Acta Cybernetica **25** (2022) 629–646.

An Efficient Sampling Algorithm for Difficult Tree Pairs[*]

Sean Cleary[a] and Roland Maio[b]

Abstract

It is an open question whether there exists a polynomial-time algorithm for computing the rotation distances between pairs of extended ordered binary trees. The problem of computing the rotation distance between an arbitrary pair of trees, (S, T), can be efficiently reduced to the problem of computing the rotation distance between a difficult pair of trees (S', T'), where there is no known first step which is guaranteed to be the beginning of a minimal length path. Of interest, therefore, is how to sample such difficult pairs of trees of a fixed size. We show that it is possible to do so efficiently, and present such an algorithm that runs in time $O(n^4)$.

Keywords: rotation distances, associahedra, rooted binary trees, sampling

1 Introduction

Trees are a fundamental data structure with wide applications ranging from efficient search (such as binary search trees) to modeling biological processes (such as phylogenetic trees). Given pairs of trees, there are numerous ways of calculating metrics of interest between trees. These metrics measure of the amount of commonality and difference, which depend on the class of trees considered and what features of the trees are regarded as important to have in common.

Trees arise in data storage and searching as efficient structures. When there is a natural order on leaves, we have binary search trees which underlie many storage and searching systems. See Knuth [10] for background and important foundational notions and algorithms. Binary search trees and their generalizations underlie almost all modern file-storage and data-storage systems. To ensure good average-time search performance of $\log(n)$, it is necessary to have reasonably balanced trees and rotations are a quick, local change which can be used to keep trees close to

[*]Partial funding provided by NSF #1417820. This work was partially supported by a grant from the Simons Foundation (#234548 to Sean Cleary).

[a]Department of Mathematics, The City College of New York and the CUNY Graduate Center, USA, E-mail: `scleary@ccny.cuny.edu`, ORCID: `0000-0002-3123-8658`

[b]Department of Computer Science, Columbia University, New York, NY, USA, E-mail: `rolandmaio38@gmail.com`, ORCID: `0000-0002-7317-770X`

DOI: `10.14232/actacyb.285522`

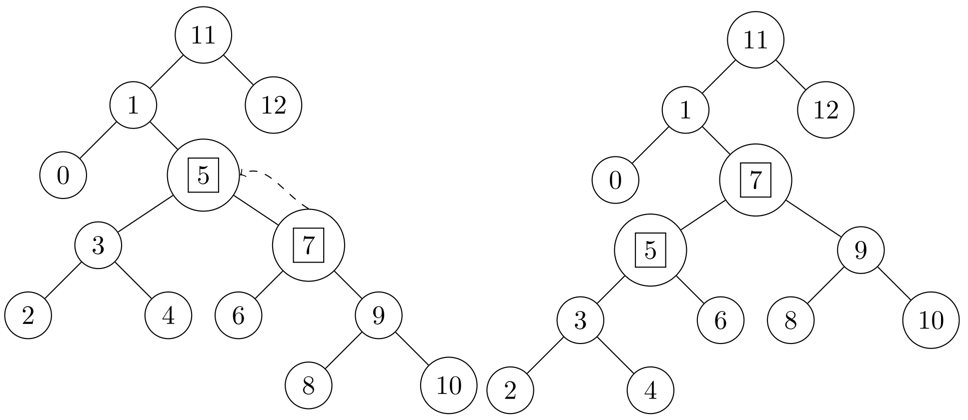

Figure 1: An example of a left rotation in an ordered tree. We rotate leftward at node 5, with the rotation promoting node 7 to the position of its former parent. Node 5 is demoted to become a left child of node 7 and node 7 is promoted to be the right child of node 1. Node 6 changes from being the left child of 7 to the right child of 5.

balanced during sequences of insertions and deletions. Furthermore, rooted binary trees are in direct correspondence with triangulations of polygons with a marked edge, and as described by Hanke, Ottmann, and Schuierer [9], such tree metrics apply to edge-flipping conversions between triangulations of planar regions.

One widely-considered tree distance metric on trees with a natural left-to-right order on leaves is that of the rotation distance between a pair of extended ordered binary trees. A rotation at a node is a simple local transformation which does not affect the order, with an example shown in Figure 1. Rotation distance between trees is the length of the shortest possible sequence of rotations to accomplish the transformation between trees. There are no known polynomial-time algorithms for computing rotation distance. Culik and Wood [7] originally described rotation distance, and ground-breaking work of Sleator, Tarjan and Thurston [13] used the correspondence between trees with n internal nodes and triangulations of the marked regular $n+2$-gon to show that if there is a common edge between the two triangulations then any shortest path does not flip this edge. Such a common edge thus breaks the rotation distance problem into two smaller sub-problems. Furthermore, they showed that if it is possible to flip an edge of either polygon to obtain a common edge, then there is a shortest path which begins by doing so. We call edges which are not common but which can be flipped to become a common edge *one-off* edges, as they are one move away from being common edges themselves. Cleary and St. John [6] used these reduction rules to show that the rotation distance problem is fixed parameter tractable.

We call a pair of trees with no common edges and no edges which can be immediately flipped to create a common edge a *difficult tree pair.* The above reductions transform the problem of computing the rotation distance on a pair of trees drawn from all possible pairs to a pair of trees drawn from the set of all such difficult tree pairs. A common edge, arising either immediately or from performing a single flip to change a one-off edge to a common edge, then naturally splits the tree pair into a pair of smaller tree pairs, as explained in Sleator, Tarjan, and Thurston [13]. The kernel of the difficult of the rotation distance problem at this point is to find distances between difficult tree pairs.

To understand how effective different approximation and partial algorithms are at evaluating and estimating rotation distance, it would be useful to sample difficult tree pairs. It is possible to find examples of difficult tree pairs by picking a tree pair of large size at random, and then performing all possible reductions and one-off moves, splitting the problem into a collection of smaller subproblems, until either the trees are identical (extremely unlikely) or until a collection of difficult tree pairs is obtained. But such a procedure is not only time-consuming, it is not possible to tell in advance how many reductions there will be and what the resulting sizes of the smaller remaining difficult piece pairs will be. Thus there is no control on the resulting size of the difficult tree pairs produced. In general (see Cleary, Rechnitzer and Wong [2]) there are a sizable number of common edges and one-off edges, resulting on average about at least a 10% reduction in the size of a randomly selected tree pair to a largest difficult remaining tree pair. It is not difficult to construct specific examples of specified size of difficult tree pairs- examples of Dehornoy [8], Pournin [11], and Cleary and Maio [3] are families of difficult pairs but in each case of a restricted type. In many of these very specific cases, analysis to that family of instances can give coincident upper and lower bounds on rotation distance, giving an exact calculation. But these families are very sparse in the set of all difficult tree pairs. The set of all difficult tree pairs appears to grow exponentially with size, but at a slower expontial growth rate than the set of all tree pairs, per work of Cleary and Maio [4] suggesting that the fraction of all tree pairs decreases exponentially at a rate of about 0.77^n, with already ratio of less than 1 in a billion tree pairs of size 70 being difficult and the fraction dropping with further increases of size.

Difficult tree pairs lie at the kernel of a number of questions of interest. Because the rotation distance problem frequently splits into smaller subproblems, the essential difficulties are contained in the set of difficult tree pairs. Difficult tree pairs can be used to test estimation algorithms for rotation distance, to find estimates for typical rotation distance between tree pairs selected at random, and to look for difficult pathological behavior for rotation distance paths.

This motivates studying difficult tree pairs in their own right. We describe below an efficient algorithm for sampling difficult tree pairs of a specified size. This sampling is not uniform across all difficult tree pairs of a prescribed size but does have wide coverage of such pairs.

The algorithm we describe can be seen as a variation on Remy's algorithm [12] for efficiently generating rooted ordered trees uniformly at random, but instead of working on growing the size of a single tree, we grow a pair of trees while applying a filtering criterion. Unlike Remy's algorithm, the difficult pairs are not sampled uniformly at random but having an efficient (polynomial-time) means of generating pairs is useful for understanding rotation distance problem instances better and for testing the performance of new algorithms. Computational experiments show that the distribution of selected tree pairs are not uniformly random but there does seem to be wide dispersion, with relatively broad coverage of difficult tree pairs.

2 Background

An *extended ordered binary tree* is a rooted binary tree where every node has exactly 0 or 2 children and whose leaves are labelled starting with 0 in their order defined by a pre-order traversal from the root. The label of a leaf node ℓ is denoted $\text{label}(\ell)$. The size of an extended ordered binary tree T, denoted $|T|$, is the number of internal nodes T contains. The set of all nodes in T is denoted $\text{nodes}(T)$. In the following *tree* will refer to an extended ordered binary tree and S and T will be trees of the same size.

A *rotation at a node* ν in a tree is a local operation which promotes an internal node ν to the position of its parent μ, demotes μ to one of ν's children, and makes one of ν's children a child of μ, illustrated in Figure 1 where the rotation is leftward at node 7 going from the left tree to the right tree. The inverse operation of going from the right tree to the left tree is a rightward rotation at node 5. We will denote the partial function that returns the parent of a node by $\pi : \text{nodes}(T) \mapsto \text{nodes}(T)$, with $\pi(\text{root}(T))$ is undefined. We adopt the convention that the statement "rotate at a node ν", denoted $\text{rotate}(\nu)$, means to perform that rotation which promotes ν to the position of its parent. Whether that is a right or left rotation depends upon whether ν is a left or right child of its parent. For a tree of size n there are $n-1$ possible rotations, one for each internal node except the root.

Given a pair of trees (S, T) of the same size, it is possible to transform one into the other by some sequences of rotations. The minimum length of any such sequence defines the *rotation distance* between S and T, which we denote $d(S, T)$.

The interval of a node ν, $\text{interval}(\nu)$, is the pair (α, β) where α is the label of the least-labelled leaf in the tree rooted at ν and β is the label of the greatest-labelled leaf in the tree rooted at ν. The label α is called the *lower bound of the interval of* ν and is denoted $\lfloor \text{interval}(\nu) \rfloor$. Similarly, the label β is called the *upper bound of the interval of* ν and is denoted $\lceil \text{interval}(\nu) \rceil$. If ν is a leaf, then its lower bound is the same as its upper bound and is defined to be its label. If ν is an internal node, then its lower bound is the lower bound of its left child, and its upper bound is the upper bound of its right child; formally

$$\text{interval}(\nu) = \begin{cases} (\text{label}(\nu), \text{label}(\nu)) & \text{if } \nu \text{ is a leaf} \\ (\lfloor \text{interval}(\text{left}(\nu)) \rfloor, \lceil \text{interval}(\text{right}(\nu)) \rceil) & \text{otherwise} \end{cases}$$

The intervals of a tree T, denoted $\text{intervals}(T)$, is the set of all intervals of the internal nodes of T.

The labels α and β are related to each other by the size of the subtree rooted at ν in the following way:

Proposition 1. *Let ν be an internal node of T, and N the subtree rooted at ν, and $(\alpha, \beta) = \text{interval}(\nu)$, then $\beta = \alpha + |N|$.*

Proof. Recall that N has $|N| + 1$ leaves. It is a property of pre-order traversal that once the traversal visits a node it will visit the entire subtree rooted at that node before it visits any other part of the tree. Consequently, when the pre-order traversal reaches ν, the next $|N| + 1$ leaf nodes that will be visited will be the leaf nodes of N. Thus, the greatest label any leaf in N can have is $\alpha + |N|$ and this must be attained by the last leaf that is visited in N. $\square$

In addition to changing one tree into another, a rotation in a tree T at a node ν has the effect of replacing one of the intervals of the tree by a new one. This new interval is uniquely determined by T and ν and is denoted 1-interval(ν). The 1-interval(ν) can be defined in terms of the intervals of ν, the parent of ν, and the children of ν. If ν is the left child of its parent, then the lower bound of 1-interval(ν) is the lower bound of the right child of ν and the upper bound of 1-interval(ν) is the upper bound of the parent of ν. If ν is the right child of its parent, then the lower bound of 1-interval(ν) is the lower bound of its parent, and the upper bound of 1-interval(ν) is the upper bound of the left child of ν. Formally

$$\text{1-interval}(\nu) = \begin{cases} (\lfloor \text{interval}(t) \rfloor, \lceil \text{interval}(\pi(\nu)) \rceil) & \text{if } \nu = \text{left}(\pi(\nu)) \\ (\lfloor \text{interval}(\pi(\nu)) \rfloor, \lceil \text{interval}(s) \rceil) & \text{otherwise} \end{cases}$$

where $s = \text{left}(\nu)$ and $t = \text{right}(\nu)$.

The 1-intervals(T) is the set of all $n - 1$ intervals that can be obtained by rotating some node in T.

Trees correspond naturally to the marked triangulations of a polygon, and we denote the corresponding triangulation by $\triangle(T)$. The edges of $\triangle(T)$ correspond to the intervals(T).

While the reduction rules were first developed from the perspective of triangulations of the polygon, they may be formulated from the tree perspective in terms of intervals and rotations. A common edge between triangulations corresponds to a common interval occuring in the intervals of both trees. A one-off edge between triangulations corresponds to a common interval that can be obtained by rotating at one of the nodes in S or T.

The binary word of T, word(T), is obtained by beginning with the empty string, traversing T in pre-order and appending at each node a '1' if the node is an internal node and a '0' otherwise. Thus the symbol at the ith index in word(T) is determined by the ith node visited in T by a pre-order traversal. This determines a mapping from symbols in word(T) to nodes(T).

Definition 1. *Let T be an extended ordered binary tree, and let ν be the i-th node visited in a pre-order traversal of T. The symbol of ν in* word(T)*, denoted* $\text{sym}_\text{T}(\nu)$*, is defined to be the the symbol of* word(T) *at index i.*

The following property of word(T) gives one method for computing the intervals of T.

Proposition 2. *Let ℓ be a leaf node of T, then the label of ℓ is given by the number of '0's that precede* $\mathrm{sym}_{\mathrm{T}}(l)$.

Proof. Suppose the label of ℓ is α. By the definition of label, ℓ is the $(\alpha + 1)$st leaf node visited in the preorder traversal of T. In computing word(T), therefore, exactly α '0's must have been appended before $\mathrm{sym}_{\mathrm{T}}(l)$ is appended. □

Proposition 3. *Let T be an extended ordered binary tree, ν an internal node of T, N the subtree of T rooted at ν, and $(\alpha, \beta) = \mathrm{interval}(\nu)$. Then α is given by the number of 0's that precede $\mathrm{sym}_{\mathrm{T}}(\nu)$, and $\beta = \alpha + |N|$.*

Proof. To prove α is given by the number of 0's that precede $\mathrm{sym}_{\mathrm{T}}(\nu)$ it suffices, by Proposition 2, to show that the symbol of the leaf node, ℓ, with label α is the first 0 that proceeds $\mathrm{sym}_{\mathrm{T}}(\nu)$. Suppose this is not the case, then there is at least one 0 that proceeds $\mathrm{sym}_{\mathrm{T}}(\nu)$ and precedes $\mathrm{sym}_{\mathrm{T}}(l)$. Then there is some leaf node k in the subtree rooted at ν that is visited after ν and before ℓ. So the label of k is at most $\alpha - 1$, but this contradicts the assumption that ℓ is the least labelled leaf. Finally, from Proposition 1 it follows that $\beta = \alpha + |N|$. □

We let $\circ$ denote string concatenation and we let ν be a node of a tree. We define the functions left(ν) and right(ν) to return the left or right child or ν respectively. A recursive definition for word(ν) can then be given as follows

$$\mathrm{word}(\nu) = \begin{cases} 1 \circ \mathrm{word}(\mathrm{left}(\nu)) \circ \mathrm{word}(\mathrm{right}(\nu)) & \text{if } \nu \text{ is an internal node} \\ 0 & \text{otherwise} \end{cases}$$

With this definition word(T) = word(r) where r is the root of T.

Remy's algorithm [12] is a method for sampling trees of a fixed size uniformly at random by growing a tree larger at each stage ensuring that each possible tree of that size is equally likely to be generated. The algorithm begins with a tree of size 1 and iteratively grows the tree until a tree of the desired size is obtained. On each iteration, one of the internal or external nodes, say ν, of the current tree, say T, is selected uniformly at random. Then a new node, μ, is created. The new node μ takes the place of ν in the tree, and ν is set as the left or right child of μ with equal probability. We say that the resulting tree is obtained from T by growing left (or right) at ν.

If a tree S may be grown in some way by an iteration of Remy's algorithm to obtain a tree T, then we call T a growth neighbor of S and denote the set of all growth neighbors of S by growthNeighbors(S).

On an iteration of Remy's algorithm, if an external node is chosen to be grown, then growing left or right will result in the same tree. Thus, an upper bound on the number of growth neighbors a tree of size n may have is $3n + 1$.

3 Difficult Pair Sampling Algorithm

The Difficult Pair Sampling algorithm, DPS, begins by randomly choosing one of the 4 difficult pairs of trees of size 4. We call these difficult pairs *primitive* because there are no difficult pairs of trees that are smaller. The algorithm then iteratively grows the pair of trees in size by 1 until a pair of the desired size is obtained. On each iteration, for the current pair of trees S and T, DPS finds all difficult pairs of trees (U, V) such that U is a growth neighbor of S and V is a growth neighbor of V and randomly selects one of these pairs to be the next S and T.

```
DPS($n$)
1  $S$, $T$ = randomPrimitiveDifficultPair()
2  for $i$ = 5 to $n$
3       $choices = \emptyset$
4       for $U$ in growthNeighbors($S$)
5            for $V$ in growthNeighbors($T$)
6                 if isDifficultPair($U, V$)
7                      $choices.add((U, V))$
8       $S$, $T$ = $choices.randomElement()$
9  return $S$, $T$
```

What is not obvious about DPS is that for an arbitrary difficult pair (S, T), it is always possible to grow S and T into a difficult pair (U, V). We will show that this is the case by examining a particular growth neighbor. Generally there are many additional growth neighbors but the existence of a single one suffices for proving the correctness of the algorithm.

Definition 2. *Let T be an extended ordered binary tree of size n, and ω be the internal node of T whose right child is the leaf with label n. The extended ordered binary tree of size $n+1$, obtained by growing T at ω left, will be denoted $\sigma(T)$.*

We will show that given a difficult pair (S, T), the pair of trees $(\sigma(S), \sigma(T))$ is also a difficult pair. The proof that $(\sigma(S), \sigma(T))$ is a hard pair will rest on the relation between intervals(T) to intervals($\sigma(T)$) and 1-intervals(T) to 1-intervals($\sigma(T)$).

Relating intervals(T) to intervals($\sigma(T)$) and 1-intervals(T) to 1-intervals($\sigma(T)$) will require relating word(T) to word($\sigma(T)$) which we will do next.

Lemma 1. *Let T be an extended ordered binary tree of size n, then* word(T) $= \Lambda\Omega$ *and* word($\sigma(T)$) $= \Lambda 1 \Omega 0$.

Proof. Let ω be the parent of the leaf node with label n in T, this implies that ω, and all of its ancestors are either the root or the right child of their parent. From the definition of word it follows that word(T) is of the form $\Lambda\Omega$ where $\Omega =$ word(ω).

When T is grown left at ω, the new node, ϕ, will take ω as its left child and become the right child of ω's former parent. Consequently, word($\sigma(T)$) will be $\Lambda\Phi$ where $\Phi =$ word(ϕ).

$$\begin{aligned} \text{word}(\phi) &= 1 \circ \text{word}(\text{left}(\phi)) \circ \text{word}(\text{right}(\phi)) \\ &= 1 \circ \text{word}(\omega) \circ 0 \\ \Phi &= 1\Omega 0 \end{aligned}$$

□

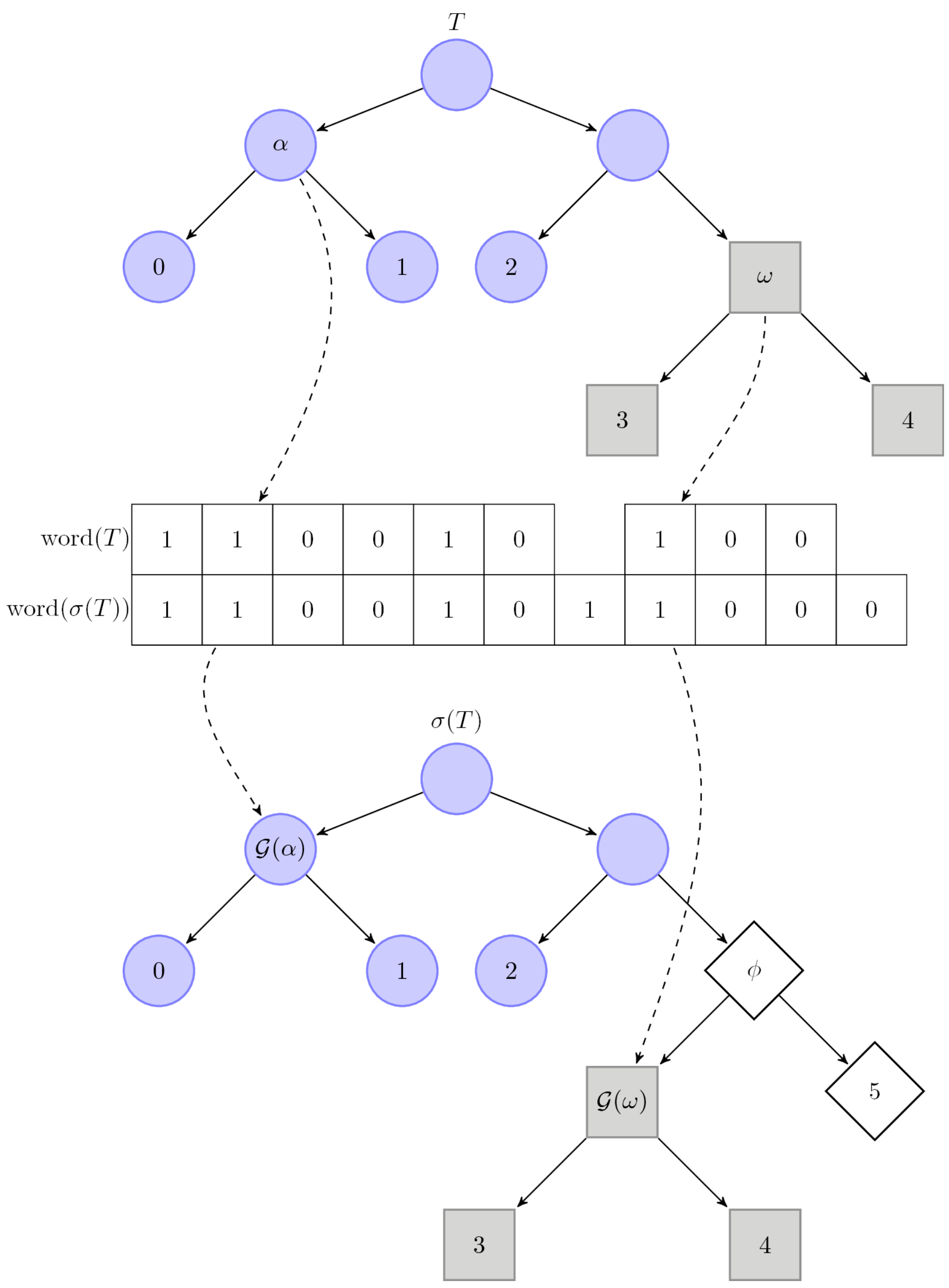

Figure 2: Growing a node and preserving the difficulty of the pair.

There is a natural way in which the nodes in T correspond to the nodes in $\sigma(T)$. The intuition for this is shown in Figure 2. This correspondence can be formalized in terms of the relation between word(T) and word$(\sigma(T))$.

Definition 3. *Let T be an extended ordered binary tree. The natural growth injection of the nodes of T to the nodes of $\sigma(T)$, $\mathcal{G} : \text{nodes}(T) \mapsto \text{nodes}(\sigma(T))$ is defined as*
$\mathcal{G}(\nu) = \text{sym}^{-1}_{\sigma(\mathrm{T})}(\text{word}(\sigma(T))(i + \mathbf{1}\{|\Lambda| < i\}))$
where i is the index of $\text{sym}_T(\nu)$ and $\mathbf{1\{\ldots\}}$ is the indicator function.

There are several properties of $\mathcal{G}$ which will be critical to proving our claim that DPS can always grow a difficult pair (S, T) into another difficult pair (U, V). The first that we will examine relates the interval of a node, ν, in T to the interval of $\mathcal{G}(\nu)$ in $\sigma(T)$.

Lemma 2. *Let T be an extended ordered binary tree of size n, ω be the node of T whose right child is the leaf with label n, ν be any node of T that is not ω and $(\alpha, \beta) = \text{interval}(\nu)$. Then $\text{interval}(\mathcal{G}(\nu)) = (\alpha, \beta + \mathbf{1}\{\beta = n\})$.*

Proof. Let $(\gamma, n) = \text{interval}(\omega)$. By Lemma 1, we have $\text{word}(T) = \Lambda\Omega$ and $\text{word}(\sigma(T)) = \Lambda 1 \Omega 0$. Now we partition $\text{nodes}(T) - \{\omega\}$ into three sets:

1. $\{\nu : (\alpha, \beta) = \text{interval}(\nu), \alpha < \gamma, \beta < n\}$ all nodes that have an interval with a lower bound less than the lower bound of ω and with an upper bound that is less than n.

2. $\{\nu : (\alpha, \beta) = \text{interval}(\nu), \alpha < \gamma, \beta = n\}$ all nodes that have an interval with a lower bound less than the lower bound of ω and with an upper bound equal to n.

3. $\{\nu : (\alpha, \beta) = \text{interval}(\nu), \gamma \leq \alpha\}$ all nodes that have an interval with a lower bound greater than or equal to the lower bound of ω.

We consider the first case. Let N be the subtree rooted at ν, a be the leaf with label α, b be the leaf with label β and g be the leaf with label γ. Since $\alpha < \gamma$, the symbol of a in word(T) must precede the symbol of g in word(T) and therefore $\text{sym}_T(a) \in \Lambda$ and $\text{sym}_T(\nu) \in \Lambda$. By the definition of $\mathcal{G}$ it follows that the index of $\text{sym}_{\sigma(T)}(\mathcal{G}(\nu))$ is the same as the index of $\text{sym}_T(\nu)$ and so $\text{sym}_{\sigma(T)}(\mathcal{G}(\nu)) \in \Lambda$. Therefore, by Proposition 2 the lower bound of $\mathcal{G}(\nu)$ must be α. Since $\beta < n$ it follows $\beta < \gamma$, otherwise g is in N, but this would imply that ω is also in N and so $\beta = n$, which is impossible. Consequently the symbol of b in word(T) must precede the symbol of g in word(T) and therefore $\text{sym}_T(b) \in \Lambda$ and $\text{sym}_{\sigma(T)}(\mathcal{G}(b)) \in \Lambda$. This implies that $\text{word}(\nu) = \text{word}(\mathcal{G}(\nu))$ and so the size of the subtree rooted at $\mathcal{G}(\nu)$ is the same as the size of N. Applying Theorem 3 it follows that the upper bound of $\mathcal{G}(\nu)$ is β. Therefore $\text{interval}(\mathcal{G}(\nu)) = (\alpha, \beta) = (\alpha, \beta + \mathbf{1}\{\beta = n\})$.

Now we consider the second case. Let N be the subtree rooted at ν, a be the leaf with label α, b be the leaf with label β and g be the leaf with label γ. Since $\alpha < \gamma$, the symbol of a in word(T) must precede the symbol of g in word(T) and so

$\text{sym}_T(a) \in \Lambda$ and $\text{sym}_T(\nu) \in \Lambda$. By the definition of $\mathcal{G}$, the index of $\text{sym}_{\sigma(T)}(\mathcal{G}(\nu))$ is the same as the index of $\text{sym}_T(\nu)$ and by Proposition 2 the lower bound of $\mathcal{G}(\nu)$ is α. Since $\beta = n$, ω must be contained in the subtree rooted at ν. Therefore, when T is grown left at ω, the subtree rooted at $\mathcal{G}(\nu)$ will be larger in size by one than N. Applying Theorem 3 it follows that the upper bound of $\mathcal{G}(\nu)$ is $\beta + 1$. And so $\text{interval}(\mathcal{G}(\nu)) = (\alpha, \beta + 1) = (\alpha, \beta + \mathbf{1}\{\beta = n\})$.

Finally, we consider the third case. Let N be the subtree rooted at ν. Since $\gamma \leq \alpha$, ν is a descendant of ω. Therefore $\text{word}(\nu)$ is a proper substring of Ω. We observe that for $|\Lambda| < i \leq |\text{word}(T)|$, we have $\text{word}(T)(i) = \text{word}(\sigma(T))(i+1)$, combined with the definition of $\mathcal{G}$ it follows that $\text{word}(\nu) = \text{word}(\mathcal{G}(\nu))$. So the size of the subtree rooted at $\mathcal{G}(\nu)$ is the same as the size of N. Since the number of '0's which precede $\text{sym}_{\sigma(T)}(\mathcal{G}(\nu))$ is the same as the number of '0's that precede $\text{sym}_T(\nu)$ it follows that the lower bound of $\mathcal{G}(\nu)$ is α. Therefore $\text{interval}(\mathcal{G}(\nu)) = (\alpha, \beta) = (\alpha, \beta + \mathbf{1}\{\beta = n\})$.

□

The growth injection $\mathcal{G}$ also captures several of the node-to-node relationships of T which are preserved in $\sigma(T)$: $\mathcal{G}$ preserves the relation between parents and their left children, $\mathcal{G}$ preserves the relation between parents and their right children, except for the parent of the node whose right child is the leaf labelled n, and taken together, a consequence of the preceding two properties is that $\mathcal{G}$ preserves the relation between parents and children except for the node whose right child is the leaf labelled n. We will next state and prove these relationships formally because we will exploit them in proving the relationship between 1-intervals(T) and 1-intervals($\sigma(T)$).

Lemma 3. *Let T be an extended ordered binary tree of size n and ω be the internal node of T whose right child is the leaf with label n. For any internal node $\nu \in T$, the image of the left child of ν under $\mathcal{G}$ is the left child of the image of ν under $\mathcal{G}$, that is, $\mathcal{G}(\text{left}(\nu)) = \text{left}(\mathcal{G}(\nu))$.*

Proof. It will suffice to show that the index of the symbol of $\mathcal{G}(\text{left}(\nu))$ in the word of $\sigma(T)$ is the same as the index of the symbol of $\text{left}(\mathcal{G}(\nu))$ in the word of $\sigma(T)$. Let i be the index of $\text{sym}_T(\nu)$ and consider two cases as to whether $|\Lambda| < i$ or not.

Case I: $|\Lambda| < i$: the index of $\text{sym}_T(\text{left}(\nu)) = i + 1$ and so the index of $\text{sym}_{\sigma(T)}(\mathcal{G}(\text{left}(\nu))) = i + 2$. By definition, the index of $\text{sym}_{\sigma(T)}(\mathcal{G}(\nu)) = i + 1$ giving index $\text{sym}_{\sigma(T)}(\text{left}(\mathcal{G}(\nu))) = i + 2$.

Case II: $|\Lambda| \geq i$:, we must verify that $i + 1 \leq |\Lambda|$. Since ν is an internal node, its symbol must be a '1' and since the suffix of Λ is the word of the left child of the parent of ω, the last symbol in Λ must be a '0' and so $i \leq |\Lambda| - 1$ which implies $i + 1 \leq |\Lambda|$. Now the index of $\text{sym}_T(\text{left}(\nu)) = i + 1$. Since $i + 1 < |\Lambda|$ the index of $\text{sym}_{\sigma(T)}(\mathcal{G}(\text{left}(\nu))) = i + 1$. By definition, the index of $\text{sym}_{\sigma(T)}(\mathcal{G}(\nu)) = i$ and so the index of $\text{sym}_{\sigma(T)}(\text{left}(\mathcal{G}(\nu))) = i + 1$. □

Lemma 4. *Let T be an extended ordered binary tree of size n and ω be the internal node of T whose right child is the leaf with label n. For any internal node $\nu \in T$,* *except for $\pi(\omega)$, the image of the right child of ν under $\mathcal{G}$ is the right child of the image of ν under $\mathcal{G}$, that is $\mathcal{G}(\mathrm{right}(\nu)) = \mathrm{right}(\mathcal{G}(\nu))$.*

Proof. It will suffice to show that the index of the symbol of $\mathcal{G}(\mathrm{right}(\nu))$ in the word of $\sigma(T)$ is the same as the index of the symbol of $\mathrm{right}(\mathcal{G}(\nu))$ in the word of $\sigma(T)$. Let i be the index of $\mathrm{sym}_T(\nu)$ and consider two cases as to whether $|\Lambda| < i$ or not.

Case I: $|\Lambda| < i$: the index of $\mathrm{sym}_T(\mathrm{right}(\nu)) = i + |\mathrm{word}(\mathrm{left}(\nu))| + 1$ and so the index of $\mathrm{sym}_{\sigma(T)}(\mathcal{G}(\mathrm{right}(\nu))) = i + |\mathrm{word}(\mathrm{left}(\nu))| + 2$. By definition, the index of $\mathrm{sym}_{\sigma(T)}(\mathcal{G}(\nu)) = i + 1$ and so the index of $\mathrm{sym}_{\sigma(T)}(\mathrm{right}(\mathcal{G}(\nu))) = i + |\mathrm{word}(\mathrm{left}(\mathcal{G}(\nu)))| + 2$. Observe that the upper bound of the interval of $\mathrm{left}(\nu)$ must be less than n since it is the left child of its parent. Applying Lemmas 2 and 3 it follows that $\mathrm{interval}(\mathrm{left}(\nu)) = \mathrm{interval}(\mathcal{G}(\mathrm{left}(\nu))) = \mathrm{interval}(\mathrm{left}(\mathcal{G}(\nu)))$. Therefore the size of the subtree rooted at $\mathrm{left}(\nu)$ is the same as the size of the subtree rooted at $\mathrm{left}(\mathcal{G}(\nu))$ and so $|\mathrm{word}(\mathrm{left}(\nu))| = |\mathrm{word}(\mathrm{left}(\mathcal{G}(\nu)))|$

Case II: $|\Lambda| \geq i$: we must verify that $i + |\mathrm{word}(\mathrm{left}(\nu))| + 1 \leq |\Lambda|$. But this is clearly true since the only case in which $|\Lambda| < i + |\mathrm{word}(\mathrm{left}(\nu))| + 1$ is when $\nu = \pi(\omega)$, but we have excluded $\pi(\omega)$ from consideration. Observe that the index of $\mathrm{sym}_T(\mathrm{right}(\nu)) = i + |\mathrm{word}(\mathrm{left}(\nu))| + 1$ and so the index of $\mathrm{sym}_{\sigma(T)}(\mathcal{G}(\mathrm{right}(\nu))) = i + |\mathrm{word}(\mathrm{left}(\nu))| + 1$. By definition, the index of $\mathrm{sym}_{\sigma(T)}(\mathcal{G}(\nu)) = i$ and so the index of $\mathrm{sym}_{\sigma(T)}(\mathrm{right}(\mathcal{G}(\nu))) = i + |\mathrm{word}(\mathrm{left}(\mathcal{G}(\nu)))| + 1$. Since $\mathrm{left}(\nu)$ is a left child the upper bound of its interval must be less than n. Applying lemmas 2 and 3 it follows that $\mathrm{interval}(\mathrm{left}(\nu)) = \mathrm{interval}(\mathcal{G}(\mathrm{left}(\nu))) = \mathrm{interval}(\mathrm{left}(\mathcal{G}(\nu)))$. This implies that the size of the subtree rooted at $\mathrm{left}(\nu)$ is the same as the size of the subtree rooted at $\mathrm{left}(\mathcal{G}(\nu))$ and so $|\mathrm{word}(\mathrm{left}(\nu))| = |\mathrm{word}(\mathrm{left}(\mathcal{G}(\nu)))|$. □

Together Lemma 3 and Lemma 4 show the preserved parent structure.

Corollary 1. *Let T be an extended ordered binary tree of size n and ω be the internal node of T whose right child is the leaf with label n. For any node $\nu \in \mathrm{nodes}(T) - \{\omega\}$, the image of the parent of ν under $\mathcal{G}$ is the parent of the image of ν under $\mathcal{G}$, that is, $\mathcal{G}(\pi(\nu)) = \pi(\mathcal{G}(\nu))$.*

The next lemma will relate $\mathrm{intervals}(T)$ to $\mathrm{intervals}(\sigma(T))$.

Lemma 5. *Let T be an extended ordered binary tree of size n and ω be the internal node of T whose right child is the leaf with label n. Then the intervals of $\sigma(T)$ are related to the intervals of T by*

$$\mathrm{intervals}(\sigma(T)) = \{(\alpha, \beta + \mathbf{1}\{\beta = n\}) : (\alpha, \beta) \in \mathrm{intervals}(T)\} \cup \{\mathrm{interval}(\omega)\}. \quad (1)$$

Proof. Let $(\gamma, n) = \mathrm{interval}(\omega)$. By Lemma 2 we have the intervals for the internal nodes of $\sigma(T)$ that are the image under $\mathcal{G}$ of some node in T, except for ω. This gives us $n-1$ intervals of $\sigma(T)$ and we have only to consider the interval of $\mathcal{G}(\omega)$ and the interval of $\pi(\mathcal{G}(\omega))$. By Lemma 1 the number of '0's which precede $\mathrm{sym}_{\sigma(T)}(\mathcal{G}(\omega))$ is the same as the number of '0's which precede $\mathrm{sym}_{\sigma(T)}(\pi(\mathcal{G}(\omega)))$ which is the same as the number of '0's which precede $\mathrm{sym}_T(\omega)$ and so the intervals of $\mathcal{G}(\omega)$ and

$\pi(\mathcal{G}(\omega))$ have the same lower bound, namely γ, which is the lower bound of ω. By construction, the subtree rooted at $\mathcal{G}(\omega)$ has the same size as the subtree rooted at ω. Applying Proposition 1 it follows that interval$(\mathcal{G}(\omega))$ = interval(ω). Also by construction, the subtree rooted at $\pi(\mathcal{G}(\omega))$ is greater in size by 1 than the subtree rooted at ω. Applying Proposition 1 again yields interval$(\pi(\mathcal{G}(\omega))) = (\gamma, n+1) = (\alpha, \beta + \mathbf{1}\{\beta = n\})$. □

With the relationship between intervals$(\sigma(T))$ and intervals(T) proven, we can state and prove the relationship between 1-intervals$(\sigma(T))$ and 1-intervals(T). Our proof strategy will be to determine for each internal node of T, ν, the local structure that determines the 1-interval(ν) in T. Then we will determine how that local structure maps to the local structure of $\mathcal{G}(\nu)$ in $\sigma(T)$ and use this to compute 1-interval$(\mathcal{G}(\nu))$.

Lemma 6. *Let T be an extended ordered binary tree of size n, ω the internal node of T whose right child is the leaf with label n, and ϕ the internal node of $\sigma(T)$ that is the parent of $\mathcal{G}(\omega)$. Then the* 1-intervals *of $\sigma(T)$ are related to the* 1-intervals *of T by*

$$\begin{aligned}\text{1-intervals}(\sigma(T)) =&\{(\alpha, \beta + \mathbf{1}\{\beta = n\}) : (\alpha, \beta) \in \Theta\}\\ &\cup\{\text{1-interval}(\text{left}(\omega)), (n, n+1), \text{1-interval}(\phi)\}\end{aligned}$$

where Θ = 1-intervals$(T) - \{$1-interval(ω), 1-interval$(\text{left}(\omega))\}$.

Proof. Let (γ, n) = interval(ω) and apply lemma 1 to obtain word$(T) = \Lambda\Omega$ and word$(\sigma(T)) = \Lambda 1 \Omega 0$. Now we partition nodes$(T)$ into 6 sets:

1. $\{\nu : (\delta, n) = \text{interval}(\nu), \nu \notin \{\omega, \text{root}(T)\}\}$ every node, excluding the root and ω, whose interval has an upperbound of n.
2. $\{\nu : (\delta, \beta) = \text{interval}(\pi(\nu)), \beta < n, \nu = \text{left}(\pi(\nu))\}$ every node whose parent's interval has an upperbound less than n and that is a left child of its parent.
3. $\{\nu : (\alpha, \delta) = \text{interval}(\pi(\nu)), \delta < n, \nu = \text{right}(\pi(\nu))\}$ every node whose parent's interval has an upperbound less than n and that is a right child of its parent.
4. $\{\nu : (\delta, n) = \text{interval}(\pi(\nu)), (\kappa, n) \neq \text{interval}(\nu), \nu \neq \text{left}(\omega)\}$ every node, ν, excluding the left child of ω, such that the upper bound of the interval of ν is not n and the upper bound of the interval of ν's parent is n.
5. $\{\omega\}$ the singleton set containing ω.
6. $\{\text{left}(\omega)\}$ the singleton set containing the left child of ω.

We proceed analyzing these six sets:

Case 1: Because we have excluded the root, the parent of ν is a node in T and its right child must be ν. Since the 1-interval$(\nu) = (\alpha, \beta)$ is obtained by taking the lower bound of ν's parent and the upper bound of the left child of ν it follows that interval$(\pi(\nu)) = (\alpha, n)$ and interval(left$(\nu)) = (\delta, \beta)$. Since we have excluded ω, α, β and δ must be less than γ. Applying Lemma 2, Lemma 3 and Corollary 1 we have interval$(\mathcal{G}(\nu)) = (\delta, n+1)$, interval$(\mathcal{G}(\pi(\nu)))$ = interval$(\pi(\mathcal{G}(\nu))) = (\alpha, n+1)$ and interval$(\mathcal{G}($left$(\nu)))$ = interval(left$(\mathcal{G}(\nu))) = (\delta, \beta)$. It follows that 1-interval$(\mathcal{G}(\nu)) = (\alpha, \beta) = (\alpha, \beta + \mathbf{1}\{\beta = n\})$ = 1-interval(ν).

Case 2: interval$(\nu) = (\delta, \epsilon)$ and interval(right$(\nu)) = (\alpha, \epsilon)$ for some ϵ and α. Since $\beta < n$, applying Lemma 2, Lemma 4 and Corollary 1 yields interval$(\mathcal{G}(\nu))$ = interval(ν), as well as interval$(\mathcal{G}(\pi(\nu)))$ = interval$(\pi(\mathcal{G}(\nu)))$ and also interval(right$(\mathcal{G}(\nu)))$ = interval(right$(\nu))$. Computing 1-interval of $\mathcal{G}(\nu)$ yields 1-interval$(\mathcal{G}(\nu)) = (\alpha, \beta) = (\alpha, \beta + \mathbf{1}\{\beta = n\})$ = 1-interval(ν).

Case 3: interval$(\nu) = (\epsilon, \delta)$ and interval(left$(\nu)) = (\epsilon, \beta)$ for some ϵ and β. Since $\delta < n$, applying Lemma 2, Lemma 3 and Corollary 1 yields interval$(\mathcal{G}(\nu))$ = interval(ν), interval$(\mathcal{G}(\pi(\nu)))$ = interval$(\pi(\nu))$, and also that interval(left$(\mathcal{G}(\nu)))$ = interval(left$(\nu))$. Therefore, 1-interval$(\mathcal{G}(\nu)) = (\alpha, \beta) = (\alpha, \beta + \mathbf{1}\{\beta = n\})$ = 1-interval(ν).

Case 4: Because the upperbound of ν is not n, it follows ν is the left child of its parent, and so (δ, ϵ) = interval(ν) and (α, ϵ) = interval(right$(\nu))$ for some ϵ and α such that $\epsilon < n$. Consequently 1-interval$(\nu) = (\alpha, n)$ for some α and interval(right$(\nu)) = (\alpha, \epsilon)$. Applying Lemma 2, Lemma 4 and Corollary 1 we have that interval$(\mathcal{G}(\nu))$ = interval(ν), interval$(\pi(\mathcal{G}(\nu))) = (\delta, n+1)$ and interval(right$(\mathcal{G}(\nu)))$ = interval(right$(\nu))$ therefore, 1-interval$(\mathcal{G}(\nu)) = (\alpha, n+1) = (\alpha, \beta + \mathbf{1}\{\beta = n\})$.

Case 5: interval$(\mathcal{G}(\omega)) = (\gamma, n)$, interval$(\pi(\mathcal{G}(\omega))) = (\gamma, n+1)$ and the right child of $\mathcal{G}(\omega)$ is the leaf with label n. Therefore, 1-interval$(\mathcal{G}(\omega)) = (n, n+1)$.

Case 6: By definition, we have $\pi($left$(\omega)) = \omega$ and so interval$(\pi($left$(\omega)))$ = interval$(\omega) = (\gamma, n)$. Because the right child of ω is the leaf with label n it follows that interval(left$(\omega)) = (\gamma, n-1)$. Therefore interval(right(left$(\omega))) = (\alpha, n-1)$ for some α and 1-interval(left$(\omega)) = (\alpha, n)$. By Lemma 2 we have interval$(\mathcal{G}($left$(\omega)))$ = interval(left$(\omega))$. By Lemma 5 and Corollary 1 we have that interval$(\mathcal{G}(\omega))$ = interval(ω). By Lemmas 2 and Lemma 3, we have interval(right$(\mathcal{G}($left$(\omega))))$ = interval(right(left$(\omega)))$, and thus 1-interval$(\mathcal{G}($left$(\omega)))$ = 1-interval(left$(\omega))$.

The preceding case analysis computes the 1-interval for every internal node of $\sigma(T)$ that is the image under $\mathcal{G}$ of some node ν in T. In order to complete the 1-intervals$(\sigma(T))$ and the proof, we add 1-interval(ϕ). □

With these substitution rules for obtaining the intervals and 1-intervals of a tree $\sigma(T)$ from the intervals and 1-intervals of T, we can now proceed to show that if (S, T) is a difficult pair, then so too is $(\sigma(S), \sigma(T))$. We will do so by first showing that the pair $(\sigma(S), \sigma(T))$ do not have a common interval between them and secondly that they have no one-off intervals between them either.

Lemma 7. *Let (S, T) be a difficult pair of extended ordered binary trees of size n, let ω_S be the internal node of S whose right child is the leaf with label n, and let ω_T be the internal node of T whose right child is the leaf with label n. Then the pair of trees $(\sigma(S), \sigma(T))$ do not have an interval in common.*

Proof. Assume, to the contrary, that the pair $(\sigma(S), \sigma(T))$ have an interval in common. Then some interval, call it t, in intervals$(\sigma(T))$ is also in intervals$(\sigma(S))$. We consider the possible forms of t given by Lemma 5. If $t = \text{interval}(\omega_T)$, then $t = (\alpha, n)$ and by Lemma 5, interval$(\omega_S) = t$ which implies (S, T) is not a difficult pair. Otherwise, $t = (\alpha, \delta)$, is some other interval in intervals$(\sigma(T))$. If $\delta = n + 1$, then the interval (α, n) is in both intervals(S) and intervals(T) which implies (S, T) is not a difficult pair. Otherwise, $\delta < n$ and so (α, δ) is in intervals(S) and intervals(T) and (S, T) is not a difficult pair. □

Lemma 8. *Let (S, T) be a difficult pair of extended ordered binary trees of size n, let ω_S be the internal node of S whose right child is the leaf with label n, and let ω_T be the internal node of T whose right child is the leaf with label n. Then the pair of trees $(\sigma(S), \sigma(T))$ have no one-off intervals between them.*

Proof. Assume, to the contrary, that the pair $(\sigma(S), \sigma(T))$ have some one-off interval between them. Without loss of generality, let t be the 1-interval of $\sigma(T)$ that is also an interval of $\sigma(S)$ and consider the possible forms of t given by Lemma 6. By construction, neither $\sigma(S)$ nor $\sigma(T)$ can have the interval $(n, n+1)$ and so $t \neq (n, n+1)$. If $t = \text{1-interval}(\text{left}(\omega_T))$, then $t = (\lfloor \text{right}(\text{left}(\omega_T)) \rfloor, n) \in \text{intervals}(\sigma(S))$ but then Lemma 5 implies $t = \text{interval}(\omega_S)$ and so (S, T) is not a difficult pair. If $t = \text{1-interval}(\phi)$ then $t = (\lfloor \pi(\phi_T) \rfloor, n) = (\lfloor \pi(\omega_T) \rfloor, n) = \text{interval}(\pi(\omega_T))$ but then Lemma 5 again implies $t = \text{interval}(\omega_S)$ and so (S, T) is not a difficult pair. If $t = (\alpha, n + 1)$, then $(\alpha, n) \in \text{1-intervals}(T)$ and $(\alpha, n) \in \text{intervals}(S)$ and (S, T) is not a difficult pair. Otherwise, $t = (\alpha, \delta)$ such that $\delta < n$, therefore $(\alpha, \delta) \in \text{1-intervals}(T)$ and $(\alpha, \delta) \in \text{intervals}(S)$ and so (S, T) is not a difficult pair. □

We have now established the fact that the pair $(\sigma(S), \sigma(T))$ is a difficult tree pair which underlies the correctness of DPS.

Theorem 1. *Let (S, T) be a difficult pair of extended ordered binary trees of size n, then the pair $(\sigma(S), \sigma(T))$ of extended ordered binary trees of size $n + 1$ is a difficult pair.*

Proof. Immediate from Lemma 7 and Lemma 8. □

Theorem 2. *Let n be a natural number greater or equal to 4, then the Difficult Pair Sampling algorithm is guaranteed to return a difficult pair of trees of size n.*

Proof. We proceed by induction on n the size of the trees in the difficult pair desired. In the base case, $n = 4$, in which case, DPS samples one of the 4 primitive difficult pairs of trees which have been found by easy enumeration.

Now we suppose that the Difficult Pair Sampling algorithm is guaranteed to return a difficult pair of trees for all $m < n$ such that $m, n \in \mathbb{N}$, $4 < n$, and let (S, T) be a difficult pair of trees of size $n - 1$ sampled by DPS. By Theorem 1 there is at least one pair of difficult trees in the set of all pairs of growth neighbors of S and T which DPS will find by enumeration. Consequently, DPS is guaranteed to return a difficult pair of trees of size n. □

4 Time Complexity of DPS

We now analyze the time complexity of DPS and show the following:

Theorem 3. *The Difficult Pair Sampling algorithm runs in $O(n^4)$ time, where n is the size of the desired difficult tree pair.*

Proof. Line 1 can be implemented to run in constant time by using a table of the primitive difficult pairs. By returning a pair of pointers line 9 can also be implemented to run in constant time. Therefore the time complexity of DPS is determined by the `for` loop of lines 2 through 8. We name the `for` loops as follows: let f be the for loop of lines 2 through 8, g be the `for` loop of lines 4 through 7, and h be the `for` loop of lines 5 through 7.

We consider one iteration of f with (S, T) being the current difficult pair and suppose we use a table, t, to hold the pairs of difficult growth neighbors of (S, T). Given n, we bound the maximum size of t by the space required for the maximum number of pairs of difficult growth neighbors of size n and so preallocate the space. The space complexity of t is $O(n^3)$. If, on each iteration of f, the candidate difficult growth neighbor pairs are stored from the beginning of the table contiguously, then line 3 can be implemented to run in constant time by starting again at the beginning of the table and keeping track of how many rows have been filled. Further, with such a scheme, line 8 can be implemented to run in $O(n)$ time by randomly selecting a row (in constant time) of one of the candidate difficult pairs and then copying the selected candidates (in linear time) to the space allocated for the current pair. The time complexity of one iteration of f is therefore $O(n + g)$.

The time complexity of one iteration of g is the sum of the time required to compute one growth neighbor of S and the time complexity of h. Using the word representation of an extended ordered binary tree, it is possible to compute a growth neighbor, including its intervals and 1-intervals, in linear time. Thus, one iteration of g takes $O(n + h)$ steps.

The time complexity of one iteration of h is the sum of the time complexity of computing one growth neighbor of T, the time complexity of checking if the resulting pair of growth neighbors, (U, V), is difficult, and the time complexity of adding (U, V) to *choices*. Now suppose we allocate two, two-dimensional tables, a and b, where we use a to store intervals(S) $\cup$ 1-intervals(T) and b to store intervals(T) $\cup$ 1-intervals(S). These tables will require at most $O(n^2)$ space. If we populate these tables as we construct the growth neighbors, then we can determine whether the pair (U, V) is difficult as we construct V and set a flag appropriately. Then line 6 can be implemented to run in constant time by checking the flag. Assuming the candidate pairs are being stored in table t, then adding the pair (U, V) to *choices* can be a constant time increment operation. So the time complexity of one iteration of h is $O(n)$.

The number of iterations of h is determined by the number of growth neighbors of T. As previously mentioned, for a tree of size n, a straightforward upper bound on the number of growth neighbors is $3n + 1$. Hence, h will execute $O(n)$ times and $O(h) = O(n^2)$. The same reasoning shows that g will also execute $O(n)$ times and so $O(g) = O(n^3)$. Finally, it is clear that f also executes $O(n)$ times and so $O(f) = O(n^4) = O(\text{DPS})$. $\square$

5 Sampling coverage of DPS

An ideal sampling algorithm not only has the potential to produce all possible instances of interest, but also produces such interests uniformly at random. One of the excellent features of Remy's algorithm for generating trees is that it selects a given tree uniformly at random from all trees and is ideal for many purposes. This DPS algorithm does not have such uniformity, with some difficult tree pairs being sampled more often than others. Since the number of difficult tree pairs of a particular size is not known exactly, the degree of non-uniformity is difficult to calculate exactly.

From work of Cleary, Elder, Rechnitzer and Taback [2], the fraction of difficult pairs (and in fact its superset, the set of reduced tree pairs) goes to zero exponentially quickly as the size of the tree pairs increase. Calculations by Cleary and Maio [4] show that the number of difficult pairs appear to grow exponentially with an exponential growth rate of about 2.17975, out of the set of equivalence classes of all pairs with an exponential growth rate of about 2.4420, giving a fraction of difficult pairs of less than one in million by size 44 and dropping with increasing size. This is consistent with the observation (proven by Cleary, Rechnitzer and Wong [5]) that for large n, the chance of selecting a difficult tree pair at random is vanishingly small.

As far as the completeness of coverage, for small n where feasible, we found that the DPS algorithm does sample from all hard cases. We considered tens of millions of cases and did not find any difficult instances which were not produced at least once by the DPS algorithm. However, though this gives evidence that the coverage might be complete, this is by no means ensured. There is the potential for there to be difficult tree pairs of size k with no growth neighbors of size $k-1$. If such difficult pairs existed, they would not be produced by the algorithm ever. We did not encounter any such problematic growth pairs in our computational experiments, but those experiments are necessarily of limited scope and the question remains for later investigation about the completeness of the sampling coverage.

The number of distinct difficult pairs for larger n appears to grow exponentially but the growth is not known exactly and even the exponential rate of growth is not known. The only current means of producing all difficult pairs is equivalent in running time to exhaustive enumeration of all tree pairs and is not feasible beyond small values, so it is not computationally feasible to check to see if instances of even moderate size are not produced by the DPS algorithm. For example, using some natural notions of equivalence described in Cleary and Maio [4], out of 7,152,629,313,600 tree pairs of size 14, there are 17,561,480,528 difficult tree pairs. It is not computationally feasible to test to see if the DPS algorithm will generate all possible instances of that size.

We note that the examples produced by DPS lie in starkly broader classes of difficult pairs than those specific known earlier examples of Dehornoy [8], Pournin [11], and Cleary and Maio [3]. Those earlier examples give difficult tree pairs of increasingly large sizes but though there are multiple possible examples of increasing size, these numbers do not grow nearly as fast as the set of all possible difficult pairs or as those constructed here. Those examples were constructed for different purposes and there was no intent to get broad coverage of representative difficult instances.

If it is the case that there is complete coverage, the question of the degree of uniformity is of interest. By "degree of uniformity" we mean some indication of the differences in rates of difficult pairs of the same size from being chosen at random. This question has some complications because some of the difficult pairs have certain symmetries arising which raise questions about what exactly is meant in these instances: uniform sampling of instances or of equivalence classes of instances with respect to some natural symmetries arising from the dihedral symmetries of regular polygons. In any case, as far as the degree of uniformity, computations for small n with exhaustive coverage show essentially complete coverage of difficult instances with factors in the range of 2 between the first and third quartiles of number of instances and a factor of about 7 between the most commonly and least commonly generated. There is no reason to presume that this DPS algorithm gives uniformity and these modest experiments show that to be extremely unlikely.

We note that though it is not clear as to the coverage, the DPS algorithm gives a range of difficult pairs. In general, there are many instances where random instances of a problem are by no means representative and often can be attacked with tools that do not capture the essential difficulty. For example, randomly chosen tree pairs do not form good test cases for rotation distance algorithms since with significant probabilities, randomly-selected tree pairs have already many common edges– on average $(16/\pi-5)n \sim 0.093n$ as proven by Cleary, Rechnitzer, and Wong [5]. Furthermore, randomly selected tree pairs generally have many possible one-off edge reductions as well – also on average $(16/\pi-5)n \sim 0.093n$ proven by Cleary, Rechnitzer and Wong [5] after compelling numerical evidence of Chu and Cleary [1] estimating those fractions numerically. There are two previously considered algorithms for generating hard cases of increasing size and neither works effectively. The "test and reject" sampling algorithm does give uniform coverage of instances, but the number of needed iterations grows exponentially with size as the scarcity of difficult pairs falls. The "test and reduce to the largest possible difficult subproblem" is not computationally efficient, may not give instances of a prescribed size or even in a prescribed size range, and may well not have uniform coverage of difficult instances. So the DPS algorithm does give a rich range of possible instances in a computationally efficient manner compared to previously known algorithms.

References

[1] Chu, Timothy and Cleary, Sean. Expected conflicts in pairs of rooted binary trees. *Involve*, 6(3):323–332, 2013. DOI: `10.2140/involve.2013.6.323`.

[2] Cleary, Sean, Elder, Murray, Rechnitzer, Andrew, and Taback, Jennifer. Random subgroups of Thompson's group F. *Groups Geom. Dyn.*, 4(1):91–126, 2010. DOI: 10.4171/GGD/76.

[3] Cleary, Sean and Maio, Roland. Edge conflicts do not determine geodesics in the associahedron. *SIAM J. Discrete Math.*, 32(2):1003–1015, 2018. DOI: 10.1137/17M1114582.

[4] Cleary, Sean and Maio, Roland. Counting difficult tree pairs with respect to the rotation distance problem. *J. Combin. Math. Combin. Comput.*, 115:199–213, 2020. DOI: 10.1080/01621459.2018.1537922.

[5] Cleary, Sean, Rechnitzer, Andrew, and Wong, Thomas. Common edges in rooted trees and polygonal triangulations. *Electron. J. Combin.*, 20(1):Paper 39, 22, 2013. DOI: 10.37236/2541.

[6] Cleary, Sean and St. John, Katherine. Rotation distance is fixed-parameter tractable. *Inform. Process. Lett.*, 109(16):918–922, 2009. DOI: 10.1016/j.ipl.2009.04.023.

[7] Culik, II, Karel and Wood, Derick. A note on some tree similarity measures. *Inform. Process. Lett.*, 15(1):39–42, 1982. DOI: 10.1016/0020-0190(82)90083-7.

[8] Dehornoy, Patrick. On the rotation distance between binary trees. *Adv. Math.*, 223(4):1316–1355, 2010. DOI: 10.1016/j.aim.2009.09.016.

[9] Hanke, Sabine, Ottmann, Thomas, and Schuierer, Sven. The edge-flipping distance of triangulations. *J.UCS*, 2(8):570–579, 1996.

[10] Knuth, Donald E. *The art of computer programming. Volume 3.* Addison-Wesley Publishing Co., Reading, Mass.-London-Don Mills, Ont., 1973. Sorting and searching, Addison-Wesley Series in Computer Science and Information Processing.

[11] Pournin, Lionel. The diameter of associahedra. *Adv. Math.*, 259:13–42, 2014. DOI: 10.1016/j.aim.2014.02.035.

[12] Rémy, Jean-Luc. Un procédé itératif de dénombrement d'arbres binaires et son application à leur génération aléatoire. *RAIRO Inform. Théor.*, 19(2):179–195, 1985. DOI: 10.1051/ita/1985190201791.

[13] Sleator, Daniel D., Tarjan, Robert E., and Thurston, William P. Rotation distance, triangulations, and hyperbolic geometry. *J. Amer. Math. Soc.*, 1(3):647–681, 1988. DOI: 10.2307/1990951.

Received: 16th February 2020