Dualities in Tree Representations

Rayan Chikhi
CNRS, Université de Lille, CRIStAL, Lille, France
rayan.chikhi@univ-lille1.fr

Alexander Schönhuth
Centrum Wiskunde & Informatica, Amsterdam, The Netherlands
alexander.schoenhuth@cwi.nl

Abstract

A characterization of the tree T^* such that $\mathrm{BP}(T^*) = \overleftrightarrow{\mathrm{DFUDS}(T)}$, the reversal of $\mathrm{DFUDS}(T)$ is given. An immediate consequence is a rigorous characterization of the tree $\hat{T}$ such that $\mathrm{BP}(\hat{T}) = \mathrm{DFUDS}(T)$. In summary, BP and DFUDS are unified within an encompassing framework, which might have the potential to imply future simplifications with regard to queries in BP and/or DFUDS. Immediate benefits displayed here are to identify so far unnoted commonalities in most recent work on the Range Minimum Query problem, and to provide improvements for the Minimum Length Interval Query problem.

2012 ACM Subject Classification Mathematics of computing → Trees

Keywords and phrases Data Structures, Succinct Tree Representation, Balanced Parenthesis Representation, Isomorphisms

Digital Object Identifier 10.4230/LIPIcs.CPM.2018.18

Related Version A full version with proofs is available at [4], `https://arxiv.org/abs/1804.04263`.

Acknowledgements The authors are grateful to Hélène Touzet for helpful discussions, and to CPM reviewers for insightful comments, providing Figure 2 and the reference to Davoodi *et al* [5].

1 Motivation

Given an array $A[1, n]$ with elements from a totally ordered set, the Range Minimum Query (RMQ) problem is to provide a data structure that on input positions $1 \leq i \leq j \leq n$ returns

$$\mathsf{rmq}_A(i, j) := \min\{A[k] \mid i \leq k \leq j\}. \tag{1}$$

In [9], Fischer and Heun presented the first data structure that uses $2n + o(n)$ bits and answers queries in O(1) time (in fact, without accessing A). They first construct a tree $T[A]$ (the 2D-Min-Heap of A). Then they observe that in a certain parenthesis representation of $T[A]$ (DFUDS), the following query leads to success for computing $\mathsf{rmq}_A(i, j)$ (where 0 and 1 refer to closing and opening parentheses in $\mathrm{DFUDS}(T[A])$, respectively):

$$w_1 \leftarrow \mathsf{rmq}_D(\mathsf{select}_0(i+1), \mathsf{select}_0(j)) \tag{2}$$
$$\text{if} \quad \mathsf{rank}_0(\mathsf{open}(w_1)) = i \text{ then return } i \tag{3}$$
$$\text{else} \quad \text{return } \mathsf{rank}_0(w_1) \tag{4}$$

where rmq_D refers to performing a range minimum query on the array $D[x] := \mathsf{rank}_1(x) - \mathsf{rank}_0(x)$ where x indexes parentheses in $\mathrm{DFUDS}(T[A])$, and 1 and 0 represent opening and

closing parentheses, respectively. $\mathsf{open}(w_1)$ returns the position of the opening parenthesis matching the one closing at position w_1. Note that $D[x] - D[x-1] \in \{-1, +1\}$ for all $x \in \{2, ..., 2N\}$, which turns rmq_D into an easier problem ($\pm$1-RMQ), as was shown in [1].

Most recently, Ferrada and Navarro suggested an alternative approach which leads to a shorter, hence faster query procedure [8]. They construct a tree $\widehat{T[A]}$ that results from a systematic while non-trivial transformation of the edges of $T[A]$ (the number of non-root nodes N remains the same). They observed that in $\mathrm{BP}(\widehat{T[A]})$ the following simpler query computes $\mathsf{rmq}_A(i, j)$:

$$w_2 \leftarrow \mathsf{rmq}_D(\mathsf{select}_0(i), \mathsf{select}_0(j)) \tag{5}$$

$$\text{return } \mathsf{rank}_0(w_2) \tag{6}$$

The *major motivation of our treatment* is the observation – which passes unnoted in both [8, 9] – that

$$\mathrm{DFUDS}(T[A]) = \mathrm{BP}(\widehat{T[A]}) \tag{7}$$

So, the shorter query raised by Ferrada and Gonzalez would have worked for Fischer and Heun as well. It further raises the question whether there are principles by which to transform trees T into trees $\hat{T}$ such that

$$\mathrm{DFUDS}(T) = \mathrm{BP}(\hat{T}) \tag{8}$$

and, if so, what these principles look like. Here, we thoroughly investigate related questions so as to obtain conclusive insight. We will show that the respective trees and their possible representations can be juxtaposed in terms of *a new duality for tree representations*. In doing so, we will obtain a proof for (7) as an easy corollary (to consolidate our findings, we also give a direct proof that [8]'s query also would have worked for [9] in the Appendix of the full version [4]). In summary, our treatment puts BP and DFUDS into a unifying context.

1.1 Related Work

RMQ's. The RMQ problem has originally been anchored in the study of Cartesian trees [21], because it is related to computing the least common ancestor (LCA) of two nodes in a Cartesian tree derived from A [10], further complemented by the realization that any LCA computation can be cast as an $\pm$1-RMQ problem [3] for which subsequently further improvements were raised [15, 19]. Fischer and Heun finally established the first structure that requires $2n + o(n)$ space and $O(1)$ time (without accessing A) [9], establishing an anchor point for many related topics (e.g. [16, 17]), which justified to strive for further improvements [8, 11].

Isomorphisms. For their latest (and likely conclusive) improvements, [8] made use of an isomorphism between binary and general ordinary trees, presented in [15], and successfully experiment with certain variations on the ground theme of this isomorphism, to finally obtain the above-mentioned $\widehat{T[A]}$. Here, we provide an explicit treatment of these trees, which [8] are implicitly making use of. From this point of view, we provide a rigorous re-interpretation of the treatments [8, 9] and the links drawn with [15] therein. Finally, note that [5] further expands on [15].

BP and DFUDS. The BP representation was first presented in [13] and developed further in many ways (e.g. [15]). Since neither the BP nor the LOUDS [6, 13] representations allow for a few basic operations relating to children and subtrees, the DFUDS representation was presented as an improvement in this regard [2, 14]. A tree-unifying approach different to ours was proposed by Farzan *et al* [7]. [5] observes relationships between BP and DFUDS and proves them via the (above-mentioned) isomorphism by [15]. Since our treatment avoids binary trees altogether, it establishes a more direct approach to identifying dualities between ordinal trees than [5].

1.2 Notation

Trees. Throughout, we consider rooted, ordered trees $T = (V, E)$ (with nodes $V = V[T]$ and (directed) edges $E = E[T]$) with root r. For the sake of notational convenience (following standard abuse of tree notation), we will write $v \in T$ instead of $v \in V[T]$ and $T_1 \subset T_2$ for $V[T_1] \subset V[T_2]$; note that induced subgraphs do not play a relevant role in this treatment. By definition of ordered trees, *siblings*, that is nodes sharing their parent node are ordered, implying the notions of left, right, immediate right, immediate left siblings. By $rmc_T(v)$, we denote the rightmost child of a node v in T if it exists (if T is understood, we write $rmc(v)$). Similarly, we denote by $ils_T(v)$ (or $ils(v)$ if T is understood) the immediate left sibling of v in T if it exists. For two siblings, $u < v$ means that u is left of v. As usual, the partial order on siblings can be extended to a full order, ordering all $v \in T$, by depth-first-traversal (or breadth-first-traversal) logic, for example; here, by default, we write $u <_T v$ (or $u < v$ if T is understood) if u comes before v in the depth-first traversal of T. We write $u = \mathsf{pa}(v)$ indicating that u is the parent of v, that is (u, v) is a directed edge in T.

Parenthesis Based Tree Representations. In the following, we will deal with parenthesis based representations for trees, which are vectors of opening parentheses '(' and closing parentheses ')'. The number of opening parentheses will match the number of closing parentheses, thereby for a tree T, each node $v \in T$ will be represented by a pair of opening and closing parentheses, for which we write $\mathrm{OP}(v)$ and $\mathrm{CP}(v)$, respectively.

The *Balanced Parenthesis (BP)* representation $\mathrm{BP}(T)$ (e.g. [13, 15]) is built by traversing T in depth-first order, writing an opening parenthesis when reaching a node for the first time, and writing a closing parenthesis when reaching a node for the second time. By depth-first order logic, this yields a balanced representation, meaning that the number of opening matches the number of closing parentheses (see Figure 1). By default, a node is identified with its opening parenthesis $\mathrm{OP}(v)$.

The *Depth-First Unary Degree Sequence (DFUDS)* representation $\mathrm{DFUDS}(T)$ [2] is again obtained by traversing T in depth-first order, but, when reaching a node with d children for the first time, writing d opening parentheses and one closing parenthesis (and writing no parentheses when reaching it for the second time). This sequence of parentheses becomes balanced when appending an opening parenthesis at the beginning. It is further convenient to identify a node with the parenthesis preceding the block of opening parentheses that represent its children[1], which for all non-root nodes is a closing parenthesis. In other words, in DFUDS, the i-th closing parenthesis reflects the i-th non-root node in DFT order. Note that,

[1] Literature references are ambiguous about the exact choice of parenthesis. None of the alternative choices, like the first opening parenthesis or the closing parenthesis following the block of opening parentheses, would lead to any real complications also in our treatment.

according to this definition, when matching opening parentheses with closing parentheses in a balanced manner, the opening parentheses in one block refer to the children of the closing parenthesis preceding the block from right to left.

Rank/Select/Open/Close. In the following, we will treat parenthesis vectors as bitvectors, where opening and closing parentheses are identified with 1 and 0. Let $B \in \{0,1\}^n$ be a bitvector and $x \in \{1,...,n\}$ (for enhanced exposition, running indices run from 1 to n). Then $\mathsf{rank}_{B,0}(x), \mathsf{rank}_{B,1}(x)$ are defined to be the number of 0's or 1' in B up to (and including) $B[x]$. Further, $\mathsf{select}_{B,0}(i), \mathsf{select}_{B,1}(i)$ are defined to be the position of the i-th 0 or 1 in B (if this exists). We omit the subscript B and write $\mathsf{rank}_0(x), \mathsf{rank}_1(x), \mathsf{select}_0(i), \mathsf{select}_1(i)$ if the choice of B is evident. As a relevant example (see (5)), for DFUDS(T) and $v \in T$, we have $\mathrm{CP}(v) = \mathsf{select}_0(i)$ if and only if $\mathrm{DFT}(v) = i+1$, that is v is the $i+1$-th node in depth-first traversal order, also counting the root. We further write $\mathsf{open}(x)$ and $\mathsf{close}(x)$ to identify the matching partner in a (balanced parenthesis) bitvector, that is $\mathsf{open}(x)$ for a position x in B with $B[x] = 0$ is the position of the 1 matching x and vice versa for $\mathsf{close}(x)$.

1.3 Outline of Sections

We will start with the definition of a dual tree T^* of T in section 2; according to this definition, T^* is a directed graph, so we still have to prove that T^* is a tree, which we will do immediately afterwards. We proceed by proving $(T^*)^* = T$, arguably necessary for a well-defined duality. In section 2.1, we then show how to decompose our duality into subdualities by introducing the definition of a reversed tree $\overleftrightarrow{T}$. We conclude by providing the definition of $\hat{T}$ as the reversed dual tree; without being able to provide a proof at this point, note that $\hat{T}$ will turn out to be the tree from (8).

In section 3, we provide the definition of a primal-dual ancestor, which is crucial for re-interpreting RMQ's in terms of the notions of duality provided here. Upon having proven the unique existence of the primal-dual ancestor in theorem 13, we re-interpret RMQ's, and beyond that not only re-interpret, but also improve on running minimal length interval queries (MLIQ's) both in terms of space requirements and query counts.

We will finally prove our main theorem in section 4.

▶ **Theorem 1.** *Let T be a tree and let the reversal $\overleftrightarrow{B}$ of a bitvector B be defined by $\overleftrightarrow{B}[x] := 1-B[n-x+1]$, $\forall x \in \{1,\ldots,n\}$. Then*

$$\mathrm{BP}(T) = \overleftrightarrow{\mathrm{DFUDS}(T^*)}. \tag{9}$$

Returning to [8], we will finally demonstrate that (7), our motivating insight, indeed holds.

2 Tree Duality: Definition

▶ **Definition 2** (Dual tree)**.** Let T be a tree. The dual tree T^* of T is a directed graph that has the same vertices as T. Edges and order (among nodes sharing a parent) are given by the following rules, where we write $\mathsf{pa}^*(v)$ for the parent of v in T^*:

- **Rule 1a**: The root r of T is also the root of T^*, that is r has no parent also in T^*.
- **Rule 1b**: If $v = rmc_T(r)$ then also $v = rmc_{T^*}(r)$, implying in particular that $\mathsf{pa}^*(v) = r$.
- **Rule 2**: If $v = rmc_T(u)$ with $u \neq r$, then $v = ils_{T^*}(u)$, implying that $\mathsf{pa}^*(v) = \mathsf{pa}^*(u)$.
- **Rule 3**: If $v = ils_T(u)$, then $v = rmc_{T^*}(u)$, implying that $\mathsf{pa}^*(v) = u$.

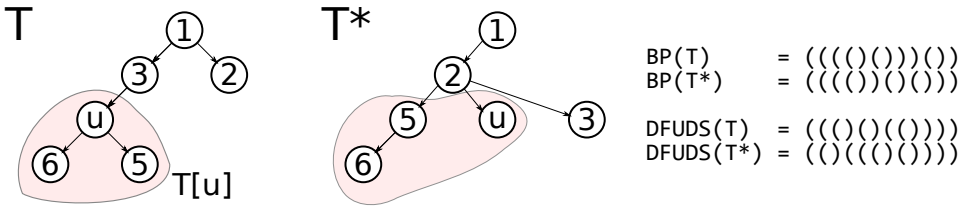

Figure 1 A tree and its dual, along with the BP and DFUDS representations. A subtree $T[u]$ is also highlighted, along with the corresponding nodes in the dual.

▶ Remark. Rules 1a, 1b, 2 and 3 immediately imply that T^* is a directed graph where each node other than r has one parent. Note that the existence of a parent due to Rule 2 is guaranteed by induction on the depth of a node in T, where Rule 1b makes the start.

▶ Remark. It is similarly immediate to observe that there is a well-defined order among nodes that share a parent. It suffices to notice that in T^* each node either is a rightmost child (Rules 1b, 3), or it is the (unique) immediate left sibling of another node (Rule 2).

All nodes but r have exactly one (incoming) edge, which implies $|E| = |V| - 1$. To conclude that T^* is a tree, it remains to show that T^* contains no cycles, which we immediately do:

▶ **Theorem 3.** *T^* is a well-defined, rooted, ordered tree.*

We do this by explicitly specifying the parents of nodes in T^*, by making use of the depth-first traversal order $<$ in T. For this, let $T[v]$ be the *subtree* of T that hangs off (and includes) $v \in T$, i.e. $T[v]$ contains v and all its descendants in T. Let further

$$R[v] := \{u \in T \setminus T[v] \mid v < u\}$$

be all nodes "right of" v according to depth-first traversal order. For two nodes u, v where u is an ancestor of v, we immediately note that

$$T[v] \subset T[u], \quad R[u] \subset R[v] \quad \text{and} \quad R[v] \subset T[u] \cup R[u] \tag{10}$$

For a node $v \in T \setminus \{r\}$, we then obtain the following lemma:

▶ **Lemma 4.**

$$\mathsf{pa}^*(v) = \begin{cases} \min R[v] & R[v] \neq \emptyset \\ r & R[v] = \emptyset \end{cases}$$

We refer to the Appendix of the full version [4] for the proof of Lemma 4. Using Lemma 4, a proof of theorem 3 can be immediately given:

Proof of Theorem 3. Lemma 4 implies that $v <_T \mathsf{pa}^*(v)$ for all $v \in T \setminus \{r\}$. Therefore, T^* can contain no cycles and we obtain that T^* is a tree as a corollary. Furthermore, lemma 4 reveals that T^* is unique. ◀

See again the Appendix of [4] for immediate corollaries which point out how parents and subtrees in T^* relate with one another.

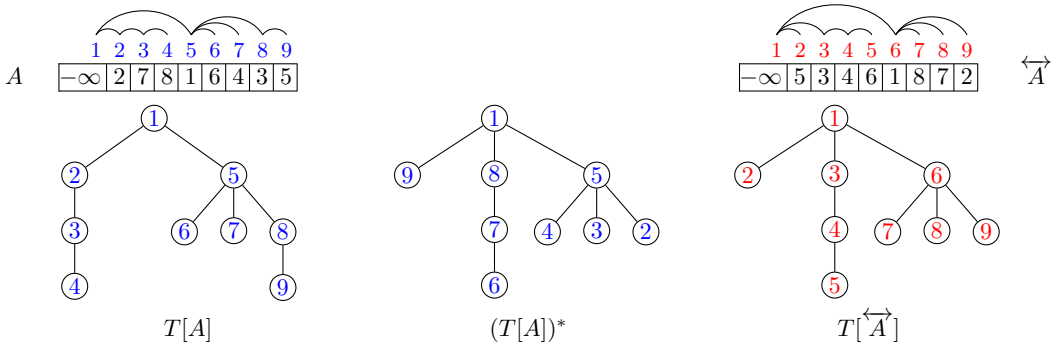

Figure 2 (left) An array A along with the 2D-Min-Heap $T[A]$. Arcs above array indices indicate tree paths. (middle) The dual tree $(T[A])^*$. (right) The reversed array $\overleftrightarrow{A}$ along with the 2D-Min-Heap $T[\overleftrightarrow{A}]$.

▶ Remark. An intuitive guideline for describing T^* in comparison to T is that parent- and siblinghood, as well as left and right are exchanged. In other words (and as will become clearer explicitly later) the duality describing T^* can be decomposed into two subdualities, one of which turns parents into siblings and vice versa, and the other one of which exchanges left and right.

This remark had left us with some choices for characterizing tree duality. Our choice is motivated by [9], arguably a cornerstone in RMQ theory development. To understand this, let $A = A[1, n]$ be the array, on which RMQ's are to be run, and let $\overleftrightarrow{A}$ be its reversal, given by $\overleftrightarrow{A}[i] = A[n - i + 1]$. Let $T[A]$ be the 2D-Min-Heap constructed from A, as described in [9] (a definition is provided in the Appendix of [4],

to which RMQ's refer (see (2),(3),(4)). An immediate question to ask is what RMQ's would look like when performing RMQ's on $\overleftrightarrow{A}$ instead of A. Here is the answer.

▶ **Theorem 5.** *Let $A[1, N]$ be an array and let $\overleftrightarrow{A} := [A[N], ..., A[1]]$ its reversal. Then*

$$(T[A])^* = T[\overleftrightarrow{A}] \tag{11}$$

An illustration of the Theorem is provided in Figure 2. See the Appendix of [4] for a more detailed treatment of this motivating example, including proofs. Thanks to theorem 5, the definition of T^* can arguably be considered a most natural choice, at least when relating tree duality with RMQ's.

Before proceeding with results on succinct tree representations, we provide the following intuitive lemma about the depth-first traversal order of T^* as a rooted, ordered tree. This lemma, in combination with lemma 4, supports the (intended) intuition that in T^* up and down, as well as left and right, are exchanged, properties that are characteristic for rooted, ordered tree duality. It also provides motivation beyond theorem 5 in the Introduction why T^* is the possibly canonical choice of the dual of a tree.

Therefore, let $<^*$ denote the depth-first traversal order in T^* (well-defined by theorem 3) while $<$ denotes the depth-first traversal order in (the primal tree) T.

▶ **Lemma 6.** *Let $u, v \in T \setminus \{r\}$. Then*

$$u <^* v \quad \textit{if and only if} \quad v < u$$

The proof of lemma 6 makes use of the following technical lemmata 7 and 8, which are of use also elsewhere. We therefore state these technical lemmata here. The proofs for all lemmata 6, 7 and 8 can finally be found in the Appendix of [4].

▶ **Lemma 7.** *Let* $w := \mathsf{pa}^*(v)$ *and* $v_2 \in T[v] \setminus v$ *such that* $\mathsf{pa}^*(v_2) = w$. *Then* $v_2 <^* v$.

▶ **Lemma 8.** *Let* v_1 *be a sibling left of* u_1 *in* T. *Then* $T[v_1] \subset T^*[u_1]$.

With lemma 6 proven, we can conclude with proving a main theorem of this treatment. It states that the dual of the dual is the primal tree, arguably a key property for a sensibly defined duality. Despite all lemmata raised so far, the proof still entails a few technically more demanding arguments.

▶ **Theorem 9.** $(T^*)^* = T$

Proof. It suffices to show that $\mathsf{pa}^{**}(v) = \mathsf{pa}(v)$, since lemma 6 establishes that the order in $(T^*)^*$ agrees with that of T. Let $u = \mathsf{pa}(v)$. In the Appendix of [4], we provide a (heavily technical) proof that

$$u = \begin{cases} \min_{<^*} R^*[v] & R^*[v] \neq \emptyset \\ r & R^*[v] = \emptyset \end{cases}$$

which completes the proof by applying lemma 4. ◀

2.1 Tree Reversal

We bring in another, simpler notion of tree duality, namely that of reversing trees. We will further elucidate what the trees are like when combining tree reversal with the tree duality (T^*) raised earlier.

▶ **Definition 10** (Reversed tree)**.** Let T be a tree. The reversed tree $\overleftrightarrow{T}$ of T is the tree resulting from reversing the order among the children of each node.

▶ **Proposition 11.** *Let* $\overleftrightarrow{T}$ *be the reversed tree of* T *and* $\overleftrightarrow{T^*}$ *be the reversed dual of* T. *We define irs (immediate right sibling) and lmc (left-most child) similarly as in Section 1.2.*

(a) *The root* r *of* T *is also the root of* $\overleftrightarrow{T}$.
(b) *Let* $u = \mathsf{pa}_T(v)$. *Then also* $u = \mathsf{pa}_{\overleftrightarrow{T}}(v)$.
(c) *Let* $u = ils_T(v)$. *Then* $u = irs_{\overleftrightarrow{T}}(v)$.
(d) *The root* r *of* T *is also the root of* $\overleftrightarrow{T^*}$.
(e) *If* $v = lmc_T(r)$ *then also* $v = lmc_{\overleftrightarrow{T^*}}(r)$, *implying in particular that* $\mathsf{pa}_{\overleftrightarrow{T^*}}(v) = r$.
(f) *If* $v = lmc_T(u)$ *with* $u \neq r$, *so* $v = ils_{\overleftrightarrow{T^*}}(u)$, *implying that* $\mathsf{pa}_{\overleftrightarrow{T^*}}(v) = \mathsf{pa}_{\overleftrightarrow{T^*}}(u)$.
(g) *If* $v = ils_T(u)$, *then* $v = lmc_{\overleftrightarrow{T^*}}(u)$, *implying that* $\mathsf{pa}_{\overleftrightarrow{T^*}}(v) = u$.
(h) $\overleftrightarrow{T^*} = \overleftrightarrow{T}^*$, *that is the reversed dual tree of* T *is the dual of the reversed tree of* T.

All of those are, in comparison with statements referring to the definition of the dual tree, rather obvious observations. See the Appendix of [4] for the proof.

Since $\overleftrightarrow{T}^*$ plays a particular role in the context of our introductory motivation, we give it a particular name: $\hat{T}$.

▶ **Definition 12** (Reversed dual tree)**.** Let T be a tree. The tree $\hat{T} := \overleftrightarrow{T}^*$ of T is the dual of the reversed (or the reversed dual) tree of T.

Based on proposition 11, we realize that $\hat{T}$ can be described as turning leftmost children into immediate left siblings.

▶ Remark. Following the arguments provided in [8], it becomes evident that the tree T in use there, on which $\mathrm{BP}(T)$ is constructed, turns indeed out to be $\widehat{T[A]} = \overleftrightarrow{T[A]}^*$.

3 The Primal-Dual Ancestor

The following theorem points out that pairs of nodes have a unique *primal-dual ancestor*. We will further point out properties of that node.

▶ **Theorem 13.** *Let $v_1, v_2 \in T \setminus \{r\}$ be two nodes where $v_1 \leq v_2$. Then there is a unique node $v \in T \setminus \{r\}$ such that $v_1 \in T^*[v]$ and $v_2 \in T[v]$.*

We henceforth refer to this unique node as *primal-dual ancestor* of v_1 and v_2, written $\mathsf{pda}(v_1, v_2)$.

Proof. Let

$$v := \max_{<_T}\{v_1 \leq x \leq v_2 \mid v_1 \in T^*[x]\} \tag{12}$$

be, relative to depth-first traversal order in T, the largest ancestor of v_1 in T^* that precedes v_2. We claim that v is the unique primal-dual ancestor of v_1 and v_2.

By definition, we immediately obtain that $v_1 \in T^*[v]$. To prove $v_2 \in T[v]$, consider $\mathsf{pa}^*(v)$, for which, by choice of v, we have that $v_2 < \mathsf{pa}^*(v)$. By lemma 4, however, $\mathsf{pa}^*(v)$ is the first node in $R[v]$, relative to depth-first traversal order in T. Hence, for any y such that $v \leq y < \mathsf{pa}^*(v)$, which includes v_2, it holds that $y \in T[v]$.

It remains to show that v is the only possible primal-dual ancestor. By definition of the primal-dual ancestor, v must be an ancestor of v_1 in T^*.

First, consider an ancestor y of v_1 in T^* such that $y < v$. By choice of v, it holds that $\mathsf{pa}^*(y) \leq v_2$, while $\mathsf{pa}^*(y) \in R[y]$. This implies that also $v_2 \in R[y]$, and not $v_2 \in T[y]$, hence y cannot be a primal-dual ancestor of v_1 and v_2.

Second, consider an ancestor y of v_1 in T^* such that $v < y$. Because v is an ancestor of v_1 in T^*, and y is larger than v, y is also an ancestor of v in T^*. By lemma 4, we know that $y \in R[v]$. This, in combination with $v_2 \in T[v]$ implies that $v_2 < y$, hence, y cannot be an ancestor of v_2 in T. ◀

For the following theorem, let

$$\mathrm{depth}_T(v_1, v_2) := \min\{\mathrm{depth}_T(y) \mid v_1 \leq y \leq v_2\}$$

be the minimal depth of nodes between (and including) v_1 and v_2.

▶ **Theorem 14.** *Let $v_1, v_2 \in T \setminus \{r\}$ such that $v_1 < v_2$. It holds that*

$$\mathsf{pda}(v_1, v_2) = \max_{<}\{v_1 \leq x \leq v_2 \mid \mathrm{depth}_T(x) = \mathrm{depth}_T(v_1, v_2)\} \tag{13}$$

That is, according to depth-first traversal order in T, the primal-dual ancestor is the greatest node whose T-depth is minimal among all nodes between (and including) v_1 and v_2.

The proof is based on the following lemma:

▶ **Lemma 15.** *Let $v < w$ such that $w \in T[\mathsf{pa}^*(v)]$. Then it holds that*

$$\mathrm{depth}_T(v, w) = \mathrm{depth}_T(\mathsf{pa}^*(v)) \tag{14}$$

See the Appendix of [4] for a proof of lemma 15 and then theorem 14.

Note immediately that theorem 14 implies that v can be found in O(1) runtime, by performing a range minimum query on the excess array D of BP(T), defined by $D[x] := \mathsf{rank}_1(x) - \mathsf{rank}_0(x)$ where rank refers to BP(T). Since $D[x+1] - D[x] \in \{-1, +1\}$, an RMQ on D means performing a ± 1-RMQ, for which convenient solutions exist [1].

Re-interpretation of RMQ's. Because it was shown [9], that the node in the 2D-Min-Heap $T[A]$ that corresponds to the solution of $\mathsf{rmq}_A(i,j)$ is given by the right hand side of (13), theorems 13 and 14 allow for a reinterpretation of an RMQ query $\mathsf{rmq}_A(i,j)$ on an array A (without going into details here, because the proof is an easy exercise based on collecting facts from here, [9] and [8]).

1. Determine the node v in $T[A]$ corresponding to i.
2. Determine the node w in $T[A]$ corresponding to j.
3. Determine $\mathsf{pda}(v,w)$ in $T[A]$; return the corresponding index i_o.

Re-interpretation and improvement of Minimal Length Interval Queries (MLIQ). To illustrate the potential practical benefits of our treatment, we further revisit the problem of *minimal length interval queries (MLIQ)*. The improvements we will be outlining are similar in spirit to the ones delivered in [8]. However, based on our results, they are considerably more convenient to obtain.

▶ **Problem 16** (MLIQ). *Let* $([a_i, b_i])_{i \in \{1,\ldots,n\}}, a_i, b_i \in \mathbb{N}$ *such that* $a_i \leq b_i$ *for all* $i \in \{1, \ldots, n\}$ *and* $a_i < a_j$ *and* $b_i < b_j$ *for* $i < j$.

- ***Input****:* (a, b) *such that* $a < b$
- ***Output****: The index* i_0 *such that* $[a_{i_0}, b_{i_0}]$ *is the shortest interval that contains* $[a, b]$*, if such an interval exists.*

This problem makes part of other relevant problems, for example the *shortest unique interval* problem. In this context, a solution for the MLIQ problem was presented in [12] that requires $O(b_n \log b_n)$ space to answer the query in $O(1)$ time. Therefore, the following strategy was suggested.

Let $l_i := |b_i - a_i + 1|$ be the length of the i-th interval, $A := [l_1, \ldots, l_n]$ and $T[A]$ the corresponding 2D-Min-Heap.

1. $i_{min} := \min\{i \mid b_i > b\}, i_{max} := \max\{i \mid a_i < a\}$; if $i_{max} < i_{min}$ **output** 'None'.
2. Determine nodes $v, w \in T[A]$ corresponding to i_{min}, i_{max}.
3. Determine $\mathsf{pda}(v,w) \in T[A]$; **output** its index.

The solution presented in [12] can immediately be improved by employing bitmaps for the first step (which, according to [18], requires $O(n \log(b_n/n)) + o(b_n)$ space). Steps 2 and 3 then reflect an ordinary RMQ, which can be dealt with following [8]. In terms of query counts, Step 1 reflects two rank queries, while the resulting RMQ, following [8], requires two select's, one ± 1-rmq, and one rank.

If $|a_i - a_{i-1}|, |b_i - b_{i-1}|$ are in $O(\log n)$ (which applies for several important applications), further improvements can be made based on suggestions made in [20] for BP representations of trees with weighted parentheses. For that, we construct $T_a = T[A]$ and $T_b = \overleftrightarrow{T[A]}$. We then assign weights $w_{a,i} := |a_i - a_{i-1}|$ to $i+1$-st opening parenthesis in T_a, whereas in T_b we assign $w_{b,i} := |b_i - b_{i-1}|$ to the i-th closing parentheses (where $a_0 = b_0 = 0$; we recall that the number of non-root nodes in $T[A]$ is n). When aiming at running queries presented in [20],

this requires $2n \log\log n + o(n)$ bits of space, an improvement over $O(n \log(b_n/n)) + o(b_n)$ for the above, naive approach. Following [20], let $\mathsf{bpselect}_{w_a,0}(a), \mathsf{bpselect}_{0,w_b}(b)$ be defined by selecting the largest index in the balanced parenthesis vector such that adding up all weights attached to opening parentheses (w_a) is at most a, or adding up all weights attached to closing parentheses (w_b) is at most b. We can then run

1. $w := \mathsf{bpselect}_{w_a,0}(a)$ in T_a and $v := 2n - \mathsf{bpselect}_{0,w_b}(b) + 3$ in T_b; if $v > w$ **output** 'None'
2. Determine $\mathsf{pda}(v, w) \in T_a$; **output** its index.

In comparison to the naive approach from above, this makes two bpselect queries, instead of two rank's and two select's. The *decisive trick* is to place a and b directly into $T[A]$, which avoids determining indices i_{min}, i_{max} first, which subsequently need to be placed. Beyond the improvements in terms of space and query counts, we argue that this solution reflects all symmetries inherent to the MLIQ problem in a particularly compact manner.

4 Relating BP and DFUDS representations

We will use the following construction to set up a tree induction for proving our main theorem.

▶ **Definition 17** (Tree joining operation)**.** Let T_1 and T_2 be two trees, let r_2 be the root of T_2, $rmc_{T_2}(r_2)$ needs to exist and be a leaf. The notation $T_1 \curvearrowright T_2$ will denote a new tree formed by taking T_2 and inserting the children of the root of T_1 as children of the rightmost child of the root of the new tree. Extend this operation to n trees $T_1, \ldots, T_n$ where $T_2, \ldots, T_n$ all satisfy the same property as T_2 above, in the following way: $T_1 \curvearrowright T_2 \curvearrowright T_3 = (T_1 \curvearrowright T_2) \curvearrowright T_3$ and so on,

$$T_1 \curvearrowright T_2 \ldots \curvearrowright T_n = ((\ldots((T_1 \curvearrowright T_2) \curvearrowright T_3) \curvearrowright \ldots) \curvearrowright T_n).$$

▶ **Observation 18.** *Let T be a tree such that its root r has a single child c (that may or may not be a leaf). Then in T^*, by Rule 1b, $rmc_{T^*}(r) = c$ and is a leaf.*

The following Lemma (proven in the Appendix of [4])

relates the dual tree to the tree joining operation. We will use the $r \to T$ notation to denote a new tree formed by adding a new root r as a parent of the root of T.

▶ **Lemma 19.** *Let T be a tree consisting of a root r and $n \geq 1$ subtrees $A_1, A_2, \ldots, A_n$ as children. When $n = 1$, T^* is $(r \to A_1)^*$. When $n \geq 2$, T^* is $(r \to A_1)^* \curvearrowright (r \to A_2)^* \curvearrowright \ldots \curvearrowright (r \to A_n)^*$.*

We are now ready to prove Theorem 1. Parentheses in BP and DFUDS representations will be denoted by $\underline{(}$ and $\underline{)}$ to avoid confusion with usual mathematical parentheses. Recall that we use $\overleftrightarrow{s}$ to mirror a string s of parentheses, e.g. $\overleftrightarrow{\underline{(()}} = \underline{())}$ and $\overleftrightarrow{\underline{)()}} = \underline{()(}$.

Proof of Theorem 1. Let T be a tree with n subtrees $A_1, \ldots, A_n$. It is clear that $\mathrm{BP}(T) = \underline{(}\mathrm{BP}(A_1)\mathrm{BP}(A_2)\ldots\mathrm{BP}(A_n)\underline{)}$. Observe that for two trees T_1 and T_2 with roots v_1 and v_2, and where $rmc_{T_1}(v_1), rmc_{T_2}(v_2)$ both exist and are leaves,

$$\mathrm{DFUDS}(T_1 \curvearrowright T_2) = \underline{(}\mathrm{DFUDS}(T_2 \setminus rmc_{T_2}(v_2))\mathrm{DFUDS}(T_1 \setminus rmc_{T_1}(v_1))\underline{)}.$$

In fact, one can show recursively that such a decomposition can be extended to $T_1 \curvearrowright \ldots \curvearrowright T_n$. We will now prove the theorem with a tree structural induction. Observe that for a tree T of depth 1 (a single root node),

$$\mathrm{BP}(T) = \underline{()} = \mathrm{DFUDS}(T^*) = \overleftrightarrow{\mathrm{DFUDS}(T^*)}.$$

Now, assume the theorem equality is true for trees of depth i and we will show it for trees of depth $i+1$. A tree T of depth $i+1$ can be decomposed into a root node r and n subtrees $A_1, \ldots, A_n$ that are all of of depth $\leq i$ with roots $a_1, \ldots, a_n$. Using Lemma 19,

$$\mathrm{DFUDS}(T^*) = \mathrm{DFUDS}((r \to A_1)^* \curvearrowright (r \to A_2)^* \curvearrowright \ldots \curvearrowright (r \to A_n)^*).$$

By the recursive decomposition that we observed above, and using Observation 18 stating that the rightmost child of r in $(r \to A_i)^*$ is a leaf,

$$\mathrm{DFUDS}(T^*) = \underline{(}\mathrm{DFUDS}((r \to A_n)^* \setminus \{a_n\}) \ldots \mathrm{DFUDS}((r \to A_1)^* \setminus \{a_1\}))\underline{)}.$$

Observe that we can take each DFUDS term in the expression above and wrap it around parentheses, i.e. $\underline{(}\mathrm{DFUDS}((r \to A_i)^* \setminus \{a_i\})\underline{)}$ which is equal to $\mathrm{DFUDS}((r \to A_i)^*)$. Furthermore, note the following identity: $\mathrm{DFUDS}((r \to A_i)^*) = \underline{(}\mathrm{DFUDS}(A_i^*)\underline{)}$. And by inductive hypothesis, $\mathrm{DFUDS}(A_i^*) = \overleftrightarrow{\mathrm{BP}(A_i)}$, thus $\mathrm{DFUDS}((r \to A_i)^* \setminus \{a_i\}) = \overleftarrow{\mathrm{BP}(A_i)}$. Hence,

$$\overleftrightarrow{\mathrm{DFUDS}(T^*)} = \underline{(}\mathrm{BP}(A_1) \ldots \mathrm{BP}(A_n)\underline{)} = \mathrm{BP}(T). \qquad \blacktriangleleft$$

Proving (7) from the Introduction. Eventually, we also realize that $\mathrm{BP}(\overleftrightarrow{T}) = \overleftrightarrow{\mathrm{BP}(T)}$ and also $\mathrm{DFUDS}(\overleftrightarrow{T}) = \overleftrightarrow{\mathrm{DFUDS}(T)}$, both of which is straightforward [$\star$]. Using this in combination with theorems 9 and 1, we obtain

$$\mathrm{DFUDS}(T[A]) \overset{[\star]}{=} \overleftrightarrow{\mathrm{DFUDS}(\overleftrightarrow{T[A]})} \overset{Th.9}{=} \overleftrightarrow{\mathrm{DFUDS}((\overleftrightarrow{T[A]}^*)^*)} \overset{Th.1}{=} \mathrm{BP}(\overleftrightarrow{T[A]}^*) \overset{D.12}{=} \mathrm{BP}(\widehat{T[A]})$$

which establishes equation (7) from the introduction.

Conclusive Remarks. In summary, we have provided a framework that unifies BP and DFUDS. From a certain point of view, we have pointed out that neither should BP based approaches have advantages over DFUDS based approaches, nor vice versa. As an exemplary perspective of our framework, BP based treatments such as [17, 20] might have an easier grasp of the advantages that DFUDS based approaches bring along. Finally, we consider it interesting future work to also characterize trees that put BP and/or DFUDS based representations into context with LOUDS based representations.

References

1. M. Bender and G. Farach-Colton. The LCA problem revisited. *Proc. 4th LATIN*, LNCS 1776:88–94, 2000.
2. D. Benoit, E.D. Demaine, J.I. Munro, R. Raman, V. Raman, and S.S. Rao. Representing trees of higher degree. *Algorithmica*, 43(4):275–292, 2005.
3. O. Berkman and U. Vishkin. Recursive star-tree parallel data structure. *SIAM Journal of Computing*, 22(2):221–242, 1993.
4. R. Chikhi and A. Schönhuth. Dualities in Tree Representations. *ArXiv e-prints*, 2018. `arXiv:1804.04263`.
5. Pooya Davoodi, Rajeev Raman, and Srinivasa Rao Satti. On succinct representations of binary trees. *Mathematics in Computer Science*, 11(2):177–189, 2017. `doi:10.1007/s11786-017-0294-4`.
6. O. Delpratt, N. Rahman, and R. Raman. Engineering the LOUDS succinct tree representation. In *Proc. of the WEA*, LNCS 4007, pages 134–145, 2006.

7 Arash Farzan, Rajeev Raman, and S. Srinivasa Rao. Universal succinct representations of trees? In Susanne Albers, Alberto Marchetti-Spaccamela, Yossi Matias, Sotiris E. Nikoletseas, and Wolfgang Thomas, editors, *Automata, Languages and Programming, 36th International Colloquium, ICALP 2009, Rhodes, Greece, July 5-12, 2009, Proceedings, Part I*, volume 5555 of *Lecture Notes in Computer Science*, pages 451–462. Springer, 2009. doi:10.1007/978-3-642-02927-1_38.

8 H. Ferrada and G. Navarro. Improved range minimum queries. *Journal of Discrete Algorithms*, 43:72–80, 2017.

9 J. Fischer and V. Heun. Space-efficient preprocessing schemes for range minimum queries on static arrays. *SIAM Journal on Computing*, 40(2):465–492, 2011.

10 H.N. Gabow, J.L. Bentley, and R.E. Tarjan. Scaling and related techniques for geometry problems. In *Proc. 16th STOC*, pages 135–143, 1984.

11 R. Grossi and G. Ottaviano. Design of practical succinct data structures for large data collections. In *Proceedings of the 12th SEA*, volume LNCS 7933, pages 5–17, 2013.

12 X. Hu, J. Pei, and Y. Tao. Shortest unique queries on strings. In *Proceedings of the International Symposium on String Processing and Information Retrieval (SPIRE)*, pages 161–172, 2014.

13 G. Jacobson. Space-efficient static trees and graphs. In *Proc. of the FOCS*, pages 549–554, 1989.

14 J. Jansson, K. Sadakane, and W.-K. Sung. Ultra-succinct representation of ordered trees. In *Proc. of the SODA*, pages 575–584, 2007.

15 J.I. Munro and V. Raman. Succinct representation of balanced parentheses and static trees. *SIAM Journal of Computing*, 31(3):762–776, 2001.

16 G. Navarro, Y. Nekrich, and L.M.S. Russo. Space-efficient data analysis queries on grids. *Theoretical Computer Science*, 482:60–72, 2013.

17 G. Navarro and K. Sadakane. Fully-functional static and dynamic succinct trees. *ACM Transactions on Algorithms*, 10(3), 2014. Article 16.

18 R. Raman, V. Raman, and S.R. Sattie. Succinct indexable dictionaries with applications to encoding k-ary trees, prefix sums and multisets. *ACM Transactions on Algorithms*, 3(4), 2007.

19 Kunihiko Sadakane. Compressed suffix trees with full functionality. *Theory of Computing Systems*, 41(4):589–607, 2007.

20 D. Tsur. Succinct representation of labeled trees. Technical Report 1312.6039, ArXiV, 2015.

21 J. Vuillemin. A unifying look at data structures. *Communications of the ACM*, 4:229–239, 1980.