearest-nd}(v)$).

Our approach for finding the nearest α-non-descendant of y is based on the following observation.

Observation 5. *Let $v_S \neq v_{S'}$ be two nodes of T^L_α, and u be a node in S. The shortest distance between u and a node in $V_\alpha(v_{S'})$ is equal to the weighted distance from v_S to $v_{S'}$ plus the distance between u and the boundary leaf of S if $v_{S'}$ is a descendant of v_S, and the distance between u and the root of S otherwise.*

Corollary 6. *Let v_S be a node of T^L_α. Let u be a node in S and u' be the nearest α-non-descendant of u. If u is not the root of S then $u' \in V_\alpha(v_S) \cup V_\alpha(\text{wnearest-nd}(v_S)) \cup V_\alpha(\text{wnearest-d}(v_S))$ and otherwise $u' \in V_\alpha(\text{parent}(v_S)) \cup V_\alpha(\text{wnearest-nd}(\text{parent}(v_S)))$.*

Based on Corollary 6, the algorithm for finding the nearest α-non-descendant of y is as follows.

1. Find z using Lemma 3.
2. Compute $y = \text{lca}(\text{succ}_\alpha(z), \text{pred}_\alpha(\text{rightmost\_leaf}(z)))$.
3. Compute $y' = \text{lca}(\text{map}(\text{succ}_\alpha(z)), \text{map}(\text{pred}_\alpha(\text{rightmost\_leaf}(z))))$ and $y'' = \text{parent}(y')$.
4. Enumerate all nodes in $V_\alpha(y') \cup V_\alpha(\text{wnearest-nd}(y')) \cup V_\alpha(\text{wnearest-d}(y')) \cup V_\alpha(y'') \cup V_\alpha(\text{wnearest-nd}(y''))$ that are not descendants of y, compute their distances to y, and return the node that is nearest to y.

In order to perform steps of the algorithm efficiently, we store the following.

- For each tree T_α we store the tree T^L_α using the data structure from [20] (see Section 2.2), where the marked nodes of T_α are the α-nodes.
- A lookup table that contains for every tree S of size at most $L' = \lfloor \frac{1}{8} \log n / \log L \rfloor$ with weights w_1, w_2, w_3 on its nodes, a vector A_S that contains for every node u in S, the node u_2 in S whose weighted distance from u is minimum.
- Let T' be a tree obtained by connecting the roots of the trees T^L_α to a new node whose w_1, w_2, w_3 weights are ∞. Apply the tree decomposition of Section 2.1 on T'. Namely, T' is partitioned into micro-trees of size at most L'. For every micro-tree S we store the nodes in T' with minimum weighted distances to the root of S and to the boundary leaf of S.
- For every micro-tree S of T', a pointer to the vector A_S in the lookup table.

Using this information, the queries map$(\cdot)$, wnearest-d$(\cdot)$ and wnearest-nd$(\cdot)$ are performed in $O(1)$ time. Therefore, the nearest α-non-descendant of y can be found in time $\omega(1)$ for $\sigma = w^{O(1)}$ and $O(\log \frac{\log \sigma}{\log w})$ for larger σ.

Finally, we describe how to find the nearest α-descendant of a node v (recall that we need to find the nearest α-descendants of x and y). For every frequent color α, let A_α be an array containing the depths of the α-nodes in preorder. Let A be the concatenation of the A_α arrays. We store the RMQ structure of Fischer and Heun [11] on A, with $c(n) = L$. Note that our structure does not store A. However, any element of A can be computed in $O(1)$ time using select query on P_T and depth query on T.

To find the nearest α-descendant of a node v, find the range $[i, j]$ of preorder ranks of the α-descendants of v as follows: $i = \text{rank}_\alpha(P_T, v-1)+1$ and $j = \text{rank}_\alpha(P_T, \text{rightmost\_leaf}(v))$. Then perform an RMQ$(A_\alpha, i, j)$ query and return the corresponding node.

We now analyze the space complexity of our structure. The space of the rank-select structure on P_T is $nH_0(P_T) + o(n)$ for $\sigma = w^{O(1)}$ and $nH_0(P_T) + o(nH_0(P_T)) + o(n)$ for

larger σ. The space for the tree T is $2n + o(n)$. We now show that all the additional structures use $o(n)$ bits. The space for the trees T_α^L is $O(n/L \cdot \log L) = o(n)$ bits. The number of unweighted trees of size at most L' is at most $2^{2L'}$. The number of ways to assign weights w_1, w_2, w_3 to the nodes of a tree of size at most L' (where each weight is from $\{0, \ldots, L-1, \infty\}$) is at most $(L+1)^{3L'}$. Therefore, the number of entries in the lookup table is at most $2^{2L'} \cdot (L+1)^{3L'} < 2^{4L' \log L} \leq 2^{\frac{1}{2} \log n} = \sqrt{n}$. Each entry takes $O(L' \log L') = O(\log n \log \log n)$ bits. Therefore, the space of the lookup table is $o(n)$. The information stored for the micro-trees of T' takes $O((n/L)/L' \cdot \log n) = O((n/L) \log L) = o(n)$ bits. The space of the RMQ structure is $O(n/L)$. Therefore, the space of the additional structures is $o(n)$.

References

[1] I. Abraham, S. Chechik, R. Krauthgamer, and U. Wieder. Approximate nearest neighbor search in metrics of planar graphs. In *Proc. 18th APPROX/RANDOM*, volume 40, 2015.

[2] S. Alstrup, J. Holm, K. de Lichtenberg, and M. Thorup. Minimizing diameters of dynamic trees. In *Proc. 24th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 270–280, 1997.

[3] S. Alstrup, Jens J. P. Secher, and M. Spork. Optimal on-line decremental connectivity in trees. *Information Processing Letters*, 64(4):161–164, 1997.

[4] J. Barbay, A. Golynski, J. I. Munro, and S. S. Rao. Adaptive searching in succinctly encoded binary relations and tree-structured documents. *Theoretical Computer Science*, 387(3):284–297, 2007.

[5] D. Belazzougui and G. Navarro. Optimal lower and upper bounds for representing sequences. *ACM Transactions on Algorithms*, 11(4):31:1–31:2, 2015.

[6] P. Bille, P. H. Cording, and I. L. Gørtz. Compressed subsequence matching and packed tree coloring. *Algorithmica*, 77(2):336–348, 2017.

[7] S. Chechik. Improved distance oracles and spanners for vertex-labeled graphs. In *Proc. 20th European Symposium on Algorithms (ESA)*, pages 325–336, 2012.

[8] A. Farzan and J. I. Munro. A uniform paradigm to succinctly encode various families of trees. *Algorithmica*, 68(1):16–40, 2014.

[9] P. Ferragina, F. Luccio, G. Manzini, and S. Muthukrishnan. Compressing and indexing labeled trees, with applications. *J. of the ACM*, 57(1), 2009.

[10] P. Ferragina and R. Venturini. A simple storage scheme for strings achieving entropy bounds. *Theoretical Computer Science*, 372(1):115–121, 2007.

[11] J. Fischer and V. Heun. Space-efficient preprocessing schemes for range minimum queries on static arrays. *SIAM J. on Computing*, 4(2):465–492, 2011.

[12] P. Gawrychowski, G. M. Landau, S. Mozes, and O. Weimann. The nearest colored node in a tree. In *Proc. 27th Symposium on Combinatorial Pattern Matching (CPM)*, pages 25:1–25:12, 2016.

[13] R. F. Geary, R. Raman, and V. Raman. Succinct ordinal trees with level-ancestor queries. *ACM Transactions on Algorithms*, 2(4):510–534, 2006.

[14] M. He, J. I. Munro, and G. Zhou. A framework for succinct labeled ordinal trees over large alphabets. *Algorithmica*, 70(4):696–717, 2014.

[15] D. Hermelin, A. Levy, O. Weimann, and R. Yuster. Distance oracles for vertex-labeled graphs. In *Proc. 38th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 490–501, 2011.

[16] J. Łacki, J. Oćwieja, M. Pilipczuk, P. Sankowski, and A. Zych. The power of dynamic distance oracles: Efficient dynamic algorithms for the steiner tree. In *Proc. 47th Symposium on Theory of Computing (STOC)*, pages 11–20, 2015.

[17] M. Li, C. C. C. Ma, and L. Ning. $(1+\varepsilon)$-distance oracles for vertex-labeled planar graphs. In *Proc. 10th International Conference on Theory and Applications of Models of Computation (TAMC)*, pages 42–51, 2013.

[18] S. Mozes and E. E. Skop. Efficient vertex-label distance oracles for planar graphs. In *Proc. 13th International Workshop on Approximation and Online Algorithms (WAOA)*, pages 97–109, 2015.

[19] S. Muthukrishnan and M. Müller. Time and space efficient method-lookup for object-oriented programs. In *Proc. 7th Symposium on Discrete Algorithms (SODA)*, volume 96, pages 42–51, 1996.

[20] D. Tsur. Succinct representation of labeled trees. *Theoretical Computer Science*, 562:320–329, 2015.