Designing ReLU Generative Networks to Enumerate Trees with a Given Tree Edit Distance

Mamoona Ghafoor and Tatsuya Akutsu
Bioinformatics Center, Institute for Chemical Research, Department of Intelligence Sciences and Technology, Kyoto University, Uji 611-0011, Japan
Email: mamoona.ghafoor@kuicr.kyoto-u.ac.jp; takutsu@kuicr.kyoto-u.ac.jp

Abstract

The generation of trees with a specified tree edit distance has significant applications across various fields, including computational biology, structured data analysis, and image processing. Recently, generative networks have been increasingly employed to synthesize new data that closely resembles the original datasets. However, the appropriate size and depth of generative networks required to generate data with a specified tree edit distance remain unclear. In this paper, we theoretically establish the existence and construction of generative networks capable of producing trees similar to a given tree with respect to the tree edit distance. Specifically, for a given rooted, ordered, and vertex-labeled tree T of size $n+1$ with labels from an alphabet Σ, and a non-negative integer d, we prove that all rooted, ordered, and vertex-labeled trees over Σ with tree edit distance at most d from T can be generated using a ReLU-based generative network with size $\mathcal{O}(n^3)$ and constant depth. The proposed networks were implemented and evaluated for generating trees with up to 21 nodes. Due to their deterministic architecture, the networks successfully generated all valid trees within the specified tree edit distance. In contrast, state-of-the-art graph generative models GraphRNN and GraphGDP, which rely on non-deterministic mechanisms, produced significantly fewer valid trees, achieving validation rates of only up to 35% and 48%, respectively. These findings provide a theoretical foundation towards construction of compact generative models and open new directions for exact and valid tree-structured data generation. An implementation of the proposed networks is available at `https://github.com/MGANN-KU/TreeGen_ReLUNetworks`.

Keywords: Generative networks; ReLU function; trees; tree edit distance; enumeration; Euler string

1 Introduction

Over the past few years, generative networks have been widely studied due to its vast applications in different fields such as natural language processing, data augmentation, DNA sequence synthesis, and drug discovery [1–4]. Generative networks are a class of machine learning models that learn the underlying patterns, structures, and dependencies in training data. By capturing this statistical information, they can generate new data samples that resemble the original data. These models are not limited to data synthesis but are also used in tasks such as data augmentation, imputation, and representation learning. Applications include generating realistic images, text, audio, and other complex data modalities [5–7]. For example, the application of generative models in bioinformatics includes motif discovery, secondary structure prediction, drug discovery, cancer research, the generation of new molecules and the analysis of single-cell RNA sequencing data [8], [9], [10], [11].

There are various types of generative models, each with distinct characteristics. Autoencoder-based models include variational autoencoders (VAEs) [12] and denoising autoencoders (DAEs) [13], which are designed to learn compact representations of data by encoding and decoding it. Generative adversarial networks (GANs) [14], including specialized versions like deep convolutional generative adversarial networks (DCGANs) [15], use adversarial training to generate realistic data by having a generator and discriminator compete against each other. Deep belief networks, such as deep Boltzmann machines (DBMs) [16], are probabilistic models that represent complex data distributions through a stack of restricted Boltzmann machines. Generative stochastic networks (GSNs) [17] use stochastic processes to generate data by iteratively refining it. Autoregressive models, including pixel convolutional neural networks (PixelCNN) [18] and pixel recurrent neural networks (PixelRNN) [19], model the distribution of image pixels, generating data pixel by pixel in a sequential manner. The deep recurrent attentive writer (DRAW) model [20] combines recurrent

neural networks and attention mechanisms to generate images, focusing on specific parts of the data during the generation process. Diffusion models have recently gained significant popularity in generative AI. These models generate data by first learning how to gradually add noise to real data until it becomes random noise. Then, they are trained to reverse this process, step by step, by removing the noise and recovering the original data distribution. During generation, the model starts with pure noise and progressively denoises it to produce a realistic sample, like an image or audio clip. This step-by-step denoising is guided by a neural network, often trained to predict the noise added at each stage [21], [22].

Selecting the right function family and network model is essential in machine learning. A function family that is too broad may cause high computational costs and overfitting, while a limited one might not produce accurate predictions [23]. Choosing the appropriate network remains challenging, as there is no clear choice for every problem. The universal approximation theorem states that a two-layer neural network can approximate any Borel measurable function [24]. But such networks often require a large number of nodes. Studies have examined how the choice of function family relates to network size, revealing that deeper networks significantly enhance representational power [25,26]. Not all functions can be efficiently represented by any architecture. For instance, Telgarsky [27] identified functions requiring exponentially more nodes in shallow networks compared to deep ones. Szymanski and McCane [28] showed that deep networks are well-suited for modeling periodic functions, while Chatziafratis et al. [29] established width lower bounds based on depth for such functions. Hanin and Rolnick [30] further found that networks with piecewise linear activation functions do not exponentially increase their expressive regions. Additionally, Bengio et al. [31] and Biau et al. [32] demonstrated that decision trees and random forests can be approximated by neural networks using sigmoidal, Heaviside, or tanh activation functions. Kumano and Akutsu [23] later extended this result to networks using ReLU and similar activations. Recently, Ghafoor and Akutsu [33] discussed the existence of generative networks with ReLU as activation function and constant depth to generate similar strings with a given edit distance.

Selkow [34] introduced the problem of tree edit distance as a generalization of the classical string edit distance problem. The tree edit distance problem has several applications in applied fields including computational biology [35–38], analysis of structured data [39–41], and image processing [42–45]. Different algorithms have been developed to compute the tree edit distance between two rooted, labeled and ordered trees. For instance, Tai [46] proposed an algorithm for the tree edit problem with time complexity $\mathcal{O}(n^6)$, where n is the size of the underlying tree. Zhang and Shasha [47], Klein [48] and Demaine et al. [49] proposed improved algorithms with a time complexity $\mathcal{O}(n^4)$, $\mathcal{O}(n^3 \log n)$ and $O(n^3)$, respectively. Later on Bringmann et al. [50] further improved the complexity to $\mathcal{O}(n^{3-\epsilon})$ for weighted trees. In 2022, Mao [51] introduced an algorithm for unweighted trees with complexity $\mathcal{O}(n^{2.9148})$. Recently, Nogler et al. [52], proposed an efficient algorithm with complexity $\mathcal{O}(n^{3/2\Omega(\sqrt{\log n})})$ for weighted trees and $\mathcal{O}(n^{2.6857})$ for unweighted trees.

Recent years have witnessed significant advancements in structured generative networks, with a growing focus on models that balance empirical performance and structural validity. GraphRNN is a foundational autoregressive model that generates graphs by sequentially adding nodes and edges, widely used in molecular design [53]. Building on this, Wang et al. [54] proposed a variational autoregressive model that learns generation order dynamically, achieving state-of-the-art molecular graph results without diffusion. AutoGraph [55] applies transformers to autoregressively generate graphs as sequences. TreeGAN by Liu et al. [56] is a syntax-aware generative adversarial network designed for sequence generation that respects tree-structured syntactic constraints. In parallel, diffusion-based generative models have recently advanced structured data generation. For instance, Huang et al. [57] proposed a continuous-time generative diffusion model, GraphGDP, to generate permutation-invariant graphs. Liu et al. [58] introduced a beta-noise process that effectively models both discrete graph structures and continuous node attributes, achieving strong results on biochemical and social network benchmarks. The framework by Madeira et al. [59] enforced hard structural constraints such as planarity or acyclicity via an edge-absorbing noise mechanism, ensuring generated graphs rigorously maintain desired properties throughout the diffusion process. While existing generative models offer powerful empirical frameworks for producing high-quality and diverse structured data, their guarantees are inherently probabilistic and data-dependent. These methods require training on limited datasets, and their performance is influenced by the quality and coverage of this data. As a result, they cannot provide exact enumeration of the underlying combinatorial space, nor can they ensure complete validity or coverage of all possible structured instances.

As a step towards addressing these issues we study the exact and deterministic generation of all rooted,

ordered, and vertex-labeled trees similar to a given tree using neural networks and prove the existence of ReLU-based generative networks for tree edit distance. Given a rooted, ordered, and vertex-labeled tree T of size $n+1$ with labels from a symbol set Σ, we theoretically establish the existence of ReLU-activated generative networks capable of producing all rooted trees over Σ within tree edit distance at most d from T. The key idea of our approach is to first construct a directed, rooted, ordered, and edge-labeled tree based on T. We then reduce the tree edit distance problem to a string edit distance problem by representing the tree as an Euler string [48], obtained through a depth-first search (DFS) traversal. The proposed networks are applied on trees with up to 21 nodes to generate similar trees, and are compared with the state-of-the-art graph generative models GraphRNN by You et al. [53] and GraphGDP by Huang et al. [57]. An implementation of the proposed networks is available at `https://github.com/MGANN-KU/TreeGen_ReLUNetworks`.

The paper is organized as follows: Preliminaries are discussed in Section 2. ReLU generative networks that can identify the indices and labels of the directed edges in the Euler string are discussed in Section 3. Existence of ReLU networks to generate all trees with tree edit distance at most d due to substitution operations is discussed in Section 4. Existence of ReLU networks to generate all trees with edit distance at most (resp., exactly) d due to deletion (resp., insertion) operations is discussed in Section 5. Generation of all trees with tree edit distance at most d due to simultaneous application of deletion, substitution and insertion operations by using a ReLU network is discussed in Section 6. Computational experiments are discussed in Section 7. A conclusion and future directions are given in Section 8. Proofs of some theorems, examples and explanations of the program codes with sample instances are given in Appendix 9.

2 Preliminaries

Edit distance between two vertex-labeled, rooted and ordered trees T and U is defined as the minimum number of operations needed to transform T into U. These operations are substitution, deletion, and insertion. Substitution involves simply changing the label of a node in T; deletion removes a non-root node a in T, reassigning its parent b as the new parent of all children of a; and insertion is the complement of the deletion operation, i.e., a node a is inserted as a child of a node b, and a is set as the new parent of an ordered subset of consecutive children of b. The order among the children is preserved during both the deletion and insertion operations, as illustrated in Fig. 1.

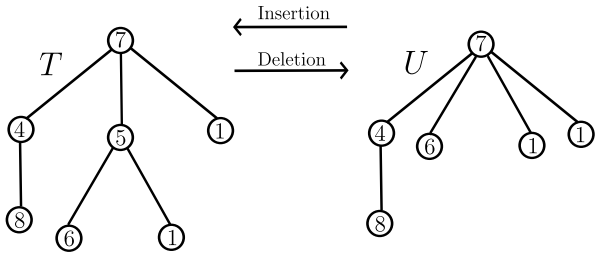


Figure 1: Tree deletion and insertion operations. In T, the node with label 7 is the parent of the node with label 5, which has been deleted. The node with label 7 becomes the new parent of the children with labels 1 and 6 of node 5 in U. Similarly, a node with label 5 is inserted as a child of the node with label 7 in U, and the nodes with labels 1 and 6 are set as children of node 5 in T. The order among the children 1 and 6 is preserved in the deletion and insertion operations.

Let T be a vertex-labeled, rooted and ordered tree with n edges ($n+1$ vertices) with vertex labels from the set $\Sigma = \{1, 2, \ldots, m\}$, and left-to-right ordering on the siblings of each vertex. We consider the depth-first search (DFS) index on the vertices of T starting from the root with index 0. For T, we define a directed edge-labeled, rooted and ordered tree with $n+1$ vertices and $2n$ edges as follows: replace the edge between any two adjacent vertices u and v with labels a and b, resp., where u is the parent of v, by two directed edges (u, v) and (v, u) with labels b and $b+m$, respectively. In this setting, we call (u, v) and (v, u), the inward edge and the outward edge, resp., of the vertex v. If i is the DFS index of v, then we call the inward and outward edges of v, the inward and outward edges of i. We call (v, u), the outward edge of the inward

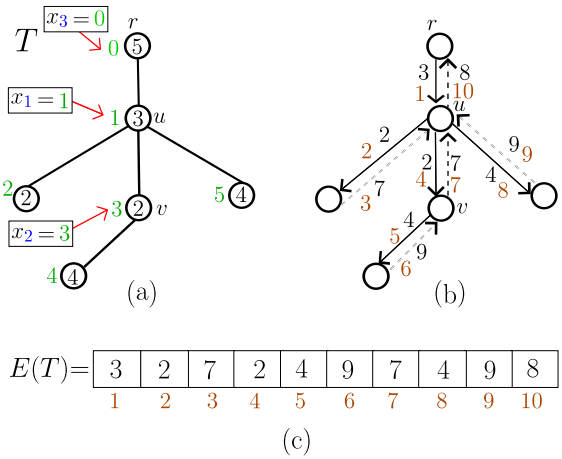


Figure 2: (a) A vertex-labeled, rooted and ordered tree T with six vertices, root r, label set $\Sigma = \{1, 2, 3, 4, 5\}$ and a random sequence $x_1, x_2, x_3 = 1, 3, 0$, where the labels are depicted inside the vertices, and the DFS indices are shown in green; (b) The directed tree corresponding to T given in (a). The inward edges and outward edges are depicted by solid and dashed directed lines, respectively. The labels and DFS indices of these edges are shown in black and brown color, respectively. The vertex u with label 3 is the parent of v with label 2. Corresponding to the edge uv in T, there is an inward edge (u, v) and an outward edge (v, u) with labels 2 and 7, respectively, in the directed tree. These edges (u, v) and (v, u) are the inward and outward, resp., edges of $x_2 = 3$ since the DFS index of v is 3 in T. Note that there is no edge that corresponds to $x_3 = 0$; and (c) The Euler string $E(T)$.

edge (u, v). The Euler string of T is defined to be the string obtained by listing the labels of the inward and outward edges of the directed tree corresponding to T in the DFS order on edges starting from index 1. We denote by $E(T) = t_1, t_2, \ldots, t_{2n}$, the Euler string of T with n edges. An example tree T, its directed tree, and Euler string are given in Figs. 2(a), (b), and (c), respectively. Henceforth, we will use the terms, edge and label interchangeably.

Observe that a tree can be completely determined by its Euler string, i.e., $E(T)$ is a canonical representation of T when the labels of roots of the underlying trees is fixed. Therefore any tree edit operations (substitution, deletion, and insertion) on a non-root vertex u of a given tree can be viewed as edit operations on the Euler string of the tree on the entries that correspond to the inward edge (u, v) and the outward edge (v, u) with some refinements to obtain the desired tree. Furthermore, a vertex in T can be specified by its DFS index. Therefore in the rest of the paper, we will use a random sequence $x_1, x_2, \ldots, x_d$, $d \geq 1$, with integers $x_j \in [0, n]$, unless stated otherwise, to specify the DFS indices of the vertices under consideration in a tree with $n + 1$ vertices. We ignore $x_j = 0$ and repeated entries in the case of substitution and deletion operations. An example random sequence is given in Fig. 2(a). In the rest of the discussion, we call a vertex-labeled or edge-labeled, rooted and ordered tree simply a tree. A list of symbols, variables, and their descriptions used throughout the discussion is provided in Table 3.

3 Identification of Edge Labels by ReLU Network

We focus on designing generative networks with ReLU as an activation function to generate all trees that are similar to a given tree. More precisely, we are interested in the following problem:

Input: A rooted, ordered, and vertex-labeled tree T with labels from an alphabet Σ, and a non-negative integer d.

Output: Construct generative networks with ReLU as an activation function that can generate all rooted, ordered, and vertex-labeled trees over Σ with tree edit distance at most d from T.

We target this problem by reducing the tree edit distance problem to the string edit distance problem by representing trees as their Euler strings as explained in Section 2. As a sub-task, the positions and

Table 1: List of symbols, variables, and their descriptions.

Symbols	Explanations
T	Vertex-labeled, rooted and ordered tree. See Fig. 1.
(u, v)	Inward edge of v, where u is the parent of v. See Fig. 2(b.)
(v, u)	Outward edge of v, where u is the parent of v. See Fig. 2(b).
$E(T)$	Euler string of tree T. See Fig. 2(c).
n	Number of edges in a tree.
d	Edit distance.
B	A sufficiently large number.
Σ	Set $\{1, 2, \ldots, m\}$ of labels.
$\text{in}_{\ell i}$	Number of inward edges between ℓ-th and i-th entries of an Euler string. See Fig. 4 .
$\text{out}_{\ell i}$	Number of outward edges between ℓ-th and i-th entries of an Euler string. See Fig. 4.
DFS index	Depth first search indexing of vertices. See Fig. 2(a)
TS_d	Generative ReLU network for tree edit distance d due to substitution. See Fig. 6.
TD_d	Generative ReLU network for tree edit distance d due to deletion. See Fig. 7.
TI_d	Generative ReLU network for tree edit distance d due to insertion. See Fig. 8.
TE_d	Generative ReLU network for tree edit distance d due to substitution, deletion or insertion. See Fig. 9.
$x_1, x_2, \ldots, x_{2d}$	A random input sequence for TS_d. $x_j, 1 \leq j \leq d$ specifies the DFS index of a vertex for substitution, and $x_{d+j}, 1 \leq j \leq d$ denote the value to be substituted. See Fig. 6.
$x_1, x_2, \ldots, x_d$	A random input sequence for TD_d. x_j specifies the DFS index of a vertex of a tree to be deleted. See Fig. 2(a).
$x_1, x_2, \ldots, x_{4d}$	A random input sequence for TI_d. $x_j, 1 \leq j \leq d$ specifies the DFS index of a vertex for insertion, and x_{d+j} and $x_{2d+j}, 1 \leq j \leq d$ specify bounds on the children of the vertex with index x_j, and $x_{3d+j}, 1 \leq j \leq d$ denotes the value to be inserted. See Fig. 8.
$x_1, x_2, \ldots, x_{7d}$	A random input sequence for TE_d. $x_j, 1 \leq j \leq d$ specifies the input for deletion, $x_{d+j}, 1 \leq j \leq 2d$ specifies the input for substitution, and $x_{3d+j}, 1 \leq j \leq 4d$ specifies the input for insertion. See Fig. 9.
	All local variables used in Lemma 1 are explained in Example 1 and Fig. 3.
	All local variables used in Proposition 1 are explained in Example 2 and Fig. 4.
	All local variables used in Lemma 2 are explained in Example 3 and Fig. 5.
	All local variables used in Theorem 1 are explained in Example 4 and Fig. 12.
	All local variables used in Theorem 2 are explained in Example 5 and Fig. 13.
	All local variables used in Theorem 3 are explained in Example 6 and Fig. 14.
	All local variables used in Theorem 4 are explained in Example 7.

labels of the under consideration inward and outward edges in the Euler string are required to perform the edit operations. However such positions and labels are not readily available. Therefore we first discuss the existence of ReLU networks to identify the positions and labels of the inward and outward edges in an Euler string corresponding to a given random sequence $x_1, x_2, \ldots, x_d$ in Lemma 1.

Lemma 1. *Let T be a tree with n edges, and $x = x_1, x_2, \ldots, x_d$ be a random DFS sequence of integers over the interval $[0, n]$. Then there exists a ReLU network with size $\mathcal{O}(dn)$ and constant depth that can identify the label of inward edge of the vertex with non-zero DFS index x_j in the Euler string $E(T)$.*

Proof. Let $E(T) = t_1, t_2, \ldots, t_{2n}$. The following system of equations can be used to obtain the labels of the required inward edges in $E(T)$ (see Example 1), where $i \in \{1, 2, \ldots, 2n\}$, $j \in \{1, 2, \ldots, d\}$ and C is a large

number.

$$p_i = \begin{cases} 1 & \text{if } t_i \leq m, \\ 0 & \text{otherwise,} \end{cases} \tag{1}$$

$$p'_i = \max(\sum_{k=1}^{i} p_k - C\delta(p_i, 0), 0), \tag{2}$$

$$p''_i = p'_i + \max(2n - C(1 - \delta(p'_i, 0)), 0), \tag{3}$$

$$q_{ji} = \delta(p''_i, x_j), \tag{4}$$

$$r'_{ji} = t_i \cdot q_{ji}. \tag{5}$$

Eq. (1) outputs a binary variable p_i which is 1 if and only if the i-th entry of $E(T)$ is the label of an inward edge. Eq. (2) identifies the DFS index of only inward edges in $E(T)$ (see Example 1). That is $p'_i = \ell \neq 0$ if and only if the i-th entry of $E(T)$ corresponds to the ℓ-th inward edge in the directed tree. Eq. (3) replaces $p'_i = 0$ by $2n$ to ignore the root case. The variable $q_{ji} = 1$ if and only if $p'_i = x_j$ in Eq. (4), and Eq. (5) is used to identify the labels of the desired inward edges. Note that all these equations involve the maximum function or δ function which can be simulated by the ReLU activation function based on Proposition 1 by Ghafoor and Akutsu [33]. Therefore we can construct an eight-layer neural network with ReLU as an activation function with size $\mathcal{O}(dn)$ and constant depth that can identify the labels of the inward edges of non-zero x_j. □

A demonstration of Lemma 1 is given in Example 1.

Example 1. *Consider the tree T shown in Fig. (2)(a) with $E(T) = 3, 2, 7, 2, 4, 9, 7, 4, 9, 8$, and the random sequence $x = 1, 3, 0$. We wish to identify the labels of the inward edges of $x_j \neq 0$ in $E(T)$. For $x_1 = 1$ and $x_2 = 3$, the labels of the inward edges are 3 and 2, resp., whereas $x_3 = 0$ does not correspond to any inward edge, and therefore it is ignored. The variables that are used in the process of obtaining the required labels by using Lemma 1 are discussed and illustrated in Table 2 and Fig. (3).*

Table 2: The variables, their meaning and example values used in Lemma 1.

Variable	Meaning	Value
x_j	Specify the inward edge of x_j of which the label is required.	$x = 1, 3, 0$ (Fig. 2(a))
p_i	A binary variable which is one if there is an inward edge at the i-th position of $E(T)$.	$p = [1, 1, 0, 1, 1, 0, 0, 1, 0, 0]$ (Fig. 3(a))
p'_i	The DFS index of the inward edge, among all the inward edges, that is at the i-th position of $E(T)$.	$p' = [1, 2, 0, 3, 4, 0, 0, 5, 0, 0]$ (Fig. 3(b))
p''_i	Replacing $p'_i = 0$ with $2n = 10$ in p' to ignore the root case.	$p'' =$ $[1, 2, 10, 3, 4, 10, 10, 5, 10, 10]$
q_{ji}	A binary variable which identifies the position i of the non-zero input x_j in $E(T)$, i.e., $q_{ji} = 1$ if and only if $p''_i = x_j$.	$q_{1,1} = q_{2,4} = 1$, other variables are zero (Fig. 3(c))
r'_{ji}	The required label of the inward edge of x_j.	$r'_{1,1} = 3, r'_{2,4} = 2$, other variables are zero (Fig. 3(c))

Proposition 1 gives a necessary and sufficient condition for the i-th entry to be the outward edge of an inward edge at ℓ-th position in an Euler string. The condition essentially depends on the number of inward edges and outward edges between the two given positions i and ℓ. Before going into the details, for any two positions i and ℓ, with $1 \leq \ell < i \leq 2n$, we denote by $\text{in}_{\ell \text{i}}$ (resp., $\text{out}_{\ell \text{i}}$), the number of inward edges

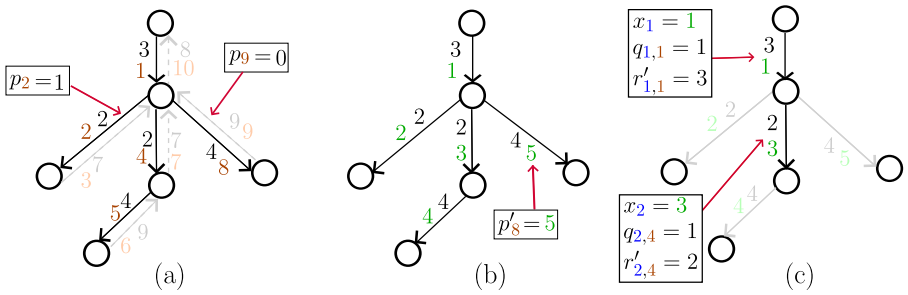


Figure 3: Illustrations of the variables used Eqs.(1)-(5) in Lemma 1: (a) The variable p_i which is 1 for the inward (black) edges and 0 for the outward (gray) edges in the directed tree corresponding to the tree T given in Fig. 2(a), e.g., $p_2 = 1$ (resp., $p_9 = 0$) as there is an inward edge (resp., outward edge) with the DFS index 2 (resp., 9); (b) The variable p'_i (green), e.g., $p'_8 = 5$ means that the inward edge of 5 has the DFS index 8 in (a); (c) For a fixed DFS index i, the variable $q_{ji} = 1$ for some j (black inward edge), and $q_{ji} = 0$ for all j (gray inward edges), e.g., for the DFS index $i = 4$, we have $j = 2$ such that $q_{2,4} = 1$ and thus the inward edge with the DFS index 4 is depicted in black, whereas for $i = 8$ there does not exist any j such that $q_{j8} = 1$, and so the inward edge with the DFS index 8 is depicted in gray. The dark edges are the desired inward edges of which labels are required. The labels of these edges are stored by the variable r'_{ji}, e.g., $r'_{2,4} = 2$ means that the desired inward edge specified by x_2 has the DFS index 4 and label 2.

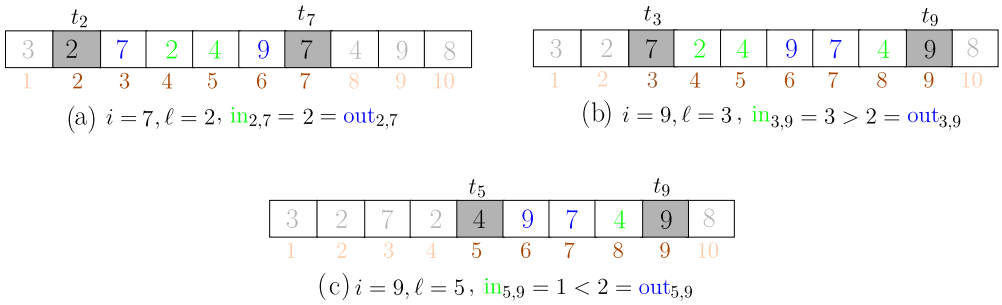


Figure 4: An illustration of the number of inward edges (green) (resp., outward edges (blue)) for the edges $t_i, i = 7, 9$ and $t_\ell, \ell = 2, 3, 5$ which are depicted in gray boxes.

(resp., outward edges) t_k, $\ell < k < i$. Consider the tree T given in Fig. 2(a) and its Euler string $E(T)$ given in Fig. 2(c), for $i = 7$, $\ell = 2$, $\text{in}_{2,7} = 2$ as there are two inward edges t_4, t_5 and $\text{out}_{2,7} = 2$ as there are two outward edges t_3, t_6 (see Fig. 4(a)). So in this case $\text{in}_{2,7} = \text{out}_{2,7}$. Similarly, for $i = 9, \ell = 3$, $\text{in}_{3,9} = 3 > 2 = \text{out}_{3,9}$ (see Fig. 4(b)) and for $i = 9, \ell = 5$, $\text{in}_{5,9} = 1 < 2 = \text{out}_{5,9}$ (see Fig. 4(c)).

Proposition 1. *Let $E(T) = t_1, t_2, \ldots, t_{2n}$ denote the Euler string of a tree T with n edges and $t_i \in \Sigma = \{1, 2, \ldots, m\}$. Then t_i, $1 \leq i \leq 2n$, is an outward edge of t_ℓ if and only if (i)-(iv) hold*

(i) $\ell \in [1, i-1]$,

(ii) $t_i = t_\ell + m$,

(iii) $\text{in}_{\ell i} = \text{out}_{\ell i}$, *and*

(iv) ℓ *is the largest index that satisfies (i)-(iii).*

Proof. We know that the Euler string follows the DFS. Therefore the DFS index of all descendant edges of a given inward edge appear between the inward edge and its outward edge from which the result follows. □

A demonstration of Proposition 1 is given in Example 2.

Example 2. *Consider the tree T given in Fig. 2(a) and its Euler string $E(T)$ shown in Fig. 2(c). We want to determine if the edges t_i, $i = 4, 7, 9$ in $E(T)$ are outward edges of some inward edges by using Proposition 1. We see that Proposition 1(ii) does not hold for $i = 4$ and any $\ell \in [1, 3]$, therefore t_4 is not an outward edge of any inward edge, which is consistent with the fact. For $i = 7$, Proposition 1(ii) and (iii) are satisfied for both $\ell = 2, 4$. For $\ell = 2$ (resp., $\ell = 4$), $\text{in}_{2,7} = 2 = \text{out}_{2,7}$ (resp., $\text{in}_{4,7} = 1 = \text{out}_{4,7}$) (see Fig. 4(a)). That is t_2 and t_4 are both candidate inward edges of t_7. By using Proposition 1(iv), t_7 is the outward edge of t_4. For $i = 9$, we see that t_5 and t_8 are the only edges that satisfy Proposition 1(i)-(ii), but t_5 does not satisfy the Proposition 1(iii). Therefore t_8 is the inward edge of t_9 implying that t_9 is an outward edge.*

The existence of a ReLU network to identify the positions and labels of outward edges in an Euler string based on Proposition 1 is discussed in Lemma 2.

Lemma 2. *Let T be a tree with n edges, and $x = x_1, x_2, \ldots, x_d$ be a random sequence of integers over the interval $[0, n]$. Then there exists a ReLU network with size $\mathcal{O}(dn^2)$ and constant depth that can identify the position and label of outward edge of each non-zero input x_j in the Euler string $E(T)$.*

Proof. Let $E(T) = t_1, t_2, \ldots, t_{2n}$. The proof completes by expressing the conditions of Proposition 1 in terms of ReLU activation function. Proposition 1 requires the labels of the inward edges of each $x_j \neq 0$ which can be computed by using Eqs.(1)-(5) of Lemma 1. Then the positions and labels of the required outward edges in $E(T)$ can be obtained by using the following system of equations (see Example 3), where $i, \ell \in \{1, 2, \ldots, 2n\}$, $j \in \{1, 2, \ldots, d\}$ and C is a large number.

$$r_i = t_i \cdot p_i, \tag{6}$$

$$s_i = t_i - r_i, \tag{7}$$

$$v_{\ell i} = \begin{cases} 0 & \text{if } i \leq \ell, \\ \max(\delta(s_i, r_\ell + m) - \\ \quad C(\sum_{k=\ell+1}^{i-1} H(s_k - 1) - \\ \quad \sum_{k=\ell+1}^{i-1} H(r_k - 1)), 0) & \text{otherwise,} \end{cases} \tag{8}$$

$$v'_{\ell i} = \delta(v_{\ell i}, 1), \tag{9}$$

$$w_{\ell i} = \max(v'_{\ell i} - \sum_{k=\ell+1}^{i-1} v'_{ki}, 0), \tag{10}$$

$$w'_{j\ell i} = \begin{cases} 0 & \text{if } i \leq \ell, \\ \max(\delta(s_i, r'_{j\ell} + m) - \sum_{k=1, k\neq\ell}^{2n} w_{ki}, 0) & \text{otherwise,} \end{cases} \tag{11}$$

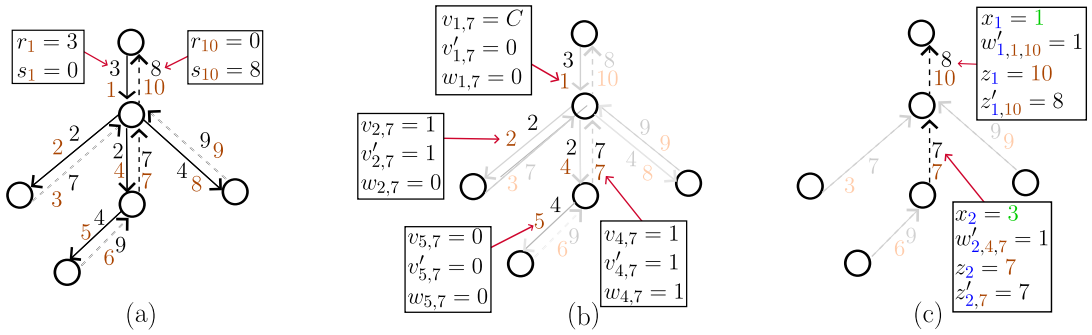


Figure 5: An illustration of the variables used in Eqs. (6)-(13) of Lemma 2 to identify the positions and labels of the desired outward edges.

$$z_j = i \cdot \sum_{i=1}^{2n} \sum_{\ell=1}^{2n} w'_{j\ell i}, \tag{12}$$

$$z'_{ji} = t_i \cdot \sum_{\ell=1}^{2n} w'_{j\ell i}. \tag{13}$$

The non-zero variable r_i (resp., s_i) in Eq. (6) (resp., Eq. (7)) stores the label of the inward edge (resp., outward edge) at the i-th position of $E(T)$. Eqs. (8) and (9) encode Proposition 1(ii) and (iii), and Eq. (10) encodes Proposition 1(iv). In Eq. (11), $w'_{j\ell i} = 1$ if and only if $s_i = r'_{j\ell} + m$ and $r'_{j\ell}$ is the largest index with this property, i.e., all conditions of Proposition 1 are satisfied. Eq. (12) (resp., Eq. (13)) determines the positions (resp., labels) in $E(T)$ of the desired outward edges. These equations involve the maximum function, δ function and Heaviside function which can be simulated by the ReLU activation function based on Proposition 1 by Ghafoor and Akutsu [33] and Theorem 1 by Kumano and Akutsu [23]. Therefore we can construct a twelve-layer neural network with ReLU as an activation function with size $\mathcal{O}(dn^2)$ and constant depth that can identify the positions and labels of the desired outward edges of non-zero x_j. □

A demonstration of Lemma 2 is given in Example 3.

Example 3. *Reconsider the tree T shown in Fig. (2)(a) with $E(T) = 3, 2, 7, 2, 4, 9, 7, 4, 9, 8$, and the random sequence $x = 1, 3, 0$. We wish to identify the positions and labels of the outward edges of $x_j \neq 0$ in $E(T)$. Note that the positions of outward edges are their DFS indices in the directed tree corresponding to T as demonstrated in Figs. (2)(b) and (c). For $x_1 = 1$ (resp., $x_2 = 3$), the positions and labels of the outward edges are 10 and 8 (resp., 7 and 7), resp., whereas $x_3 = 0$ does not correspond to any outward edge, and therefore it is ignored. We discuss the variables used in Lemma 2 to get the required positions and labels as follows. The variables $p_i, p'_i, p''_i, q_{ji}, r'_{ji}$ used in Eqs. (1)-(5) are discussed, in detail, in Example 1 and Fig. 3. We discuss the variables of Eqs. (6)-(13). An illustration of these variables is given in Fig. (5). In the rest of the discussion, more than one subscripts are separated by the commas to avoid confusion.*

- x_j *Specify the outward edge of x_j of which the position and label are required. In this case $x = 1, 3, 0$, which is illustrated in Fig. 2(a).*
- $p_i, p'_i, p''_i, q_{ji}, r'_{ji}$ *are explained in Example 1, Table 2 and Fig. 3.*
- r_i *The label of the inward edge of i, if it exists, e.g., in Fig. 5(a), $r_1 = 3$ (resp., $r_{10} = 0$) as there exists (resp., does not exist) an inward edge of 1 (resp., 10). The label of the inward edge of 1 is 3. The values of these variables are listed in $r = [3, 2, 0, 2, 4, 0, 0, 4, 0, 0]$.*
- s_i *The label of the outward edge of i, if it exists, e.g., in Fig. 5(a), $s_{10} = 8$ (resp., $s_1 = 0$) as there exists (resp., does not exist) an outward edge of 10 (resp., 1). The label of the outward edge of 10 is 8. Similarly, we get $s = [0, 0, 7, 0, 0, 9, 7, 0, 9, 8]$.*

$v_{\ell i}$ *A variable which can take a value from* $\{1, C, 0\}$*:* $v_{\ell i} = 1$ *if the conditions of Proposition 1(ii) and (iii) are satisfied, e.g.,* $i = 7, \ell = 2, 4$, $\text{in}_{\ell \text{i}} = \text{out}_{\ell \text{i}}$ *(see Example 2 and Fig. 4(a));* $v_{\ell i} = C$ *if Proposition 1(iii) is violated with the number of number of inward edges greater than the number of outward edges, i.e.,* $\text{in}_{\ell \text{i}} > \text{out}_{\ell \text{i}}$ *holds, e.g.,* $v_{3,9} = C$ *as* $\text{in}_{3,9} > \text{out}_{3,9}$ *(see Fig. 4(b));* $v_{\ell i} = 0$ *if Proposition 1(iii) is violated with the number of inward edges less than that of outward edges, i.e.,* $\text{in}_{\ell \text{i}} < \text{out}_{\ell \text{i}}$ *holds, e.g.,* $v_{5,9} = 0$ *as* $\text{in}_{5,9} < \text{out}_{5,9}$ *(see Fig. 4(c)). The values* $v_{2,7} = v_{4,7} = 1$, $v_{5,7} = 0$ *and* $v_{1,7} = C$ *are illustrated in Fig. 5(b). Thus we have,* $v_{1,10} = v_{2,3} = v_{2,7} = v_{4,7} = v_{5,6} = v_{8,9} = 1$, $v_{1,3} = v_{1,5} = v_{1,6} = v_{1,7} = v_{1,9} = v_{2,6} = v_{3,5} = v_{3,6} = v_{3,7} = v_{3,9} = v_{4,6} = v_{7,9} = C$*, and other variables are zero.*

$v'_{\ell i}$ *Replacing* C *with* 0 *in* $v_{\ell i}$*, e.g.,* $v_{2,6} = C$*, and therefore* $v'_{2,6} = 0$ *(see Fig. 5(b)). Thus* $v'_{1,10} = v'_{2,3} = v'_{2,7} = v'_{4,7} = v'_{5,6} = v'_{8,9} = 1$*, and other variables are zero.*

$w_{\ell i}$ *A binary variable which is one when* ℓ *is the largest number that satisfies Proposition 1(i)-(iii), i.e., Proposition 1(iv)is satisfied, e.g.,* $v'_{2,7} = v'_{4,7} = 1$ *are the only non-zero variables for* $i = 7$*. Since* 4 *is largest among such variables, we have* $w_{4,7} = 1$ *(see Fig. 5(b)). Similarly,* $w_{1,10} = w_{2,3} = w_{4,7} = w_{5,6} = w_{8,9} = 1$*, and other variables are zero.*

$w'_{j\ell i}$ *A binary variable to identify the desired outward edges corresponding to* $x_j \neq 0$*. More precisely* $w'_{j\ell i}$ *is one when* t_i *is the outward edge of the inward edge* t_ℓ *(Proposition 1 is satisfied), and* t_ℓ *is the inward edge of* x_j*, e.g.,* $w_{2,4,7} = 1$ *because* t_7 *is the outward edge of the inward edge* t_4 *which corresponds to* $x_2 = 3$ *(see Fig. 3(c)). In Fig. 5(c) the outward edges that have non-zero (resp., zero) value of* $w'_{j\ell i}$ *are depicted by dark (resp., gray) edges. Thus,* $w'_{1,1,10} = w'_{2,4,7} = 1$*, and other variables are zero.*

z_j *Position* i *of the outward edge of* $x_j \neq 0$*. In Fig. 5(c),* $z_1 = 10$ *and* $z_2 = 7$ *means that the position of the outward edges of* x_1 *(resp.,* x_2*) is* 10 *(resp.,* 7*). Thus* $z = [10, 7, 0]$*.*

z'_{ji} *Label of the outward edge of* $x_j \neq 0$ *which is at the position* i*. In this case* $z'_{1,10} = 8, z'_{2,7} = 7$*, and other variables are zero (see Fig. 5(c)).*

4 TS_d-generative ReLU

Let T be a tree with $n+1$ nodes and labels from $\Sigma = \{1, 2, \ldots, m\}$, Euler string $E(T) = t_1, t_2, \ldots, t_{2n}$, and a non-negative integer d. We define the *TS_d-generative ReLU* to be a ReLU neural network with $2d$ input nodes $x = x_1, x_2, \ldots, x_{2d}$ over $\{0, \ldots, n\}$, and $2n$ output nodes $u = u_1, u_2, \ldots, u_{2n}$ over Σ such that all Euler strings u of trees with the tree edit distance at most $2d$ from $E(T)$ can be obtained by the substitution of appropriate x_{j+d} and $x_{j+d} + m$ at the inward edge and outward edge of $x_j \neq 0$, respectively. In this context $x_1, \ldots, x_d$ represents the inward edges for the substitution operations, while $x_{d+1}, \ldots, x_{2d}$ denotes the values to be substituted during these operations. An illustration of such a network is given in Fig. 6, where $m = 5$, $d = 3$, $x = 1, 3, 0, 5, 1, 2$, means that the labels of the inward edges (resp., outward edges) of $x_1 = 1, x_2 = 3$ are substituted by $x_4 = 5, x_5 = 1$ (resp., $10, 6$). The entry $x_3 = 0$ corresponds to the root, and is ignored.

The existence of TS_d-generative ReLU network is discussed in Theorem 1.

Theorem 1. *For a rooted ordered tree* T *of size* $n+1$ *with a label set* $\Sigma = \{1, 2, \ldots, m\}$*, and a non-negative integer* d*, there exists a TS_d-generative ReLU network with size* $\mathcal{O}(dn^2)$ *and constant depth.*

A proof and an explanation of each variable of Theorem 1 is given in Example 4 in Appendix 9.

5 TD_d, TI_d-generative ReLU

Let T be a tree with $n+1$ nodes, labels from $\Sigma = \{1, 2, \ldots, m\}$ and Euler string $E(T) = t_1, t_2, \ldots, t_{2n}$, and a non-negative integer d. We define the *TD_d-generative ReLU* to be a ReLU neural network that can generate all Euler strings over Σ with the tree edit distance at most $2d$ from $E(T)$ by deleting from $E(T)$, the appropriate inward and outward edges of $x = x_1, x_2, \ldots, x_d$, over $\{0, \ldots, n\}$. To output a fixed number of nodes, we assume that $E(T)$ is padded with $2d$ Bs, $B \gg m$, so that the network can delete the inward and outward edges of $x_j \neq 0$ and delete two Bs corresponding to each $x_j = 0$. We call such a string a *padded Euler string*. In this setting, we fix $2n$ output nodes $y = y_1, \ldots, y_{2n}$ of the network, where $y_1, \ldots, y_{2n-2d'}$, $d' \leq d$ is the Euler string of some tree U with the tree edit distance $2d$ from $E(T)$ by deleting d' inward and outward edges of non-zero entries of x, and the remaining $2d'$ entries $y_{2n-2d'+1}, \ldots, y_{2n}$ are Bs. An illustration of a TD_d-generative ReLU is given in Fig. 7, where $m = 5$, $d = 3$, $x = 1, 3, 0$, and T is given in

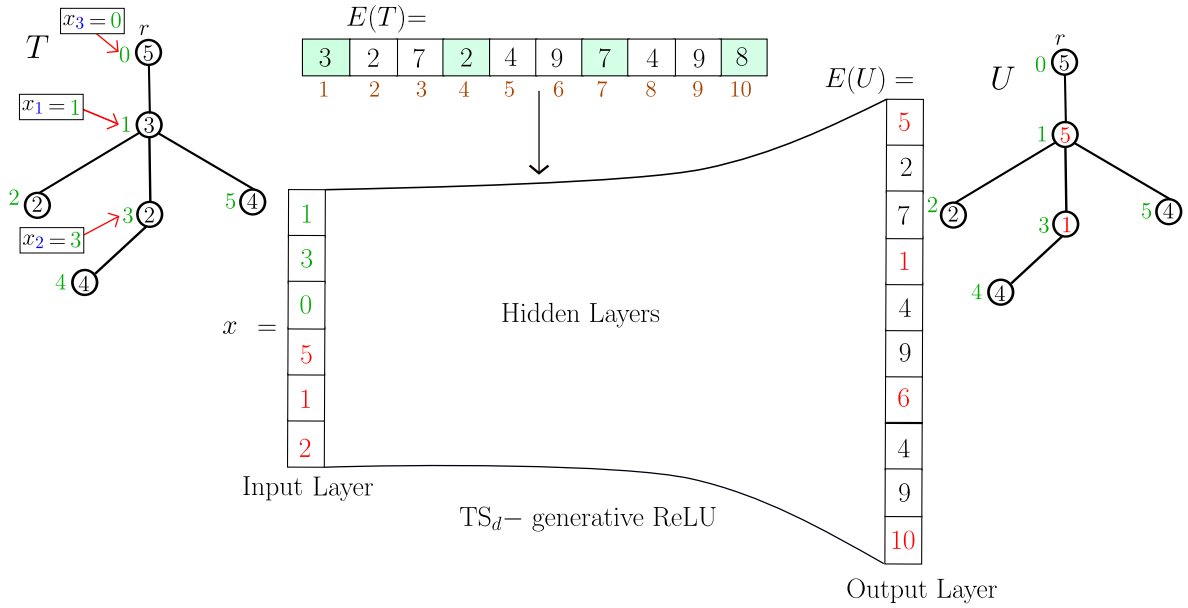


Figure 6: An illustration of a TS$_d$-generative ReLU with the input layer $x = 1, 3, 0, 5, 1, 2$, $E(T) = 3, 2, 7, 2, 4, 9, 7, 4, 9, 8$ and output layer $u = E(U) = 5, 2, 7, 1, 4, 9, 6, 4, 9, 10$. The substitution operations on $E(T)$ and $E(U)$ are depicted with green boxes and red values, respectively.

Fig. 2 with $E(T) = 3, 2, 7, 2, 4, 9, 7, 4, 9, 8$, and the padded $E(T)$ with six Bs. The network will delete the inward and outward edges of 1 and 3 as depicted in the figure, and delete two Bs corresponding to 0. The resultant string is $y = 2, 7, 4, 9, 4, 9, B, B, B, B$, and by removing Bs from y we get the desired Euler string $E(U) = 2, 7, 4, 9, 4, 9$ of the tree U as shown in the Fig. 7.

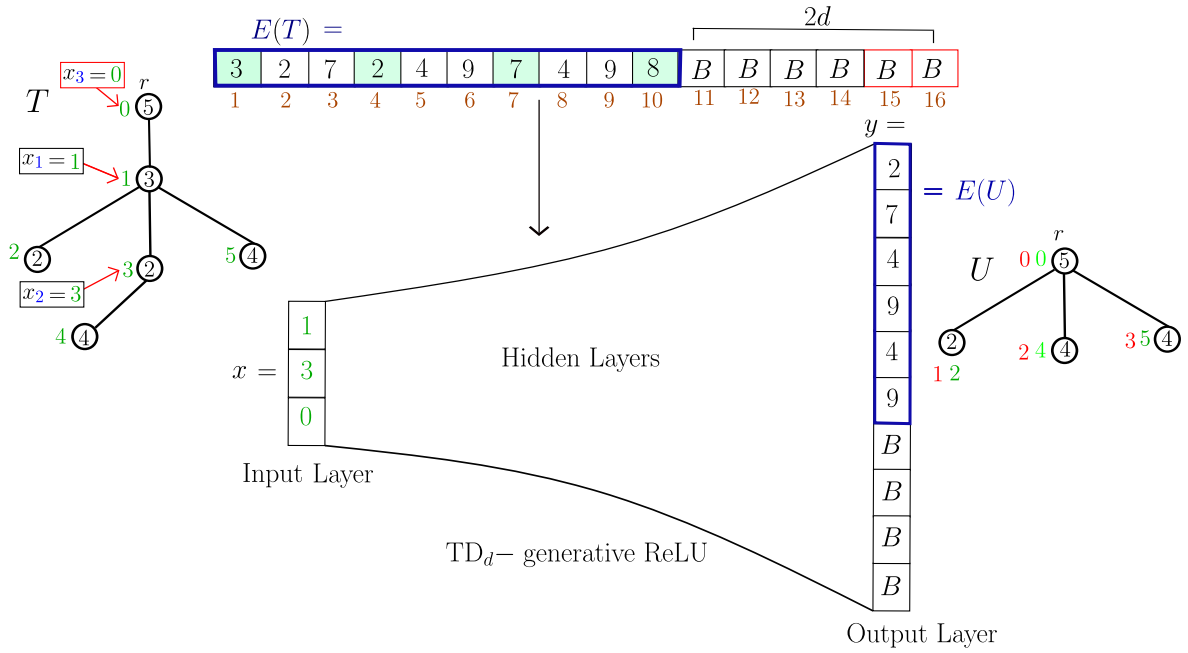


Figure 7: An illustration of a TD$_d$-generative ReLU with the input layer $x = 1, 3, 0$, padded Euler string $3, 2, 7, 2, 4, 9, 7, 4, 9, 8, B, B, B, B, B, B$ and the output layer $2, 7, 4, 9, 4, 9, B, B, B, B$. By deleting Bs we can get the resultant string $E(U) = 2, 7, 4, 9, 4, 9$ obtained by deleting the inward and outward edges of $x = 1, 3$.

The existence of TD_d-generative ReLU network is discussed in Theorem 2.

Theorem 2. *For a rooted ordered tree T of size $n+1$ with nodes from $\Sigma = \{1, 2, \ldots, m\}$, and a non-negative integer d, there exists a TD_d-generative ReLU network with size $\mathcal{O}(n^2)$ and constant depth.*

A proof and an explanation of each variable of Theorem 2 is given in Example 5 in Appendix 9.

We define TI_d-generative ReLU as follows. Let T be a tree with $n+1$ nodes and labels from $\Sigma = \{1, 2, \ldots, m\}$, Euler string $E(T)$, and a non-negative integer d. We define the *TI_d-generative ReLU* to be a ReLU neural network with $4d$ input nodes $x = x_1, x_2, \ldots, x_{4d}$ over $\{0, \ldots, n\}$, and $2n + 2d$ output nodes $u = u_1, u_2, \ldots, u_{2n+2d}$ over Σ such that all Euler strings u of trees with the tree edit distance exactly $2d$ from $E(T)$ can be obtained by the insertion of appropriate child nodes x'_j of x_j with inward edges and outward edges of labels x_{j+3d} and $x_{j+3d} + m$, resp., of x'_j, and setting the appropriate $(x_{j+d}, x_{j+d} + 1, \ldots, x_{j+2d})$-th children of the node x_j as the children of x'_j. In this study, we do not consider inserting new nodes as children of a newly inserted node. In this context, $x_1, \ldots, x_d$ represent the nodes for the insertion operations, $x_{1+d}, \ldots, x_{d+d}$ and $x_{1+2d}, \ldots, x_{d+2d}$ represent the lower and upper bounds to determine the subsequences of children that will be set as the children of the inserted nodes, and $x_{1+3d}, \ldots, x_{d+3d}$ represents the labels to be inserted. For convenience, we denote $x_1, \ldots, x_d$, $x_{1+d}, \ldots, x_{2d}$, $x_{1+2d}, \ldots, x_{3d}$, and $x_{1+3d}, \ldots, x_{4d}$ by x^1, x^2, x^3, and x^4, respectively. Note that some lower and upper bounds of children may not be valid due to the random nature of x, and thus need to be refined to perform appropriate insertion operations. Such invalid bounds, their refinements and appropriate insertions are listed in Table 3, where $D(x^1_j)$ denotes the number of children of the node x^1_j.

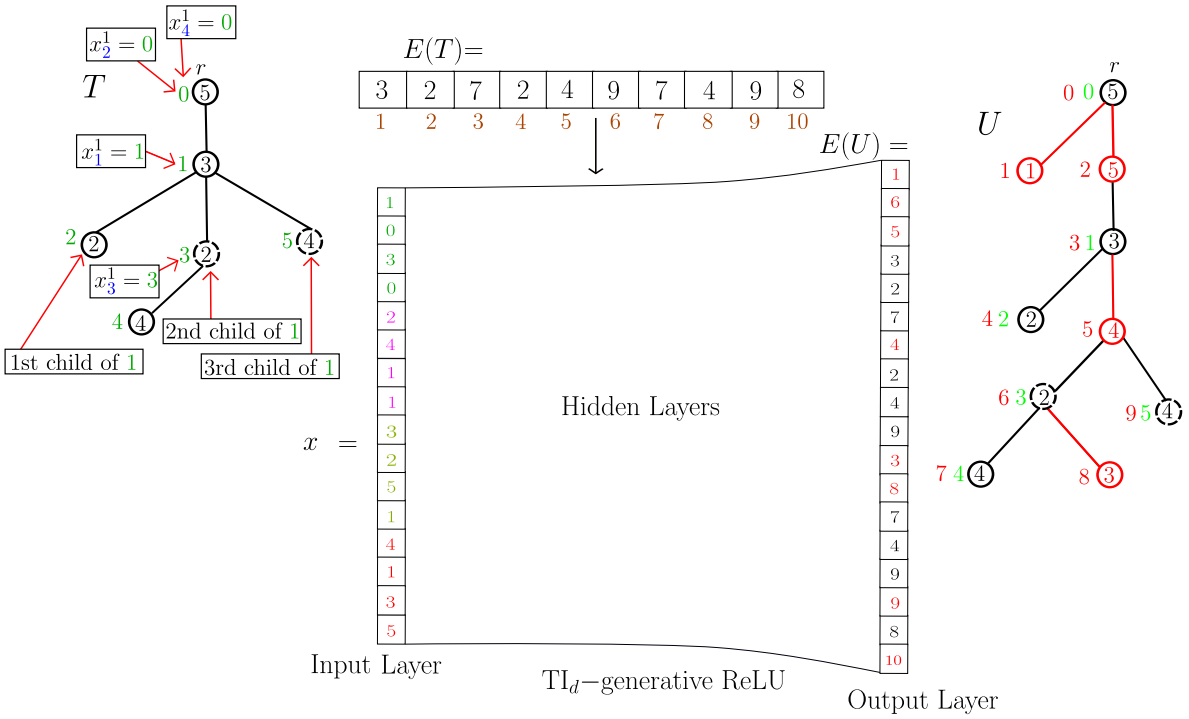


Figure 8: An illustration of a TI_d-generative ReLU with the input layer $x = 1, 0, 3, 0, 2, 4, 1, 1, 3, 2, 5, 1, 4, 1, 3, 5$, $E(T) = 3, 2, 7, 2, 4, 9, 7, 4, 9, 8$ and output layer $u = E(U) = 1, 6, 5, 3, 2, 7, 4, 2, 4, 9, 3, 8, 7, 4, 9, 9, 8, 10$. The insertions are depicted in red in U.

An illustration of a TI_d-generative ReLU is given in Fig. 8, where $m = 5$, $d = 4$ and x =1, 0, 3, 0, 2, 4, 1, 1, 3, 2, 5, 1, 4, 1, 3, 5. For convenience, we perform the insertion operations in the ascending order of the values x^1, i.e., in this case, the insertion is performed by considering the sequence $0, 0, 1, 3, 4, 1, 2, 1, 2, 1, 3, 5, 1, 5, 4, 3$. We discuss the insertion process as follows. For the node $x^1_2 = 0$, the bounds are $x^2_2 = 4$ and $x^3_2 = 2$, which are invalid as $D(x^1_2) = 1$. Therefore by applying refinements (i) and (ii), we set $x^2_2 := 0$ and $x^3_2 := 0$, and

Table 3: Invalid bounds of children, their refinements and appropriate insertions.

S. no.	Invalid bounds	Refinements	Insertions
(i)	$D(x_j^1) < x_j^2$	$x_j^2 := 0$	Insert a leaf before the first child of x_j^1
(ii)	$D(x_j^1) < x_j^3$	$x_j^3 := 0$	Insert a leaf after the x_j^2-th child of x_j^1
(iii)	$x_j^2 > x_j^3$	$x_j^3 := 0$	Insert a leaf after the x_j^2-th child of x_j^1
(iv)	$x_j^1 = x_k^1, j < k, x_j^3 > x_k^2$	$x_j^3 := 0$	Insert a leaf after the x_j^2-th child of x_j^1
(v)	$x_j^1 = x_k^1, j > k, x_j^2 = x_k^3$	$x_j^3 := 0$	Insert a leaf after the x_j^2-th child of x_j^1
(vi)	$x_j^1 = x_k^1, j < k, x_j^2 > x_k^2$	$x_j^2 := 0$	Insert a leaf before the first child of x_j^1
(vii)	$x_j^1 = x_k^1, x_j^2 = x_k^2, x_j^3 > x_j^2$	$x_j^2 := 0$	Insert two leaves before the first child of x_j^1
(viii)	$x_j^2 = 0$	$x_j^3 := 0$	Insert a leaf before the first child of x_j^1
(ix)	$x_j^2 \neq 0, x_j^3 = 0$	$x_j^2 := x_j^2 + 1$	Insert a leaf after the x_j^2-th child of x_j^1

thus insert a leaf with label 1 and DFS index 1 (see Fig. 8), inward and outward edges with labels 1 and 6 at 1st and 2nd positions of the resultant Euler string $E(U)$, respectively. The bounds for the node $x_4^1 = 0$ are valid, and hence a new node with label 5 is inserted with index 2, as shown in Fig. 8, and insert inward and outward edges 5 and 10 at 3rd and 18th positions of $E(U)$. For the node $x_1^1 = 1$, the given lower bound and upper bound for the children are $x_1^2 = 2$ and $x_1^3 = 3$, which are valid as the number $D(x_1^1)$ of children of x_1^1 are 3 as depicted in Fig. 8. The 2nd and 3rd children of x_1^1 have the DFS indices 3 and 5, respectively. Thus a new node ${x'}_1^1$ with label 4 and index 5 is inserted by setting the 2nd and 3rd children of x_1^1 as the children of ${x'}_1^1$. The revised indices of the children are 6 and 9, resp., as shown in Fig. 8. Inward and outward edges with labels 4 and 9 are inserted at 7th and 16th position of $E(U)$, respectively. For the node $x_3^1 = 3$, the bounds are $x_3^2 = 1$ and $x_3^3 = 5$, where the upper bound is invalid as $D(x_3^1) = 1$. By applying the refinement (iii), we set $x_3^3 := 0$, and by (ix) we have $x_3^2 := 2$, therefore insert a leaf after the first child of x_3^1 with DFS index 8, and insert inward and outward edges with labels 3 and 8 at the 11th and 12th positions of $E(U)$, respectively. The resultant tree U has the Euler string $E(U) = 1, 6, 5, 3, 2, 7, 4, 2, 4, 9, 3, 8, 7, 4, 9, 9, 8, 10$.

The existence of TI_d-generative ReLU network is discussed in Theorem 3.

Theorem 3. *For a rooted ordered tree T of size $n+1$ with nodes from $\Sigma = \{1, 2, \ldots, m\}$, and a non-negative integer d, there exists a TI_d-generative ReLU network with size $\mathcal{O}(n^3)$ and constant depth.*

A proof of Theorem 3 and an explanation of each variable used in it are given in Example 6 in Appendix 9.

6 TE_d-generative ReLU

Let T be a tree with $n+1$ nodes and labels from $\Sigma = \{1, 2, \ldots, m\}$, Euler string $E(T)$, and a non-negative integer d. We define the *TE_d-generative ReLU* to be a ReLU neural network such that each Euler string over Σ with edit distance at most $2d$ from $E(T)$ due to deletion, substitution and insertion operations can be obtained by appropriately choosing an input $x = x_1, x_2, \ldots, x_{7d}$ of $7d$ nodes with $x_j \in [0, 1)$, where x_j is of the form $i \cdot \Delta$, i is an integer and Δ is a small constant. The input $x_1, x_2, \ldots, x_d$ (resp., $x_{d+1}, x_{d+2}, \ldots, x_{3d}$ and $x_{3d+1}, x_{3d+2}, \ldots, x_{7d}$) represents the nodes for deletion (resp., substitution and insertion) operations. As a preprocessing step, the random inputs x_j for $1 \leq j \leq 2d$ and $3d+1 \leq j \leq 6d$ (resp., $2d+1 \leq j \leq 3d$ and $6d+1 \leq j \leq 7d$) are converted into integers $i \in \{0, \ldots, n\}$ (resp., $\ell \in \Sigma$) if $x_j \in ((i-1)/n, i/n]$ (resp., $x_\ell \in [(\ell-1)/m, \ell/m]$ for $\ell = 1$, $((\ell-1)/m, \ell/m]$ otherwise). For example, when $n = 5$ and $m = 10$, the conversion table is given in Table 7 in Appendix 9. To output a fixed number of nodes, we assume that $E(T)$ is padded with $2d$ Bs, where $B \gg \max(m, n)$. The network outputs $2n + 2d$ nodes $y = y_1, \ldots, y_{2n+2d}$ from which the desired string $E(U)$ can be obtained by trimming all Bs from the start and end. More precisely, if d_1 (resp., d_2) deletion (resp., insertion) operations are performed, then $2d - 2d_1$ (resp., $2d_2$) number of Bs will be trimmed from the end (resp., start) of the output y as shown in Fig. 9 in Appendix 9.

The existence of TE_d-generative ReLU network is discussed in Theorem 4.

Theorem 4. *For a rooted ordered tree T of size $n+1$ with nodes from $\Sigma = \{1, 2, \ldots, m\}$, and a non-negative integer d, there exists a TE_d-generative ReLU network with size $\mathcal{O}(n^3)$ and constant depth.*

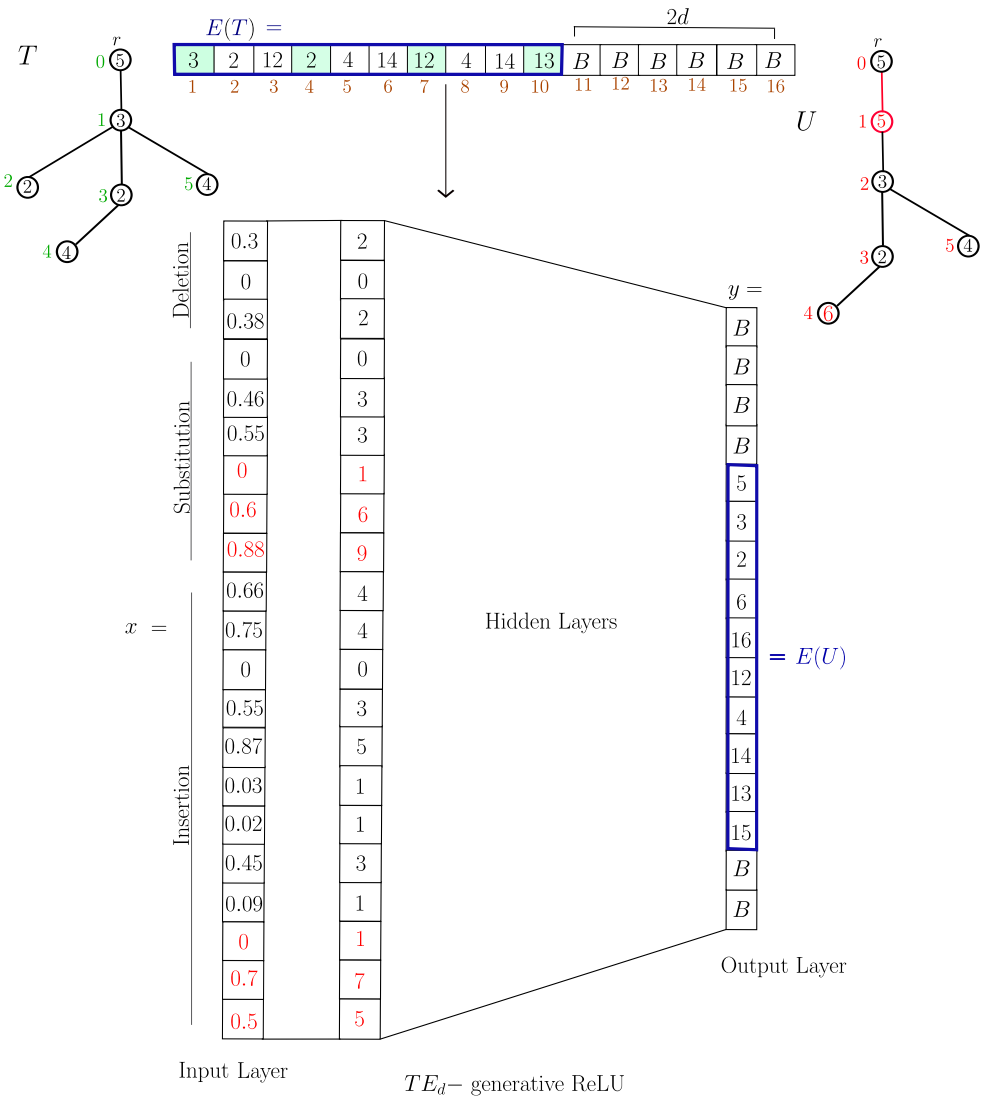


Figure 9: An illustration of a TE_d-generative ReLU for $d = 3$, with the input layer $x = 0.3, 0, 0.38, 0, 0.46, 0.55, 0, 0.6, 0.88, 0.66, 0.75, 0, 0.55, 0.87, 0.03, 0.02, 0.45, 0.09, 0, 0.7, 0.5$, padded Euler string $3, 2, 12, 2, 4, 14, 12, 4, 14, 13, B, B, B, B, B, B$ and the output layer $B, B, B, B, 5, 3, 2, 6, 16, 12, 4, 14, 13, 15, B, B, B, B$ with $d_1 = d_2 = 1$. By trimming Bs, we can get the resultant string $E(U) = 5, 3, 2, 6, 16, 12, 4, 14, 13, 15$ obtained by deleting, substituting and inserting the indicated nodes.

A proof of Theorem 4, and an explanation of each variable used in Theorem 4 is given in Example 7 in Appendix 9.

7 Computational Experiments

We implemented the proposed networks on a machine with an AMD Ryzen 7 4800H, Radeon Graphics processor (2.90 GHz), 16 GB of RAM, and Windows 11 Pro using Python `version 3.11.6`. For each

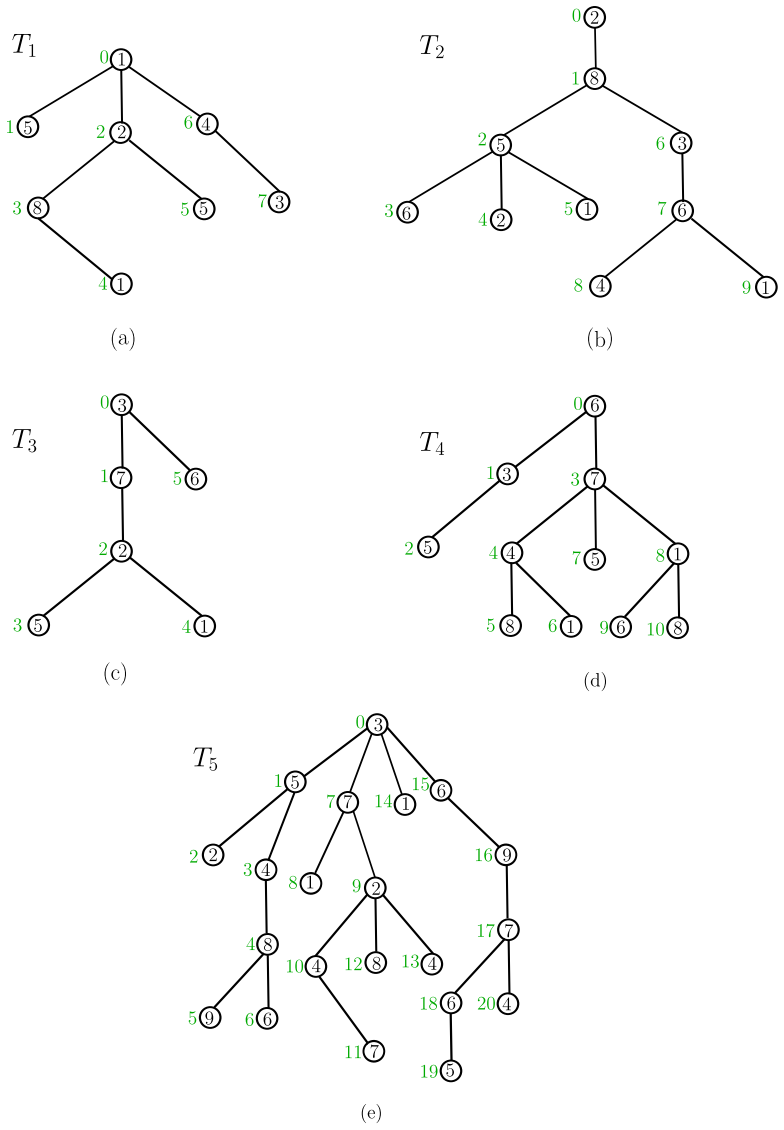
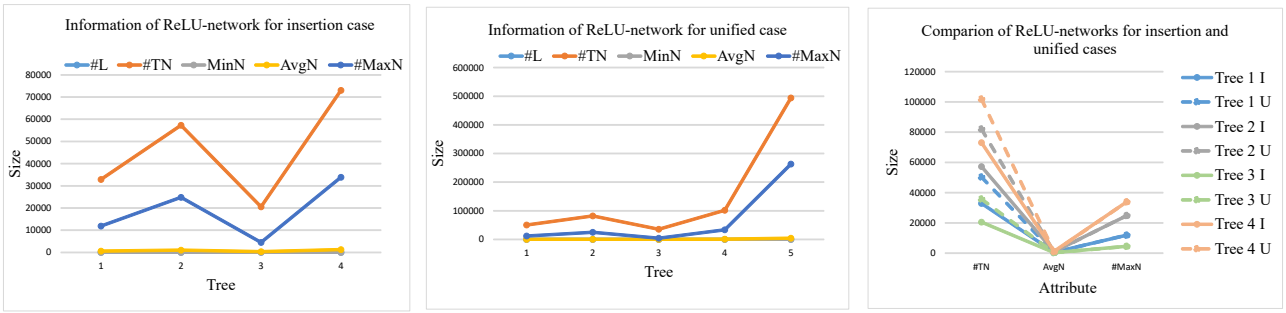


Figure 10: Trees used in the computational experiments of TI_d-generative ReLU and TE_d-generative ReLU.

tree T_i, $i = 1, 2, 3, 4$ illustrated in Figs. 10(a)-(d) with $8, 10, 6, 11$ nodes, labels from $\Sigma = \{1, 2, \ldots, 10\}$, and $d = 2, 2, 3, 2$, resp., we generated all non-isomorphic trees U that have tree edit distance exactly d by using the proposed TI_d-generative ReLU networks. The newly inserted nodes in T_1 are 7, 7, in T_2 are 9, 9, in T_3 are 8, 8, 8, and in T_4 are 2, 2. The computational results such as the number of nodes in each layer of the constructed TI_d network along with the number of generated trees for each input tree by the network are provided in Table 4. A summary of these computational results is given below. For T_i, let $(\#\mathrm{L}, \#\mathrm{TN}, \mathrm{MinN}, \mathrm{AvgN}, \#\mathrm{MaxN})_{\mathrm{i}}^{\mathrm{I}}$ denote the sequence of number of hidden layers, total number of hidden nodes, minimum number of hidden nodes, average number of hidden nodes, and maximum number of hidden nodes, resp., in the TI_d-generative ReLU network for T_i. From these experiments, we have $(57, 32876, 6, 576.77, 11903)_1^{\mathrm{I}}$, $(57, 57324, 6, 1005.68, 24841)_2^{\mathrm{I}}$, $(57, 20556, 9, 360.63, 4500)_3^{\mathrm{I}}$, $(57, 73064, 6, 1281.82, 33860)_4^{\mathrm{I}}$. These values are illustrated in Fig. 11(a) for each tree T_i, $i = 1, 2, 3, 4$.



(a) (b) (c)

Figure 11: Information of the architectures of the proposed ReLU-networks for insertion and unified cases and their comparison: (a) Information of $(\#\text{L}, \#\text{TN}, \text{MinN}, \text{AvgN}, \#\text{MaxN})_{\text{i}}^{\text{I}}$ of the proposed ReLU-networks for insertion case for each tree T_i, $i = 1, 2, 3, 4$; (b) Information of $(\#\text{L}, \#\text{TN}, \text{MinN}, \text{AvgN}, \#\text{MaxN})_{\text{i}}^{\text{E}}$ of the proposed ReLU-networks for unified case for each tree T_i, $i = 1, 2, 3, 4, 5$; (c) A comparison of the information of $(\#\text{TN}, \text{AvgN}, \#\text{MaxN})_{\text{i}}$ of the proposed ReLU-networks for insertion and unified cases for each tree T_i, $i = 1, 2, 3, 4$.

Observe that the depth remains fixed across all trees, confirming Theorem 3. Moreover, the widest layers are significantly larger than the others; for example, in T_4 with 11 nodes, the widest layer contains 33860 nodes which is over 26.41 times the average layer size (1281.82). Additionally, the total number of nodes grows faster than the tree size, e.g., while 11 nodes are roughly twice 6, the corresponding total node count (73064) is 3.56 times that for 6 nodes (20556).

Similarly, TE_d-generative ReLU neural networks were constructed for the trees T_i, $i = 1, 2, 3, 4, 5$ given in Figs. 10(a)-(e), where T_5 has 21 nodes, labels from $\Sigma = \{1, 2, \ldots, 10\}$, and $d = 2$. The newly inserted (resp., substituted) nodes in T_1 are 0.7, 0.7 (resp., 0.55, 0.55), in T_2 are 0.9, 0.9 (resp., 0.7, 0.7), in T_3 are 0.8, 0.8, 0.8 (resp., 0.88, 0.88, 0.88), in T_4 are 0.2, 0.2 (resp., 0.9, 0.9) and in T_5 are 0.3, 0.3 (resp., 0.99, 0.99).

Table 4: Experimental results for TI_d-generative ReLU

Input trees	Size of each hidden layer of TI_d-generative ReLU	Number of generated trees
T_1, Fig. 10(a)	36, 106, 806, 666, 232, 902, 400, 204, 23, 848, 218, 428, 218, 1073, 443, 758, 11903, 458, 670, 377, 252, 524, 520, 528, 524, 520, 544, 528, 520, 540, 528, 518, 516, 528, 518, 518, 516, 640, 962, 482, 6, 26, 10, 22, 12, 6, 278, 23, 22, 352, 82, 90, 50, 26, 262, 78, 36	318
T_2, Fig. 10(b)	44, 134, 1322, 1070, 368, 1446, 656, 332, 27, 1376, 350, 692, 350, 1737, 711, 1186, 24841, 730, 1074, 583, 392, 812, 808, 816, 812, 808, 832, 816, 808, 828, 816, 806, 804, 816, 806, 806, 804, 960, 1522, 762, 6, 26, 10, 22, 12, 6, 342, 27, 26, 432, 102, 94, 54, 30, 330, 98, 44	518
T_3, Fig. 10(c)	32, 82, 422, 362, 132, 529, 212, 112, 23, 452, 122, 232, 122, 573, 243, 474, 4500, 320, 378, 258, 164, 453, 447, 465, 456, 447, 501, 465, 447, 489, 459, 441, 438, 456, 441, 441, 438, 576, 795, 399, 9, 51, 15, 45, 27, 9, 345, 23, 22, 436, 88, 172, 88, 28, 286, 82, 32	546
T_4, Fig. 10(d)	48, 148, 1628, 1308, 448, 1766, 808, 408, 29, 1688, 428, 848, 428, 2129, 869, 1436, 33860, 890, 1312, 704, 474, 980, 976, 984, 980, 976, 1000, 984, 976, 996, 984, 974, 972, 984, 974, 974, 972, 1144, 1850, 926, 6, 26, 10, 22, 12, 6, 374, 29, 28, 472, 112, 96, 56, 32, 364, 108, 48	660

The computational results of these experiments are given in Table 5 which are summarized below. For T_i, let $(\#\text{L}, \#\text{TN}, \text{MinN}, \text{AvgN}, \#\text{MaxN})_{\text{i}}^{\text{E}}$ denote the sequence of number of hidden layers, total number of hidden nodes, minimum number of hidden nodes, average number of hidden nodes, and maxi-

Table 5: Experimental results for TE_d-generative ReLU

Input trees	Size of each hidden layer of TE_d-generative ReLU	Number of generated trees
T_1, Fig. 10(a)	2016, 28, 120, 42, 34, 40, 42, 46, 30, 24, 22, 14, 66, 144, 1418, 482, 392, 1470, 696, 1650, 678, 354, 48, 30, 84, 30, 102, 30, 292, 82, 26, 70, 156, 54, 124, 908, 374, 262, 934, 458, 1816, 640, 444, 80, 164, 80, 276, 78, 36, 22, 50, 148, 50, 120, 890, 356, 218, 806, 414, 218, 37, 862, 232, 442, 232, 1087, 457, 772, 11917, 472, 684, 391, 266, 538, 534, 542, 538, 534, 558, 542, 534, 554, 542, 532, 542, 532, 532, 530, 654, 976, 496, 20, 40, 24, 36, 26, 20, 292, 37, 36, 366, 82, 90, 50, 26, 262, 78, 36	747
T_2, Fig. 10(b)	2240, 28, 120, 42, 34, 40, 42, 46, 30, 24, 22, 14, 74, 172, 2082, 674, 564, 2146, 1024, 2454, 1002, 518, 56, 34, 100, 34, 122, 34, 372, 102, 30, 82, 196, 66, 156, 1452, 550, 406, 1486, 730, 2980, 1036, 712, 100, 208, 100, 352, 98, 44, 26, 62, 188, 62, 152, 1430, 528, 350, 1322, 674, 350, 45, 1394, 368, 710, 368, 1755, 729, 1204, 24859, 748, 1092, 601, 410, 830, 826, 834, 830, 826, 850, 834, 826, 846, 834, 824, 834, 824, 824, 822, 978, 1540, 780, 24, 44, 28, 40, 30, 24, 360, 45, 44, 450, 102, 94, 54, 30, 330, 98, 44	1223
T_3, Fig. 10(c)	2688, 42, 204, 63, 51, 60, 63, 69, 45, 36, 33, 21, 89, 142, 1141, 405, 325, 1250, 562, 1314, 546, 290, 50, 34, 82, 34, 98, 34, 298, 88, 28, 84, 127, 48, 98, 498, 275, 165, 555, 265, 1355, 455, 355, 85, 175, 85, 295, 82, 32, 22, 42, 112, 42, 92, 482, 259, 122, 422, 222, 122, 33, 462, 132, 242, 132, 583, 253, 484, 4510, 330, 388, 268, 174, 463, 457, 475, 466, 457, 511, 475, 457, 499, 469, 451, 466, 451, 451, 448, 586, 805, 409, 19, 61, 25, 55, 37, 19, 355, 33, 32, 446, 88, 172, 88, 28, 286, 82, 32	2525
T_4, Fig. 10(d)	2352, 28, 120, 42, 34, 40, 42, 46, 30, 24, 22, 14, 78, 186, 2462, 782, 662, 2532, 1212, 2916, 1188, 612, 60, 36, 108, 36, 132, 36, 412, 112, 32, 88, 216, 72, 172, 1772, 650, 490, 1810, 890, 3670, 1270, 870, 110, 230, 110, 390, 108, 48, 28, 68, 208, 68, 168, 1748, 626, 428, 1628, 828, 428, 49, 1708, 448, 868, 448, 2149, 889, 1456, 33880, 910, 1332, 724, 494, 1000, 996, 1004, 1000, 996, 1020, 1004, 996, 1016, 1004, 994, 1004, 994, 994, 992, 1164, 1870, 946, 26, 46, 30, 42, 32, 26, 394, 49, 48, 492, 112, 96, 56, 32, 364, 108, 48	1550
T_5, Fig. 10(e)	3472, 28, 120, 42, 34, 40, 42, 46, 30, 24, 22, 14, 118, 326, 8022, 2302, 2082, 8152, 3972, 9736, 3928, 1992, 100, 56, 188, 56, 232, 56, 812, 212, 52, 148, 416, 132, 332, 6732, 2090, 1770, 6810, 3370, 14530, 4930, 3330, 210, 450, 210, 770, 208, 88, 48, 128, 408, 128, 328, 6688, 2046, 1648, 6448, 3248, 1648, 89, 6608, 1688, 3328, 1688, 8289, 3369, 5296, 263350, 3410, 5052, 2614, 1774, 3580, 3576, 3584, 3580, 3576, 3600, 3584, 3576, 3596, 3584, 3574, 3584, 3574, 3574, 3572, 3904, 6930, 3486, 46, 66, 50, 62, 52, 46, 734, 89, 88, 912, 212, 116, 76, 52, 704, 208, 88	6309

mum number of hidden nodes, resp., in the TE_d-generative ReLU network for T_i. For the computational results listed in Table 5 we have $(108, 50360, 14, 466.30, 11917)_1^{\mathrm{E}}$, $(108, 82060, 14, 759.81, 24859)_2^{\mathrm{E}}$, $(108, 35803, 19, 331.51, 4510)_3^{\mathrm{E}}$, $(108, 101930, 14, 943.80, 33880)_4^{\mathrm{E}}$, $(108, 493790, 14, 4572.13, 263350)_5^{\mathrm{E}}$ for TE_d-generative ReLU networks. These values are illustrated in Fig. 11(b) for each tree T_i, $i = 1, 2, 3, 4, 5$.

Moreover a comparison of the TI_d and TE_d is given in Fig. 11(c). Observe that TE_d networks are deeper and, on average, narrower than TI_d networks for the same tree. Similarly, TE_d networks exhibit larger minimum widths compared to TI_d networks, while the maximum layer widths are nearly identical in both cases. These findings indicate that the insertion operation contributes the most to the overall size of the unified networks, which are composed of substitution, deletion, and insertion components. This observation also supports Theorems 3 and 4.

For each tree T_i, the inputs x and the corresponding Euler strings $E(U)$ generated by the TI_d and TE_d networks are listed in the supplementary material S1 which is available on `https://github.com/MGANN-KU/TreeGen_ReLUNetworks`.

Additionally, we conducted experiments to generate trees with a given edit distance by using state-of-the-art graph generative models called GraphRNN by You et al. [53] and GraphGDP by Huang et al. [57] for comparison. For this purpose, we randomly generated datasets of sizes 100, 150, 150, 200, and 800 of trees whose distances from T_1, T_2, T_3, T_4 and T_5 are $2, 2, 3, 2$ and 2, respectively. For simplicity, the labels of the underlying trees were ignored. We trained GraphRNN (dependent Bernoulli variant) (resp., GraphGDP) on each of these datasets by using a 4-layer RNN (resp., 4-layer GNN) with hidden neuron size 128 (resp., 128). Training ran for around 3000 epochs with batch size 32, learning rate 0.003 for GraphRNN

and 0.00002 for GraphGDP by using 80% of the input dataset, whereas 20% dataset was used for testing, and default parameters were retained for other settings. As a result, for each tree, 1024 (resp., 512) samples were generated by using each GraphRNN (resp., GraphGDP). Computational results are given in Table 6.

The results demonstrate notable variability in the performance of both models across different tree structures. It is important to note that both GraphRNN and GraphGDP were expected to generate tree structures, yet neither model guarantees that all generated samples are valid trees. Additionally, among the generated trees, only those that satisfy the specified edit distance constraint are considered valid in this evaluation.

GraphRNN consistently generated a high percentage of trees (ranging from 86.3% to 96.6%), yet only a small fraction of these were valid with respect to the target distance constraint. For instance, for trees T_1, T_2, and T_4 with distance $d = 2$, the proportion of valid trees remained low 6.4%, 6.9%, and 2.0%, respectively. In the case of T_5, GraphRNN failed to produce any valid tree, despite generating over 92% of trees. The only tree where GraphRNN achieved relatively high success was T_3 (with $d = 3$), where 35.8% of the generated samples were valid.

In contrast, GraphGDP generated fewer trees overall, with percentages ranging between 3.7% and 48.6% across different trees. However, it was able to generate a higher proportion of valid trees among those it produced, especially in the case of T_3, where 47.8% of its outputs were valid. For the other trees, especially those with $d = 2$, the valid tree percentages were noticeably lower: 9.2% for T_1, 3.9% for T_2, and 0.0% for both T_4 and T_5.

Table 6: Percentage of trees and valid trees generated by GraphRNN [53] and Huang et al. [57] with a given distance d

Tree	$n+1$	d	GraphRNN [53]		GraphGDP [57]	
			Trees	Valid trees	Trees	Valid trees
T_1	8	2	86.3%	6.4%	48.6%	9.2%
T_2	10	2	96.6%	6.9%	35.0%	3.9%
T_3	6	3	92.0%	35.8%	47.9%	47.8%
T_4	11	2	94.1%	2.0%	32.8%	0.0%
T_5	21	2	92.1%	0.0%	3.7%	0.0%

These findings suggest that GraphRNN and GraphGDP struggle to enforce the tree edit distance constraint, particularly as tree size and complexity increase, and may generate samples that are not even trees.

The datasets and the graphs generated by GraphRNN and GraphGDP are available in the supplementary materials S2 and S3, resp., which are available on `https://github.com/MGANN-KU/TreeGen_ReLUNetworks`

8 Conclusion

We study the existence of ReLU-based generative networks for producing trees similar to a given tree with respect to the tree edit distance. Our approach transforms a rooted, ordered, and vertex-labeled tree into a rooted, ordered, and edge-labeled directed tree. This directed tree is then encoded as an Euler string, which serves as both the input and output of the ReLU generative networks. First, we proved that there exists a ReLU network of size $\mathcal{O}(dn)$ and constant depth that can identify the labels of d inward edges in the Euler string. Furthermore, we showed that the outward edges corresponding to these d inward edges can be identified using a ReLU network of size $\mathcal{O}(dn^2)$ and constant depth. Building on these results, we demonstrated that all similar trees generated through substitution (resp., deletion and insertion) operations can be constructed by ReLU networks of size $\mathcal{O}(dn^2)$ (resp., $\mathcal{O}(n^2)$ and $\mathcal{O}(n^3)$), all with constant depth. Finally, we proved that there exists a ReLU network of size $\mathcal{O}(n^3)$ and constant depth that can generate all trees with a distance at most d to the original tree under combined substitution, deletion, and insertion operations. These findings provide a theoretical foundation towards construction of compact generative models and open new directions for efficient tree-structured data generation.

In this study, we do not consider scenarios where a newly inserted node becomes the parent of subsequent inserted nodes. This design choice simplifies the construction and supports tractable enumeration, but it limits the completeness of the editing model by excluding certain nested insertions. Addressing this limitation would extend the expressiveness of the framework and is a promising direction for future work. comparison with the state-of-the-art graph generative models GraphRNN by You et al. [53] and GraphGDP by Huang et al. [57] revealed that these models struggled to generate trees at a specified edit distance, particularly as tree size and structural complexity increased. For instance, GraphRNN and GraphGDP could not generate a single valid tree with 21 vertices and edit distance 2. While this experiment serves as an initial benchmark, it also underscores the need for further comparative evaluations with other models, such as TreeGAN and diffusion-based tree generators, to more clearly position the strengths of our proposed ReLU-based construction. On the other side, our construction demonstrates that our proposed ReLU network with constant depth and polynomial size can generate all trees within a given edit distance; however, the number of neurons in certain hidden layers increases rapidly with tree size. For instance, generating trees with 21 nodes can require layers containing over 263350 neurons. Although this wide structure ensures theoretical expressivity and completeness, it presents challenges for scalability, implementation, and deployment in resource-constrained environments. To address this limitation, future work may focus on strategies such as width pruning, parameter sharing, and compression to control and reduce the rapid growth in network width. An implementation of the proposed networks is available at `https://github.com/MGANN-KU/TreeGen_ReLUNetworks`.

Author contributions

Conceptualization, M.G. and T.A.; methodology, M.G. and T.A.; software, M.G. validation, T.A.; formal analysis, M.G. and T.A.; investigation, M.G. data curation, M.G.; writing—original draft preparation, M.G. and T.A.; writing—review and editing, M.G. and T.A..; supervision, T.A.; project administration, T.A. All authors have read and agreed to the published version of the manuscript.

Acknowledgments

The work of Tatsuya Akutsu was supported in part by Grants 22H00532 and 22K19830 from Japan Society for the Promotion of Science (JSPS), Japan. The authors would like to thank Dr. Naveed Ahmed Azam, Quaid-i-Azam University Pakistan, for the useful technical discussions.

Conflict of interest

All authors have no conflicts of interest in this paper.

References

[1] N. Killoran, L. J. Lee, A. Delong, D. Duvenaud, and B. J. Frey, Generating and designing DNA with deep generative models, *arXiv preprint arXiv:1712.06148*, (2017). `https://doi.org/10.48550/arXiv.1712.06148`

[2] T. Buschmann and L. V. Bystrykh, Levenshtein error-correcting barcodes for multiplexed DNA sequencing, *BMC Bioinformatics*, **14** (2013), 1–10. `http://www.biomedcentral.com/1471-2105/14/272`

[3] B. Al Kindhi, M. A. Hendrawan, D. Purwitasari, T. A. Sardjono, and M. H. Purnomo, Distance-based pattern matching of DNA sequences for evaluating primary mutation, in *Proceedings of the 2017 Second International Conference on Information Technology, Information Systems and Electrical Engineering*, pp. 310–314, 2017.

[4] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A. R. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury, Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups, *IEEE Signal Processing Magazine*, **29** (2012), 82–97. https://doi.org/10.1109/MSP.2012.2205597

[5] C. H. Wan, S. P. Chuang, and H. Y. Lee, Towards audio-to-scene image synthesis using generative adversarial network, in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, 2019.

[6] [arXiv:1609.03499] A. V. D. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, Wavenet: A generative model for raw audio, *arXiv preprint arXiv:1609.03499*, (2016). https://doi.org/10.48550/arXiv.1609.03499

[7] N. Aldausari, A. Sowmya, N. Marcus, and G. Mohammadi, Video generative adversarial networks: A review, *ACM Computing Surveys*, **55** (2022), 1–25. https://doi-org.kyoto-u.idm.oclc.org/10.1145/3487891

[8] J. Zhou and O. Troyanskaya, Deep supervised and convolutional generative stochastic network for protein secondary structure prediction, in *Proceedings of the International Conference on Machine Learning*, 2014.

[9] [10.1186/s13321-018-0287-6] J. Lim, S. Ryu, J. W. Kim, and W. Y. Kim, Molecular generative model based on conditional variational autoencoder for de novo molecular design, *Journal of Cheminformatics*, **10** (2018), 1–9. https://doi.org/10.1186/s13321-018-0286-7

[10] D. Schwalbe-Koda and R. Gómez-Bombarelli, Generative models for automatic chemical design, in *Machine Learning Meets Quantum Physics*, pp. 445–467, 2020.

[11] C. H. Gronbech, M. F. Vording, P. N. Timshel, C. K. Sonderby, T. H. Pers, and O. Winther, scVAE: Variational auto-encoders for single-cell gene expression data, *Bioinformatics*, **36** (2020), 4415–4422. https://doi.org/10.1093/bioinformatics/btaa293

[12] D. P. Kingma and M. Welling, An introduction to variational autoencoders, *Foundations and Trends in Machine Learning*, **12** (2019), 307–392. http://dx.doi.org/10.1561/2200000056

[13] Y. Bengio, L. Yao, G. Alain, and P. Vincent, Generalized denoising auto-encoders as generative models, in *Advances in Neural Information Processing Systems*, vol. 26, 2013.

[14] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, Generative adversarial networks, *Communications of the ACM*, **63** (2020), 139–144. https://doi-org.kyoto-u.idm.oclc.org/10.1145/3422622

[15] F. Gao, Y. Yang, J. Wang, J. Sun, E. Yang, and H. Zhou, A deep convolutional generative adversarial networks-based semi-supervised method for object recognition in synthetic aperture radar images, *Remote Sensing*, **10** (2018), 846. https://doi.org/10.3390/rs10060846

[16] G. Hinton and R. Salakhutdinov, Deep Boltzmann machines, in *Journal of Machine Learning Research Workshop and Conference Proceedings*, 2009.

[17] G. Alain, Y. Bengio, L. Yao, J. Yosinski, E. Thibodeau-Laufer, S. Zhang, and P. Vincent, GSNs: Generative stochastic networks, *Information and Inference: A Journal of the IMA*, **5** (2016), 210–249. https://doi.org/10.1093/imaiai/iaw003

[18] V. den Oord, Conditional image generation with PixelCNN decoders, in *Advances in Neural Information Processing Systems*, vol. 29, 2016.

[19] A. Van Den Oord, N. Kalchbrenner, and K. Kavukcuoglu, Pixel recurrent neural networks, in *Proceedings of the International Conference on Machine Learning*, 2016.

[20] K. Gregor, I. Danihelka, A. Graves, D. Rezende, and D. Wierstra, DRAW: A recurrent neural network for image generation, in *Proceedings of the International Conference on Machine Learning*, 2015.

[21] J. Ho, A. Jain, and P. Abbeel, Denoising diffusion probabilistic models, in *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS)*, vol. 33, pp. 6840–6851, 2020.

[22] L. Yang, Z. Zhang, Y. Song, S. Hong, R. Xu, Y. Zhao, W. Zhang, B. Cui, and M.-H. Yang, Diffusion models: A comprehensive survey of methods and applications, *ACM Computing Surveys*, **56** (2023), 1–39. https://doi-org.kyoto-u.idm.oclc.org/10.1145/3626235

[23] S. Kumano and T. Akutsu, Comparison of the representational power of random forests, binary decision diagrams, and neural networks, *Neural Computation*, **34** (2022), 1019–1044. https://doi.org/10.1162/neco_a_01486

[24] K. Hornik, M. Stinchcombe, and H. White, Multilayer feedforward networks are universal approximators, *Neural Networks*, **2** (1989), 359–366. https://doi.org/10.1016/0893-6080(89)90020-8

[25] G. F. Montúfar, R. Pascanu, K. Cho, and Y. Bengio, On the number of linear regions of deep neural networks, in *Proceedings of Advances in Neural Information Processing Systems*, vol. 27, pp. 1–9, 2014.

[26] M. Raghu, B. Poole, J. Kleinberg, S. Ganguli, and J. S. Dickstein, On the expressive power of deep neural networks, in *Proceedings of the International Conference on Machine Learning*, vol. 70, pp. 2847–2854, 2017.

[27] M. Telgarsky, Representation benefits of deep feedforward networks, *arXiv preprint arXiv:1509.08101*, (2015). https://doi.org/10.48550/arXiv.1509.08101

[28] L. Szymanski and B. McCane, Deep networks are effective encoders of periodicity, *IEEE Transactions on Neural Networks and Learning Systems*, **25** (2014), 1816–1827. https://doi.org/10.1109/TNNLS.2013.2296046

[29] V. Chatziafratis, S. G. Nagarajan, I. Panageas, and X. Wang, Depth-width trade-offs for ReLU networks via Sharkovsky's theorem, *arXiv preprint arXiv:1912.04378*, (2019). https://doi.org/10.48550/arXiv.1912.04378

[30] B. Hanin and D. Rolnick, Complexity of linear regions in deep networks, in *Proceedings of the International Conference on Machine Learning*, pp. 2596–2604, 2019.

[31] Y. Bengio, O. Delalleau, and C. Simard, Decision trees do not generalize to new variations, *Computational Intelligence*, **26** (2010), 449–467. https://doi-org.kyoto-u.idm.oclc.org/10.1111/j.1467-8640.2010.00366.x

[32] G. Biau, E. Scornet, and J. Welbl, Neural random forests, *Sankhyā: The Indian Journal of Statistics, Series A*, **81** (2019), 347–386. https://doi.org/10.48550/arXiv.1604.07143

[33] M. Ghafoor and T. Akutsu, On the Generative Power of Rectified Linear Unit Network for Generating Similar Strings, *IEEE Access*, **12** (2024), 52603–52622. https://doi.org/10.1109/ACCESS.2024.3387306

[34] S. M. Selkow, The tree-to-tree editing problem, *Information Processing Letters*, **6** (1977), 184–186. https://doi.org/10.1016/0020-0190(77)90064-3

[35] D. Gusfield, *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*, Cambridge University Press, 1997.

[36] B. A. Shapiro and K. Zhang, Comparing multiple RNA secondary structures using tree comparisons, *Bioinformatics*, **6** (1990), 309–318. https://doi.org/10.1093/bioinformatics/6.4.309

[37] M. Höchsmann, T. Töller, R. Giegerich, and S. Kurtz, Local similarity in RNA secondary structures, in *Proceedings of the 2003 IEEE Bioinformatics Conference on Computational Systems Bioinformatics*, 2003.

[38] M. S. Waterman, *Introduction to Computational Biology: Maps, Sequences and Genomes*, CRC Press, 1995.

[39] P. Buneman, M. Grohe, and C. Koch, Path queries on compressed XML, in *Proceedings of the Twenty-Ninth International Conference on Very Large Data Bases*, pp. 141–152, 2003.

[40] S. S. Chawathe, Comparing hierarchical data in external memory, in *Proceedings of the Twenty-Fifth International Conference on Very Large Data Bases*, pp. 90–101, 1999.

[41] [10.1145/1613676.1613680] P. Ferragina, F. Luccio, G. Manzini, and S. Muthukrishnan, Compressing and indexing labeled trees, with applications, *Journal of the ACM*, **57** (2009), 4:1–4:33. `http://doi.acm.org/10.1145/1613676.1613680`

[42] J. Bellando and R. Kothari, Region-based modeling and tree edit distance as a basis for gesture recognition, in *Proceedings of the Tenth International Conference on Image Analysis and Processing*, pp. 698–703, 1999.

[43] P. N. Klein, S. Tirthapura, D. Sharvit, and B. B. Kimia, A tree-edit-distance algorithm for comparing simple, closed shapes, in *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 696–704, 2000.

[44] P. N. Klein, T. B. Sebastian, and B. B. Kimia, Shape matching using edit-distance: An implementation, in *Proceedings of the Twelfth Annual Symposium on Discrete Algorithms*, pp. 781–790, 2001.

[45] T. B. Sebastian, P. N. Klein, and B. B. Kimia, Recognition of shapes by editing their shock graphs, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **26** (2004), 550–571. `https://doi.org/10.1109/TPAMI.2004.1273924`

[46] K.-C. Tai, The tree-to-tree correction problem, *Journal of the ACM*, **26** (1979), 422–433. `https://doi-org.kyoto-u.idm.oclc.org/10.1145/322139.322143`

[47] K. Zhang and D. Shasha, Simple fast algorithms for the editing distance between trees and related problems, *SIAM Journal on Computing*, **18** (1989), 1245–1262. `https://doi-org.kyoto-u.idm.oclc.org/10.1137/0218082`

[48] P. N. Klein, Computing the edit-distance between unrooted ordered trees, in *Proceedings of the Sixth Annual European Symposium on Algorithms*, pp. 91–102, 1998.

[49] E. D. Demaine, S. Mozes, B. Rossman, and O. Weimann, An optimal decomposition algorithm for tree edit distance, *ACM Transactions on Algorithms*, **6** (2010), 1–19.

[50] K. Bringmann, P. Gawrychowski, S. Mozes, and O. Weimann, Tree Edit Distance Cannot be Computed in Strongly Subcubic Time (unless APSP can), *ACM Transactions on Algorithms*, **16** (2020), 1–22. `https://doi.org/10.48550/arXiv.1703.08940`

[51] X. Mao, Breaking the cubic barrier for unweighted tree edit distance, in *Proceedings of the IEEE Annual Symposium on Foundations of Computer Science*, pp. 792–803, 2022. `https://doi.org/10.48550/arXiv.2106.02026`

[52] J. Nogler, A. Polak, B. Saha, V. V. Williams, Y. Xu, and C. Ye, Faster weighted and unweighted tree edit distance and all-pairs shortest paths equivalence, *arXiv preprint arXiv:2411.06502*, (2024). `https://doi.org/10.48550/arXiv.2411.06502`

[53] J. You, R. Ying, X. Ren, W. Hamilton, and J. Leskovec, GraphRNN: Generating Realistic Graphs with Deep Auto-regressive Models, in *Proceedings of the 35th International Conference on Machine Learning*, pp. 5708–5717, 2018.

[54] Z. Wang, J. Shi, N. Heess, A. Gretton, and M. K. Titsias, Learning-Order Autoregressive Models with Application to Molecular Graph Generation, *arXiv preprint arXiv:2503.05979*, (2025). https://doi.org/10.48550/arXiv.2503.05979

[55] D. Chen, M. Krimmel, and K. Borgwardt, Flatten Graphs as Sequences: Transformers are Scalable Graph Generators, *arXiv preprint arXiv:2502.02216*, (2025). https://doi.org/10.48550/arXiv.2502.02216

[56] X. Liu, X. Kong, L. Liu, and K. Chiang, TreeGAN: Syntax-Aware Sequence Generation with Generative Adversarial Networks, in *Proceedings of the IEEE International Conference on Data Mining (ICDM)*, pp. 1140–1145, 2018.

[57] H. Huang, L. Sun, B. Du, Y. Fu, and W. Lv, GraphGDP: Generative Diffusion Processes for Permutation Invariant Graph Generation, in *Proceedings of the 2022 IEEE International Conference on Data Mining (ICDM)*, pp. 201–210, 2022.

[58] X. Liu, Y. He, B. Chen, and M. Zhou, Advancing Graph Generation through Beta Diffusion, *arXiv preprint arXiv:2406.09357*, (2025). https://doi.org/10.48550/arXiv.2406.09357

[59] M. Madeira, C. Vignac, D. Thanou, and P. Frossard, Generative Modelling of Structurally Constrained Graphs, in *Proceedings of the 38th Conference on Neural Information Processing Systems*, pp. 137218–137262, 2024.

9 Appendix

Proofs and Examples

Proof of Theorem 1. Suppose $E(T) = t_1, t_2, \ldots, t_{2n}$ and $E(U) = u_1, u_2, \ldots, u_{2n}$ are two Euler strings over Σ of trees T and U, resp., such that $E(U)$ is obtained from $E(T)$ by substituting $x_{1+d}, x_{2+d}, \ldots, x_{2d}$ (resp., $x_{1+d}+m, x_{2+d}+m, \ldots, x_{2d}+m$) at the inward edges (resp., outward edges) of $x_1, x_2, \ldots, x_d$. We claim that the substitution operations on $E(T)$ to obtain $E(U)$ can be performed in the following three steps, where $i \in \{1, 2, \ldots, 2n\}$, $j \in \{1, 2, \ldots, d\}$ and C is a constant with $C \gg \max\{m, n\}$. These steps are demonstrated on an example tree in Example 4.

Step 1. Remove non-zero repetitions from $x_1, \ldots, x_d$ by setting repeated non-zero values to 0 to get x'.

$$x'_j = \max(x_j - C\sum_{k=1}^{j-1} \delta(x_j, x_k), 0). \tag{14}$$

Step 2. Get the labels of the inward and outward edges that will remain unchanged after the substitution operation by using P'_i. The non-zero value of P'_i is the unchanged label at the i-th entry in $E(T)$.

$$P_{ji} = r'_{ji} + z'_{ji}, \tag{15}$$

$$P'_i = t_i - \sum_{j=1}^{d} P_{ji}, \tag{16}$$

where r'_{ji} and z'_{ji} are the labels of the inward and outward edges corresponding to x' which can be obtained by Lemmas 1 and 2, respectively.
Step 3. Perform substitution at the inward and outward edges corresponding to x' by using Q_{ji} and Q'_{ji}, respectively. R_i stores all the substituted labels in the resultant Euler string.

$$Q_{ji} = \max(x_{j+d} - C\delta(r'_{ji}, 0), 0), \tag{17}$$

$$Q'_{ji} = \max(x_{j+d} + m - C\delta(z'_{ji}, 0), 0), \tag{18}$$

$$R_i = \sum_{j=1}^{d} Q_{ji} + Q'_{ji}. \tag{19}$$

Finally, combine the original and substituted entries to get the required Euler string $E(U)$ by

$$u_i = P'_i + R_i. \tag{20}$$

All the above equations involve the maximum function or δ function which can be simulated by ReLU activation function by using Proposition 1 by Ghafoor and Akutsu [33]. Therefore there exists a TS_d-generative ReLU network with size $\mathcal{O}(dn^2)$ and constant depth. □

Example 4. *Consider the tree T as shown in Fig. 2(a) with $E(T) = 3, 2, 7, 2, 4, 9, 7, 4,\ 9, 8$, $d = 3$, $m = 5$, and $x = 1, 3, 1, 5, 1, 2$. The resultant tree $E(U)$ obtained by applying the substitution operations on $E(T)$ due to the given x is shown in Fig. 6, where the repetition $x_3 = 1$ is ignored by setting it 0. We demonstrate the process of obtaining $E(U) = 5, 2, 7, 1,\ 4, 9, 6, 4, 9, 10$ by using Theorem 1 as follows.*

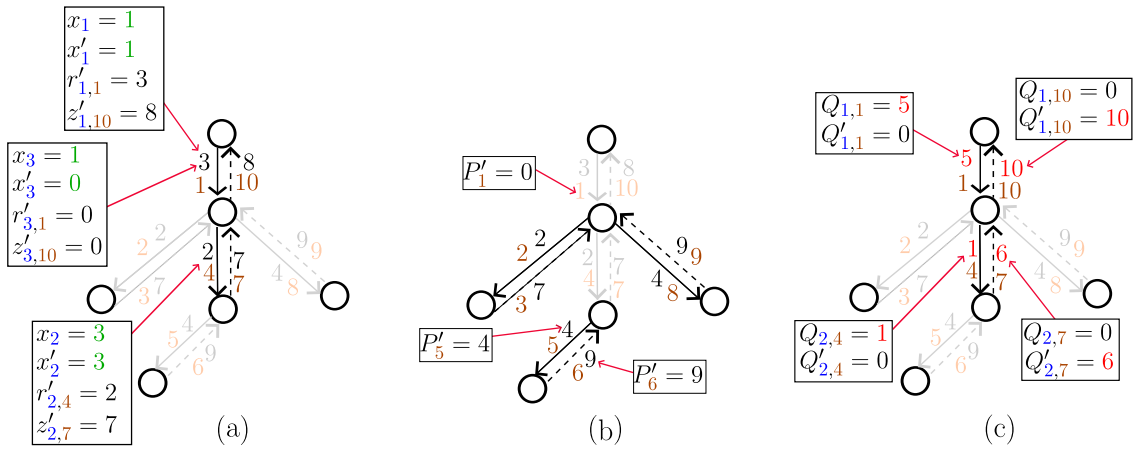


Figure 12: An illustration of the variables used in Theorem 1.

x_j	*Specify the inward edge and outward edge of $x_j \neq 0$ to substitute x_{j+d} and $x_{j+d} + m$. In this case $x = 1, 3, 1, 5, 1, 2$ as illustrated in Fig. 12(a), where the inward and outward edges that correspond to x are depicted in black.*
x'_j	*A variable that replaces repeated x_j with zero, e.g., $x_1 = x_3 = 1$, therefore $x'_3 = 0$. The values of the variables are $x' = 1, 3, 0$ as illustrated in Fig. 12(a).*
r'_{ji}, z'_{ji}	*The labels of inward and outward edges of x'_j, resp., as explained in Examples 1 and 3. The non-zero values of r'_{ij} and z'_{ji} are $r'_{1,1} = 3$, $r'_{2,4} = 2$, $z'_{1,10} = 8$ and $z'_{2,7} = 7$, and are depicted in Fig. 12(a).*
P_{ji}	*A variable that keeps the labels of the inward and outward edges simultaneously by taking the sum of r'_{ji} and z'_{ji}. Since $r'_{1,1} = 3$ and $z'_{1,1} = 0$ therefore, $P_{1,1} = r'_{1,1} + z'_{1,1} = 3$. Similarly, $P_{1,10} = r'_{1,10} + z'_{1,10} = 0 + 8 = 8$, $P_{2,4} = r'_{2,4} + z'_{2,4} = 2 + 0 = 2$, $P_{2,7} = r'_{2,7} + z'_{2,7} = 0 + 7 = 7$, and all other variables are zero.*
P'_i	*Stores the original entries of $E(T)$ where no substitution operation is performed by setting the i-th entry of $E(T)$ zero if P_{ji} is non-zero for some x'_j, i.e., the inward and outward edges that correspond to x' are set to zero in $E(T)$. For example, $P_{1,1} = 3 \neq 0$, therefore $P'_1 = 0$, whereas $P_{j,5} = 0$ for all j, therefore $P'_5 = 4$ which is the 5-th entry of $E(T)$. In this case $P' = [0, 2, 7, 0, 4, 9, 0, 4, 9, 0]$ as depicted in Fig. 12(b).*
Q_{ji}	*Performs substitution at the inward edges, i.e., $Q_{ji} = x_{j+d}$ if $r'_{ji} \neq 0$, e.g., $Q_{2,4} = 1$ as $r'_{2,4} = 2 \neq 0$, implying that the inward edge of x'_2 has index 4 in $E(T)$ and is substituted by $1 = x_{2+3}$. Similarly $Q_{1,1} = 5$, and all other variables are zero as depicted in Fig. 12(c).*
Q'_{ji}	*Performs substitution at outward edges, i.e., $Q'_{ji} = x_{j+d} + m$ if $z'_{ji} \neq 0$, e.g., $Q'_{2,7} = 6$ as $z'_{2,7} = 7 \neq 0$, implying that the outward edge of x'_2 has index 7 in $E(T)$ and is substituted by $6 = x_{2+3} + 5$. Similarly $Q'_{1,10} = 10$, and all other variables are zero as depicted in Fig. 12(c).*
R_i	*Stores substituted value at index i. The values of the variables in this case are $R = [5, 0, 0, 1, 0, 0, 6, 0, 0, 10]$.*
u_i	*The resultant string $E(U)$. The values of the variables are $u = [5, 2, 7, 1, 4, 9, 6, 4, 9, 10]$. The corresponding tree U is shown in Fig. 6.*

Proof of Theorem 2. Suppose $E(T) = t_1, t_2, \ldots, t_{2n}$ and $E(U) = u_1, u_2, \ldots, u_{2(n-d')}$, $d' \leq d$ are two Euler strings over Σ corresponding to the trees T and U, resp., such that $E(U)$ is obtained from $E(T)$ by deleting at most $2d$ edges of $x_1, x_2, \ldots, x_d$ from T. We claim that the $E(U)$ can be obtained by using the following system of equations, where $i, \ell \in \{1, 2, \ldots, 2n + 2d\}$, $j \in \{1, 2, \ldots, d\}$ unless stated otherwise, and B, C are large numbers such that $C \gg B \gg \max(m, n)$.

Step 1. Remove non-zero repetitions from x to get x' as explained in Theorem 1.
Step 2. Identify the positions and labels of the inward and outward edges to be deleted by using Lemmas 1 and 2 as follows.

$$q_i = \sum_{j=1}^{d} q_{ji}, \tag{21}$$

$$r'_i = t_i q_i, \tag{22}$$

$$w'_{\ell i} = \begin{cases} 0 & \text{if } i \leq \ell, \\ \max(\delta(s_i, r'_\ell + m) - \sum_{k=1, k\neq\ell}^{2n} w_{ki}, 0) & \text{otherwise,} \end{cases} \tag{23}$$

$$z'_i = t_i \cdot \sum_{\ell=1}^{2n} w'_{\ell i}, \tag{24}$$

$$P_i = r'_i + z'_i. \tag{25}$$

Step 3. Identify the labels to be retained to construct the resultant string after the deletion operations as follows.

$$Q_i = \delta(P_i, 0), \tag{26}$$

$$R_i = \max(B \sum_{k=1}^{i} Q_k - C\delta(Q_i, 0), 0), \tag{27}$$

$$R'^{j}_i = [iB \leq R_{i+j-1} \leq iB + 1] \cdot t_{i+j-1} \text{ for } i \in \{1, 2, \ldots, 2n\} \quad j \in \{1, 2, \ldots, 2d + 1\}, \tag{28}$$

Finally, get the required Euler string $E(U)$ from y_i by removing Bs.

$$y_i = \sum_{j=1}^{2d+1} R'^{j}_i \text{ for } i \in \{1, 2, \ldots, 2n\}. \tag{29}$$

All the above equations involve the maximum function, δ function, or threshold function, which can be simulated by ReLU activation function by using Proposition 1 by Ghafoor and Akutsu [33]. Therefore there exists a TD_d-generative ReLU network with size $\mathcal{O}(n^2)$ and constant depth. □

Example 5. *Reconsider the tree T given in Fig. 2 with $E(T) = 3, 2, 7, 2, 4, 9, 7, 4,\ 9, 8$, $d = 3$, $m = 5$, and $x = 1, 3, 1$. The resultant tree U obtained by applying the deletion operations on T due to the given x is shown in Fig. 7, where the repetition $x_3 = 1$ is ignored by setting it 0 and deleting two Bs from the padded Euler string $E(T)$. We demonstrate the process of obtaining $E(U) = 2, 7, 4, 9, 4, 9$ by using Eqs. (21)- (29) as follows. An illustration of the variables used in these equations is given in Fig. 13.*

- x_j *Specify the inward edge and outward edge of $x_j \neq 0$ to be deleted. In this case $x = 1, 3, 1$ as illustrated in Fig. 13(a), where the inward and outward edges that correspond to x are depicted in black.*
- x'_j *A variable that replaces repeated non-zero x_j with 0, e.g., $x_3 = 1$ is repeated, and therefore $x'_j = 0$. The values of the variables in this case are $x' = [1, 3, 0]$, and are depicted in Fig. 13(a).*

p_i, p'_i, p''_i, q_{ji} *and* $s_i, w_{\ell i}$ *are explained in Examples 1 and 3, respectively.*

q'_i *A binary variable which is one if the inward edges of* $x'_j \neq 0$ *is the* i*-th entry of* $E(T)$*. In other words, this variable identifies the positions of the inward edges to be deleted from* $E(T)$*. In this case* $q'_1 = q'_4 = 1$*, since* $q_{1,1} = q_{2,4} = 1$ *as shown in Fig. 13(a).*

r'_i *Stores the label of the inward edge of* $x'_j \neq 0$ *which is the* i*-th entry of* $E(T)$*. The non-zero values of this variable are* $r'_1 = 3$ *and* $r'_4 = 2$ *as shown in Fig. 13(a).*

$w'_{\ell i}$ *A binary variable to identify the outward edges of* $x_j \neq 0$*. More precisely,* $w'_{\ell i}$ *is one when* t_i *is the outward edge of the inward edge* t_ℓ*, and* t_ℓ *is the inward edge of* x_j*, e.g.,* $w'_{1,10} = w'_{4,7} = 1$ *because* t_{10} *and* t_7 *are the outward edges of the inward edges* t_1 *and* t_4*, resp., as depicted in Fig. 13(a). All other values are zero.*

z'_i *Stores the label of the outward edge of* $x'_j \neq 0$ *which is the* i*-th entry of* $E(T)$*. The non-zero values of this variable are* $z'_7 = 7$ *and* $z'_{10} = 8$ *as shown in Fig. 13(a).*

P_i *A variable that keeps the labels of both inward and outward edges to be deleted by taking the sum of* r'_i *and* z'_i*. In this case,* $P = [3, 0, 0, 2, 0, 0, 7, 0, 0, 8, 0, 0, 0, 0, 0, 0]$.

Q_i *A binary variable to identify which entries of the padded* $E(T)$ *should be retained to get the string of the resultant tree, e.g.,* $P_5 = 0$ *implies that the* 5*th entry of* $E(T)$ *should appear in the resultant tree, and so* $Q_5 = 1$ *as depicted in Fig. 13(b). Therefore* $Q = [0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1]$.

R_i *Assigns weights to the retained entries in the ascending order, e.g.,* $t_5 = 4$ *is the* 3*rd entry to be retained as* $Q_5 = 1$*, and therefore* $R_5 = 3B$*. In this case,* $R = [0, B, 2B, 0, 3B, 4B, 0, 5B, 6B, 0, 7B, 8B, 9B, 10B, 11B, 12B]$ *as depicted in Fig. 13(b).*

R'^j_i *Determines the label of the* i*-th entry of the resultant string obtained after the deletion operation. More precisely, the non-zero* R'^j_i *is equal to the label of the entry of* $E(T)$ *which has value* iB *in* R*, e.g., when* $i = 4$*,* $R_6 = 4B$ *and the* 6*th entry of* $E(T)$ *is* 9*, therefore* $R'^3_4 = 9$ *where* $i + j - 1 = 4 + 3 - 1 = 6$*, shows that the element of* 6*th position of* $E(T)$ *has a shift of* $j - 1 = 2$ *and becomes the* 4*th element of resultant string as depicted in Fig. 13(c). The non-zero values of* R'^j_i *are* $R'^2_1 = 2$*,* $R'^2_2 = 7$*,* $R'^3_3 = 4$*,* $R'^3_4 = 9$*,* $R'^4_5 = 4$*,* $R'^4_6 = 9$*,* $R'^5_7 = R'^5_8 = R'^6_9 = R'^6_{10} = B$*.*

y_i *Returns the non-zero entries of* R'^j_i *for a fixed* i *from which* $E(U)$ *can be obtained by removing* B*s. In this case* $y = [2, 7, 4, 9, 4, 9, B, B, B, B]$ *and so* $E(U) = 2, 7, 4, 9, 4, 9$ *as required.*

Proof of Theorem 3. Consider two trees T and U over Σ such that $E(U)$ is obtained from $E(T)$ by inserting exactly d inward and d outward edges based on an appropriate $x = x_1, \ldots, x_{4d}$. We claim that the insertion operations on $E(T)$ to obtain $E(U)$ can be performed in the following 10 steps, where $\ell \in \{0, 1, \ldots, 2n\}$, $i \in \{1, 2, \ldots, 2n\}$, $j \in \{1, 2, \ldots, d\}$, unless stated otherwise, and C is a large number.

Step 1. Determine the positions of inward and outward edges. The variable q_j determines the position of the inward edge of x^1_j by using Eq. (4), and b_ℓ determines the position of each outward edge by using Eq. (10), where b_0 corresponds to the root.

$$q_j = \sum_{i=1}^{2n} i \cdot q_{ji}, \tag{30}$$

$$b_0 = 2n + 1, b_\ell = \sum_{i=1}^{2n} i \cdot w_{\ell i}, \text{ for } \ell \in \{1, 2, \ldots, 2n\}. \tag{31}$$

Step 2. Determine the number of children of the node x^1_j, which is equal to the number of the descendant inward edges adjacent to the inward edge of x^1_j. The variable $A_{\ell i}$ is non-zero if and only if the i-th edge (inward or outward) is adjacent to the ℓ-th inward edge in $E(T)$, whereas a_ℓ is the number of adjacent inward

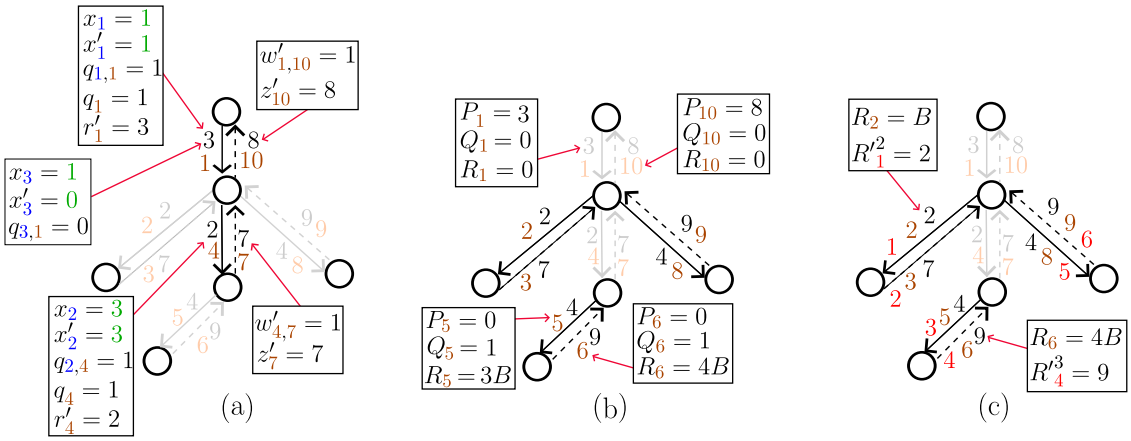


Figure 13: An illustration of the variables used in Theorem 2.

edges and D_j is the number of children of x_j^1.

$$A_{\ell i} = \max([\ell + 1 \le i \le b_\ell - 1] - \sum_{k=\ell+1}^{2n} [k + 1 \le i \le b_k - 1], 0), \tag{32}$$

$$a_\ell = \sum_{i=1}^{2n} A_{\ell i}/2, \tag{33}$$

$$D_j = \sum_{k=1}^{n} \sum_{\ell=0}^{2n} k \cdot (\delta(k, a_\ell) \wedge \delta(\ell, q_j)). \tag{34}$$

Step 3. Refine the invalid lower bound x^2 and the upper bound x^3 as follows, where the refinements (i)-(ix) are performed by Eqs. (35)-(43), respectively.

$$Q_j^1 = \max(x_j^2 - C(1 - H(D_j - x_j^2)), 0), \tag{35}$$

$$P_j^1 = \max(x_j^3 - C(1 - H(D_j - x_j^3)), 0), \tag{36}$$

$$P_j^2 = \max\left(P_j^1 - C \cdot H(Q_j^1 - P_j^1 - 1), 0\right), \tag{37}$$

$$P_j^3 = \max(P_j^2 - C \cdot \sum_{k=j+1}^{d} (\delta(q_j, q_k) \wedge H(P_j^2 - Q_k^1 - 1)), 0), \tag{38}$$

$$P_j^4 = \max(P_j^3 - C \cdot \sum_{k=1}^{j-1} (\delta(q_j, q_k) \wedge \delta(Q_j^1, P_k^3)), 0), \tag{39}$$

$$Q_j^2 = \max(Q_j^1 - C \cdot \sum_{k=j+1}^{d} (\delta(q_j, q_k) \wedge H(Q_j^1 - Q_k^1 - 1)), 0), \tag{40}$$

$$P_j^5 = \max(P_j^4 - C \cdot \sum_{k=1, k\neq j}^{d} (\delta(q_j, q_k) \wedge \delta(Q_j^2, Q_k^2) \wedge H(P_j^4 - Q_j^2 - 1)), 0), \tag{41}$$

$$P_j^6 = \max(P_j^5 - C \cdot \delta(Q_j^2, 0), 0), \tag{42}$$

$$Q_j^3 = \max\left(Q_j^2 + 1 - C(1 - \delta(P_j^6, 0) \wedge H(Q_j^2 - 1)), 0\right) + \max\left(Q_j^2 - C(\delta(P_j^6, 0) \wedge H(Q_j^2 - 1)), 0\right). \tag{43}$$

Step 4. For the inward edge at the ℓ-th position, find the position of the k-th adjacent inward edge (child) and its outward edge in $E(T)$, where $k \in \{1, \ldots, n\}$. The variables G_ℓ^k and G'^k_ℓ are equal to i if and only if the inward edge and the outward edge, resp,. of the k-th child of the ℓ-th edge has index i in $E(T)$. G_ℓ^k and G'^k_ℓ are 0 if ℓ corresponds to an outward edge. The variable G''^k_ℓ identifies the position for the insertion of an edge after the last child. $H^0_{\ell j}$ and $H'^0_{\ell j}$ are used to insert leaves before the first child of a node.

$$F_{\ell i} = \max(\sum_{k=1}^{i} A_{\ell k}/2 - C\delta(A_{\ell i,0}), 0), \tag{44}$$

$$G_\ell^k = \sum_{i=1}^{2n} i \cdot \delta(F_{\ell i} + 1/2, k), \tag{45}$$

$$G'^k_\ell = \sum_{i=1}^{2n} i \cdot \delta(F_{\ell i}, k), \tag{46}$$

$$G''^k_\ell = G_\ell^k + \max(G'^{k-1}_\ell + 1 - C(1 - \delta(G_\ell^k, 0)), 0), \tag{47}$$

$$J^0_{\ell j} = q_j + 1, J^k_{\ell j} = G''^k_\ell, \tag{48}$$

$$J'^0_{\ell j} = q_j, J'^k_{\ell j} = G'^k_\ell. \tag{49}$$

Step 5. Find the positions before which new inward and outward edges to be inserted by using the variables L_j and L''_j, respectively.

$$L_j = \sum_{k=0}^{n} \sum_{\ell=0}^{2n} \max(J^k_{\ell j} - C(1 - \delta(\ell, q_j) \wedge \delta(k, Q_j^3)), 0), \tag{50}$$

$$L'_j = \sum_{k=0}^{n} \sum_{\ell=0}^{2n} \max(J'^k_{\ell j} - C(1 - \delta(\ell, q_j) \wedge \delta(k, P_j^6)), 0), \tag{51}$$

$$L''_j = \max(L_j - 1 - C(1 - H(L_j - L'_j)), 0) + \max(L'_j - C \cdot H(L_j - L'_j), 0) + 1. \tag{52}$$

Step 6. Arrange L_j in the ascending order, and then adjust the corresponding entries L''_j and x_j^4 accordingly.

$$R_j = \max(\sum_{k=1}^{d} H(L_j - L_k) - \sum_{k=j}^{d} \delta(L_k, L_j), 0), \tag{53}$$

$$R'_j = \sum_{k=1}^{d} \max(L_k - C(1 - \delta(k, R_j + 1)), 0), \tag{54}$$

$$R''_j = \sum_{k=1}^{d} \max(L''_k - C(1 - \delta(k, R_j + 1)), 0), \tag{55}$$

$$x'^4_j = \sum_{k=1}^{d} \max(x_k^4 - C(1 - \delta(k, R_j + 1)), 0). \tag{56}$$

Step 7. Determine the increment and new positions of the entries of $E(T)$ in $E(U)$ due to the insertions.

$$M_i = \sum_{j=1}^{d} \left(\delta(R'_j, i) + \delta(R''_j, i)\right), \text{ for } i \in \{1, 2, \ldots, 2n+1\}, \tag{57}$$

$$M'_i = i + \sum_{k=1}^{i} M_i, \tag{58}$$

$$N_i^k = \max(t_i - C(1 - \delta(M'_i, i + k - 1)), 0), \quad \text{for } k \in \{1, \ldots, 2d+1\}, \tag{59}$$

$$N'_h = \sum_{h=i+k-1} N_i^k, \text{ for } h \in \{1, \ldots, 2n + 2d\}. \tag{60}$$

Step 8. Determine the positions of the new inward and outward edges in $E(U)$.

$$S_j = R'_j + 2(j-1) - \sum_{k=1}^{j-1} H(R''_k - R'_j) + \sum_{k=1}^{j-1} \delta(R''_k, R'_j), \tag{61}$$

$$S'_j = R''_j + 2d - \sum_{k=j+1}^{d} H(R'_k - R''_j) - \sum_{k=1}^{d} H(R''_k - R''_j) + \sum_{k=1}^{j-1} \delta(R''_k, R''_j). \tag{62}$$

Step 9. Arrange S_j and S'_j in the ascending order, and then adjust the corresponding labels of the new edges.

$$V_k = S_k, 1 \le k \le d, V_k = S'_{k-d}, d+1 \le k \le 2d, \tag{63}$$

$$V'_k = x'^4_k, 1 \le k \le d, V'_k = x'^4_{k-d} + m, d+1 \le k \le 2d, \tag{64}$$

$$W_k = \max(\sum_{r=1}^{2d} H(V_k - V_r) - \sum_{r=k}^{2d} \delta(V_r, V_k), 0), \tag{65}$$

$$W'_k = \sum_{r=1}^{2d} \max(V_r - C(1 - \delta(r, W_k + 1)), 0), \tag{66}$$

$$W''_k = \sum_{r=1}^{2d} \max(V'_r - C(1 - \delta(r, W_k + 1)), 0). \tag{67}$$

Step 10. Insert the new inward and outward edges.

$$Z_i^k = \max(W_k'' - C(1 - \delta(W_k', i + k - 1)), 0), \quad \text{for } k \in \{1, \ldots, 2d\}, i \in \{1, \ldots, 2n + 1\}, \tag{68}$$

$$Z_h' = \sum_{h=i+k-1} Z_i^k, \text{ for } h \in \{1, 2, \ldots, 2(n + d)\}. \tag{69}$$

Finally obtain the Euler string of the desired tree as follows:

$$u_h = N_h' + Z_h', \text{ for } h \in \{1, 2, \ldots, 2(n + d)\}. \tag{70}$$

Notice that all the equations involve maximum function, Heaviside function, δ or $[a \geq \theta]$ function which can be simulated by ReLU activation function by using Theorem 1 by Kumano and Akutsu [23] and Proposition 1 by Ghafoor and Akutsu [33]. The number of variables in these equations is $\mathcal{O}(n^3)$. Therefore we can construct a TI_d-generative ReLU with size $\mathcal{O}(n^3)$ and constant depth. □

Example 6. *Reconsider the rooted tree T with $E(T) = 3, 2, 7, 2, 4, 9, 7, 4, 9, 8$, as shown in Figure 2(a), $d = 4$. We discussed, in detail, the process of insertion to obtain $E(U) = 1, 6, 5, 3, 2, 7, 4,\ 2, 4, 9, 3, 8, 7, 4, 9, 9, 8, 10$ by using $x = 1, 0, 3, 0, 2, 4, 1, 1, 3, 2, 5, 1, 4, 1, 3, 5$ in Fig. 8, and demonstrate the same by using Eqs. (30)-(70) as follows.*

Step 1. Determine the positions of inward and outward edges.

x_j^1 *Specify the node of x_j^1 for insertion with bounds x_j^2 and x_j^3 on the children and insertion value x_j^4. In this case $x = 1, 0, 3, 0, 2, 4, 1, 1, 3, 2, 5, 1, 4, 1, 3, 5$ as depicted in Fig. 8.*

$q_{ji}, w_{\ell i}$ *are explained in Examples 1 and 3, respectively.*

q_j *A variable that gives the position of the inward edge of x_j^1 in $E(T)$, where we consider $q_j = 0$ for $x_j^1 = 0$ and so $q_2 = q_4 = 0$. For example, the inward edges of $x_1^1 = 1$ and $x_3^1 = 3$ have positions 1 and 4, resp., in $E(T)$, and therefore $q_1 = 1$ and $q_3 = 4$ as depicted in Fig. 14(a).*

b_ℓ *A variable that stores the position of the outward edge corresponding to the inward edge, if any, at the ℓ-th position of the Euler string, e.g., $b_1 = 10$ since the inward edge at the 1st position has the outward edge at the 10th position as shown in Fig. 14(a). Similarly, $b_0 = 11 = 2n+1$ (by default), $b_2 = 3$, $b_4 = 7$, $b_5 = 6$, $b_8 = 9$, and all other values are zero.*

Step 2. Determine the number of children of x_j^1.

$A_{\ell i}$ *A binary variable which is one if the i-th inward or outward edge is adjacent with the ℓ-th inward edge in the directed T, e.g., $A_{1,3} = 1$ as the outward edge at 3rd position in the directed T is adjacent with the inward edge at the 1st position as shown in Fig. 14(b). Similarly, $A_{0,1} = A_{0,10} = 1$ (by default), $A_{1,2} = A_{1,4} = A_{1,7} = A_{1,8} = A_{1,9} = A_{4,5} = A_{4,6} = 1$.*

a_ℓ *A variable that gives the number of descendant inward edges that are adjacent with the ℓ-th inward edges, e.g., $a_1 = 3$ as there are three inward edges at the positions $2, 4, 8$ that are adjacent with the edge at the 1st position as shown in Fig. 14(b). In this case, $a_0 = 1$, and $a_4 = 1$. All other values are zero.*

D_j *This variable gives the number of children of x_j^1, e.g., when $x_1^1 = 1$ $D_1 = 3$ as $a_1 = 3$, and there is an inward edge at the 1st position of $E(T)$. Similarly for $x_2^1 = x_4^1 = 0$ and $x_3^1 = 3$, we have $D_2 = D_4 = 1$ and $D_3 = 1$, respectively. The children of $x_1^1 = 1$ are shown in gray in Fig. 14(b).*

Step 3. Refine the invalid lower and upper bounds.

Q_j^1 *A variable that sets $x_j^2 := 0$ if $x_j^2 > D_j$ following the refinement (i) of Table 3. This means that the lower bound is set to 0 if it is greater than the number of children. In this case, $Q_2^1 = 0$ because $x_2^2 = 4 > D_2 = 1$. Whereas $Q_1^1 = x_1^2 = 2$, $Q_3^1 = x_3^2 = 1$ and $Q_4^1 = x_4^2 = 1$.*

P_j^1 *A variable that sets* $x_j^3 := 0$ *if* $x_j^3 > D_j$ *following the refinement (ii) of Table 3. This means that the upper bound is set to* 0 *if it is greater than the number of children. Here,* $P_2^1 = P_3^1 = 0$ *because* $x_2^3 = 2 > D_2 = 1$ *and* $x_3^3 = 5 > D_3 = 1$. *Whereas* $P_1^1 = x_1^3 = 3$ *and* $P_4^1 = x_4^3 = 1$.

P_j^2 *A variable that sets the upper bound* $P_j^1 := 0$ *following the refinement (iii) of Table 3. This means that the upper bound is set to* 0 *if it is smaller than the lower bound. Here,* $P_j^2 = P_j^1$.

P_j^3 *A variable that sets* $P_j^2 := 0$ *following the refinement (iv) of Table 3. Here,* $P_j^3 = P_j^2$.

P_j^4 *A variable that sets* $P_j^3 := 0$ *following the refinement (v) of Table 3. Here,* $P_j^4 = P_j^3$.

Q_j^2 *A variable that sets* $Q_j^1 := 0$ *following the refinement (vi) of Table 3. Here,* $Q_j^2 = Q_j^1$.

P_j^5 *A variable that sets* $P_j^4 := 0$ *following the refinement (vii) of Table 3. Here,* $P_j^5 = P_j^4$.

P_j^6 *A variable that sets* $P_j^5 := 0$ *if* $Q_j^2 = 0$ *following the refinement (viii) of Table 3. This means that the upper bound is set to* 0 *if the lower bound is 0. Here,* $P_j^6 = P_j^5$.

Q_j^3 *A variable to compute* $Q_j^2 + 1$ *if* $P_j^6 = 0$ *following the refinement (ix) of Table 3. Here,* $Q_3^3 = 2$ *as* $P_3^6 = 0$, *whereas* $Q_j^3 = Q_j^2$ *for* $j = 1, 2, 4$.

Step 4. Identify the position of the k-th child of a given node.

$F_{\ell i}$ $F_{\ell i} = k$ *(resp.,* $F_{\ell i} = k - 1/2$*) represents that the outward (resp., inward) edge of the k-th child of the* ℓ*-th inward edge has index i, e.g.,* $F_{0,10} = 1$ *and* $F_{0,1} = 1/2$, *show that the outward edge and the inward edge of the first child of the root have positions* 10 *and* 1, *respectively, as Fig. 14(b). Similarly,* $F_{1,3} = 1$, $F_{1,2} = 1/2$, $F_{1,7} = 2$, $F_{1,4} = 3/2$, $F_{1,9} = 3$, $F_{1,8} = 5/2$, $F_{4,6} = 1$, $F_{4,5} = 1/2$ *and all other values of this variable are* 0.

G_ℓ^k *It gives the position of the inward edge of the k-th child of the inward edge at the* ℓ*-th position, e.g.,* $G_0^1 = 1$ *and* $G_4^1 = 5$ *show that the inward edges of the* 1*st child of the root and the inward edge at* 4 *are* 1 *and* 5, *resp., as shown in Fig. 14(b). Similarly,* $G_1^1 = 2, G_1^2 = 4, G_1^3 = 8, G_4^1 = 5$.

G'^k_ℓ *It gives the position of the outward edge of the k-th child of the inward edge with position* ℓ, *e.g.,* $G'^1_0 = 10$ *and* $G'^1_4 = 6$ *show that the outward edges of the* 1*st child of the root and inward edge at* 4 *are* 10*th and* 6*th positions, resp., as shown in the Fig. 14(b). Similarly* $G'^1_1 = 3, G'^2_1 = 7, G'^3_1 = 9, G'^1_4 = 6$.

G''^k_ℓ *This variable determines the position of the child, if any, to be inserted to the right of the children of the inward edge at the* ℓ*-th position, when k is equal to* $D(\ell) + 1$. *When* $k < D(\ell)$, *this variable determines the position of the k-th child, whereas* G''^k_ℓ *will be ignored when* $k > D(\ell) + 1$, *e.g., when* $\ell = 0$, $D(0) = 1$, $G''^2_0 = 11$ *means that an inward edge as a child of the root on the right will be inserted at the 11th position as shown in Fig. 14(b),* $G''^1_0 = 1$ *is the position of the 1st child, and* $G''^3_0 = 1$ *will be ignored. Similarly,* $G''^4_1 = 10$ *and* $G''^2_4 = 7$.

$J_{\ell j}^k$ *For* $k = 0$, *this variable determines the position of the inward edge of the child, if any, to be inserted to the left of children of the inward edge at the* ℓ*-th position when* ℓ *is the position of the inward edge of* x_j^1. *For example,* $J_{0,2}^0 = 1$ *means that an inward edge as a child of the root will be inserted before the 1st position as shown in Fig. 14(c). Also,* $J_{1,1}^0 = 2$, $J_{4,3}^0 = 5$, *and* $J_{0,4}^0 = 1$. *For* $k \geq 1$ *and any j,* $J_{\ell j}^k = G''^k_\ell$. *For example,* $J_{1,2}^4 = G''^4_1 = 10$.

$J'^k_{\ell j}$ *For* $k = 0$, *this variable determines the position of the outward edge of the child, if any, to be inserted to the left of children of the inward edge at the* ℓ*-th position when* ℓ *is the position of the inward edge of* x_j^1. *For example,* $J'^0_{0,2} = 0$ *means that an outward edge as a child of the root will be inserted after the 0th position as shown in Fig. 14(c). Also,* $J'^0_{1,1} = 1$, $J'^0_{4,3} = 4$, *and* $J'^0_{0,4} = 0$. *For* $k \geq 1$ *and any j,* $J'^k_{\ell j} = G'^k_\ell$. *For example,* $J'^1_{4,2} = G'^1_4 = 6$

Step 5. Identify the insertion positions.

L_j *This variable identifies the position for the new inward edge to be inserted corresponding to* x_j^1, *e.g.,* $L_1 = 4$ *means that the first new inward edge will be inserted at the 3rd position (before the 4th entry of* $E(T)$*). Similarly, we get* $L = [4, 1, 7, 1]$, *as shown in the Fig. 14(c).*

L'_j *This variable keeps the valid* $J'^k_{\ell j}$, *i.e., when the position of the inward edge of* x_j^1 *is* ℓ *and k is equal to the refined, if necessary, upper bound corresponding to* x_j^3. *Thus* $L' = [9, 0, 4, 10]$.

L''_j *This variable identifies the position of the new outward edge to be inserted corresponding to* x_j^1*, e.g.,* $L''_1 = 10$ *means that the new outward edge corresponding to* x_j^1 *will be inserted at the 9th position (before the 10th entry of* $E(T)$*). Similarly, we get* $L'' = [10, 1, 7, 11]$*, as shown in the Fig. 14(c).*

Step 6. Arrange L_j *in the ascending order, and adjust* L''_j*,* x_j^4 *accordingly.*

R_j *This variable identifies the position of* L_j *in the arranged* L*, e.g.,* $R_1 = 2$ *means that in the ascending order,* L_1 *will appear at the 2nd position. So* $R = [2, 0, 3, 1]$.

R'_j *This variable arranges the value of* L_j *in the ascending order w.r.t.* R_j*. In this case,* $R' = [1, 1, 4, 7]$.

R''_j *This variable arranges the value of* L''_j *w.r.t.* R_j*. So,* $R'' = [1, 11, 10, 7]$.

x'^4_j *This variable arranges the value of* x_j^4 *w.r.t.* R_j*. So,* $x'^4 = [1, 5, 4, 3]$.

Step 7. Determine the increment and new positions of the entries of $E(T)$ *in* $E(U)$ *due to the insertions.*

M_i *This variable identifies the number of insertions before the* i*-th position that corresponds to some* x_j^k*, e.g.,* $M_4 = 1$ *means that there will be one insertion before the 4th position of* $E(T)$ *as shown in Fig. 14(d). The non-zero values are* $M_1 = 3$*,* $M_7 = 2$*,* $M_{11} = 1$.

M'_i *Determines the new position of the* i*-th entry of* $E(T)$ *by summing up the increments before it, e.g.,* $M'_4 = 8$ *since there are four increments* $M_1 = 3$ *and* $M_4 = 1$*, which implies that the 4th entry of* $E(T)$ *will be at the 8th entry of* $E(U)$*, as shown in the Fig. 14(d). In this case* $M' = [4, 5, 6, 8, 9, 10, 13, 14, 15, 17]$.

N_i^k *This variable links the* i*-th position with its increment and label, e.g.,* $N_4^5 = 2$ *means that the 4th entry of* $E(T)$ *has an increment of* $k - 1 = 5 - 1 = 4$ *and label* 2*, as shown in the Fig. 14(d). Similarly,* $N_1^4 = 3, N_2^4 = 2, N_3^4 = 7, N_5^5 = 4, N_6^5 = 9, N_7^7 = 7, N_8^7 = 4, N_9^7 = 9, N_{10}^8 = 8$.

N'_h *Finally, this variable lists the* i*-th entry of* $E(T)$ *with an increment* k *at the position* h *if* $h = i + k - 1$*. In this case* $N' = [0, 0, 0, 3, 2, 7, 0, 2, 4, 9, 0, 0, 7, 4, 9, 0, 8, 0]$.

Step 8. Determine the positions of the new inward and outward edges in $E(U)$.

S_j *It gives the position of the new inward edge corresponding to* R'_j*, e.g.,* $S_j = 1$ *means that an inward edge corresponding to* $R'_1 = 1$ *will be inserted at the 1st position of* $E(U)$*. Thus* $S = [1, 3, 7, 11]$*, as shown in the Fig. 14(d).*

S'_j *It gives the position of the new outward edge corresponding to* R'_j*. In this case* $S' = [2, 18, 16, 12]$*, as shown in the Fig. 14(d).*

Step 9. Arrange S_j *and* S'_j *in the ascending order, and then adjust the corresponding labels of the new edges.*

V_k *This variable concatenates* S *and* S'*, and so* $V = [1, 3, 7, 11, 2, 18, 16, 12]$.

V'_k *This variable lists the label of the new inward edge or outward edge corresponding to* V_k*. In this case* $V' = [1, 5, 4, 3, 6, 10, 9, 8]$.

W_k *This variable identifies the position of* V_k *in the arranged* V*. In this case* $W = [0, 2, 3, 4, 1, 7, 6, 5]$.

W'_k *This variable arranges* V_k *w.r.t.* W_k*. In this case,* $W' = [1, 2, 3, 7, 11, 12, 16, 18]$.

W''_k *This variable arranges the value of the label* V'_k *w.r.t.* W_k*. So,* $W'' = [1, 6, 5, 4, 3, 8, 9, 10]$.

Step 10. Insert the new inward and outward edges.

Z_i^k *The variable* $Z_i^k = W''_k$ *if and only if* W''_k *will be inserted as a new label at the* $(i + k - 1)$*-th position in the resultant string, e.g.,* $Z_7^5 = 3$ *means that the 5th insertion has label* 3 *is performed before* 7*th position of* $E(T)$*, and at* $i + k - 1 = 11$*-th position in the resultant string as shown in the Fig. 14(d). Also,* $Z_1^1 = 1, Z_1^2 = 6, Z_1^3 = 5, Z_4^4 = 4, Z_7^6 = 8, Z_{10}^7 = 9, Z_{11}^8 = 10$*, and all other values are* 0.

Z'_h *This variable lists the label of the inward edge or outward edge* W''_k *at the position* h *if* $h = i + k - 1$*. Thus,* $Z' = [1, 6, 5, 0, 0, 0, 4, 0, 0, 0, 3, 8, 0, 0, 0, 9, 0, 10]$.

u_h *This variable sum up the entries of* N'_h *and* Z'_h *to obtained the resultant Euler string which, in this case, is* $E(U) = [1, 6, 5, 3, 2, 7, 4, 2, 4, 9, 3, 8, 7, 4, 9, 9, 8, 10]$.

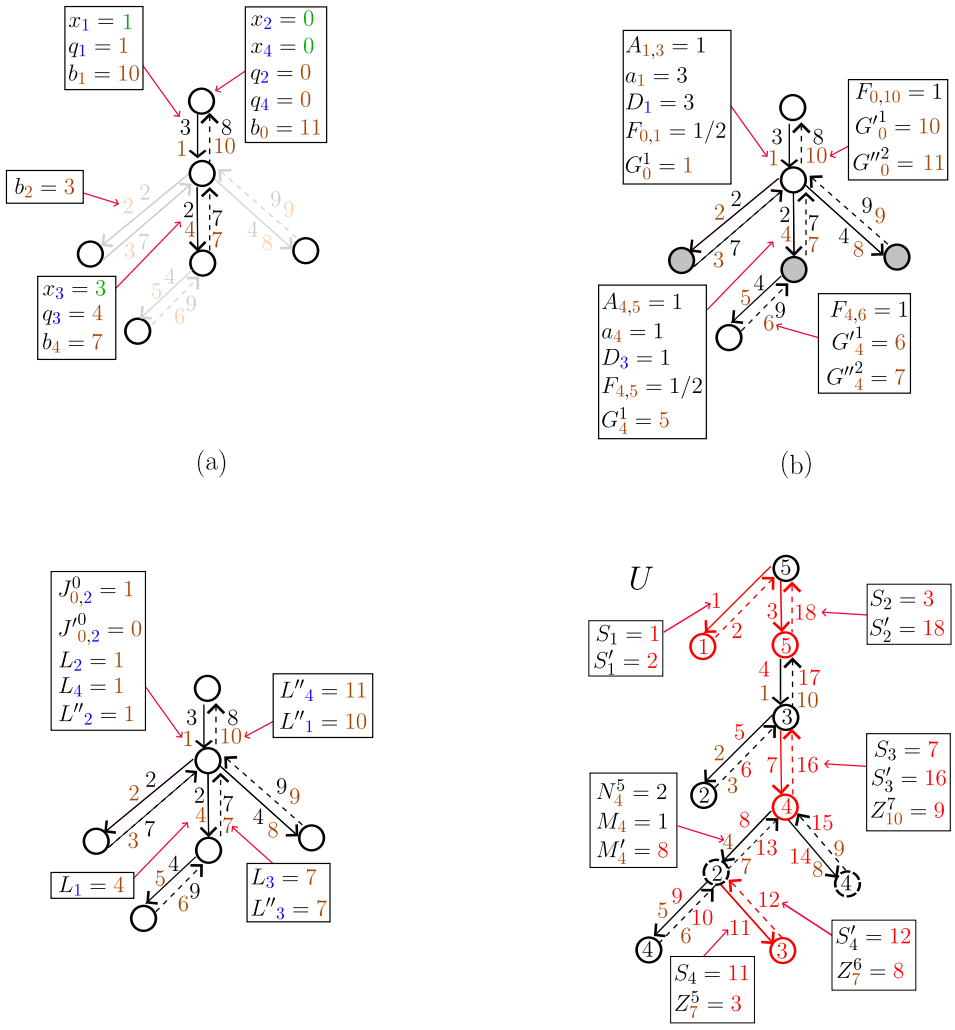


Figure 14: An illustration of the variables used in Theorem 3.

Proof of Theorem 4. Consider two trees T and U over Σ such that $E(U)$ is obtained from $E(T)$ by performing d edit operations based on an appropriate $x = x_1, \ldots, x_{7d}$. The edit operations on $E(T)$ to obtain $E(U)$ can be performed in the following steps, where B and C are large numbers with $C \gg B \gg \max(m, n)$.

Step 1. Convert the input x_j into integers. The variables P'_j and Q'_j store the integer values corresponding

to x_j, i.e., $P'_j = i$ where $i \in \{0, \ldots, n\}$, (resp., $Q'_j = \ell$ where $\ell \in \Sigma$) if and only if $P_i^j = 1$ (resp., $Q_\ell^i = 1$).

$$P_i^j = [(i-1)/n \le x_j \le i/n] - \delta(x_j, (i-1)/n), \\ \text{for } i \in \{0, 1, \ldots, n\}, j \in \{1, \ldots, 2d, 3d+1, \ldots, 6d\}, \tag{71}$$

$$Q_\ell^j = \begin{cases} [(\ell-1)/m \le x_j \le \ell/m] & \text{if } \ell = 1, \\ [(\ell-1)/m \le x_j \le \ell/m] - \delta(x_j, (\ell-1)/m), & \text{if } \ell \in \{2, \ldots, m\}, \end{cases} \\ \text{for } j \in \{2d+1, \ldots, 3d, 6d+1, \ldots, 7d\}, \tag{72}$$

$$P'_j = \sum_{i=0}^{n} p_i^j \cdot i \text{ for } j \in \{1, \ldots, 2d, 3d+1, \ldots, 6d\}, \tag{73}$$

$$Q'_j = \sum_{\ell=1}^{m} q_\ell^j \cdot \ell \text{ for } j \in \{2d+1, \ldots, 3d, 6d+1, \ldots, 7d\}. \tag{74}$$

Step 2. Ignore x_j, $1 \le j \le 2d$, which are zero or repeated to avoid redundant deletion and substitution operations. Similarly, ignore x_j, $1 \le j \le 2d$ which has index greater than d among the non-zero and non-repeated positions. For $x_{j=3d+h}$, $1 \le h \le d$ set weights $d - h + 1$. Finally, identify the valid operation positions in x with index at most d using R'_j.

$$R_j = \begin{cases} \max(1 - (\delta(P'_j, 0) + \sum_{k=1}^{j-1} \delta(P'_j, P'_k)), 0) \\ \quad \text{for } j \in \{1, \ldots, d\}, \\ \max(1 - (\delta(P'_j, 0) + \sum_{k=d+1}^{j-1} \delta(P'_j, P'_k)), 0) \\ \quad \text{for } j \in \{d+1, \ldots, 2d\}, \\ d + 1 - \sum_{k=3d+1}^{j} H(P'_k) \quad \text{for } j \in \{3d+1, \ldots, 4d\}, \end{cases} \tag{75}$$

$$R'_j = \begin{cases} \left[\sum_{k=1}^{j} R_k \ge d+1\right] \\ \quad \text{for } \ j \in \{1, \ldots, 2d\}, \\ \left[\sum_{k=1}^{2d} R_k + R_j \ge d+1\right] \\ \quad \text{for } j \in \{3d+1, \ldots, 4d\}. \end{cases} \tag{76}$$

Step 3. Set the value of the redundant positions and their corresponding bounds and values B.

$$S_j = \max(B - C(1 - R'_j), 0) + \max(P'_j - \\ C \cdot R'_j), 0) \text{ for } j \in \{1, \ldots, 2d, 3d+1, \ldots, 4d\}, \tag{77}$$

$$S'_j = \begin{cases} \max(B - C(1 - \delta(S_{j-d}, B)), 0) + \max(Q'_j - \\ \quad C\delta(S_{j-d}, B), 0) \text{ for } j \in \{4d+1, \ldots, 5d\}, \\ \max(B - C(1 - \delta(S_{j-2d}, B)), 0) + \max(Q'_j - \\ \quad C\delta(S_{j-2d}, B), 0) \text{ for } j \in \{5d+1, \ldots, 6d\}, \\ \max(B - C(1 - \delta(S_{j-3d}, B)), 0) + \max(Q'_j - \\ \quad C\delta(S_{j-3d}, B), 0) \text{ for } j \in \{6d+1, \ldots, 7d\}. \end{cases} \tag{78}$$

Finally, get the preprocessed input x'_j as follows:

$$x'_j = \begin{cases} S_j & \text{for } j \in \{1, \ldots, 2d\}, \\ x_j & \text{for } j \in \{2d+1, \ldots, 3d\}, \\ S_j & \text{for } j \in \{3d+1, \ldots, 4d\}, \\ S'_j & \text{for } j \in \{4d+1, \ldots, 7d\}. \end{cases} \tag{79}$$

Step 4. Apply deletion operations on padded $E(T)$ by following Theorem 2 with x'_j, $j \in \{1, \ldots, d\}$ as an input to get $E(T')$. Apply substitution operations on $E(T')$ following Theorem 1 using x'_j, $j \in \{d+1, \ldots, 3d\}$, to get $E(T'')$. Apply insertion operations on $E(T'')$ using Theorem 3 and x'_j, $j \in \{3d+1, \ldots, 7d\}$ to get $E(T''')$. During substitution and insertion operations, replace Eq. (5) $r'_{ji} = t_i \cdot q_{ji}$ with $r'_{ji} = \max(t_i - C(1-\delta(q_{ji}, 1), 0)$. Similarly, replace Eq. (6) $r_i = t_i \cdot p_i$ and Eq. (13) $z'_{ji} = t_i \cdot \sum_{\ell=1}^{2n} w'_{j\ell i}$ with $r_i = \max(t_i - C(1 - \delta(p_i, 1), 0)$ and $z'_{ji} = \max(t_i - C(1 - \delta(\sum_{\ell=1}^{2n} w'_{j\ell i}, 1), 0)$, respectively. Finally, the required $E(U)$ can be obtained by trimming Bs from $E(T''')$. Notice that all equations involve maximum function, Heaviside function, δ or $[a \geq \theta]$ function, and therefore due to Theorems 1, 2 and 3, there exists a TE_d-generative ReLU network with size $\mathcal{O}(n^3)$ and constant depth. □

Table 7: Conversion table from real to integers.

For positions x_j with $1 \leq j \leq 2d$ and $3d+1 \leq j \leq 6d$	For values x_j with $2d+1 \leq j \leq 3d$ and $6d+1 \leq j \leq 7d$
$(-1/5, 0] \to 0$	$[0, 1/10] \to 1$
$(0, 1/5] \to 1$	$(1/10, 2/10] \to 2$
$(1/5, 2/5] \to 2$	$(2/10, 3/10] \to 3$
$(2/5, 3/5] \to 3$	$(3/10, 4/10] \to 4$
$(3/5, 4/5] \to 4$	$(4/10, 5/10] \to 5$
$(4/5, 5/5] \to 5$	$(5/10, 6/10] \to 6$
	$(6/10, 7/10] \to 7$
	$(7/10, 8/10] \to 8$
	$(8/10, 9/10] \to 9$
	$(9/10, 10/10] \to 10$

Example 7. *Reconsider the tree T given in Fig. 2 with $E(T) = 3, 2, 12, 2, 4, 14, 12, 4, 14, 13$, $d = 3$, $m = 10$, and $x = 0.3, 0, 0.38, 0, 0.46, 0.55, 0, 0.6, 0.88, 0.66, 0.75, 0, 0.55, 0.87, 0.03, 0.02, 0.45, 0.09, 0, 0.7, 0.5$. We demonstrate the process of obtaining $E(U)$ by using Theorem 4. We explain the meaning of Eqs. (71)- (79), whereas the details of the deletion, substitution and insertion operations can be followed from Examples 5, 4 and 6.*

- P_i^j *Specify the interval for each position x_j, where $j \in \{1, \ldots, 2d, 3d+1, \ldots, 6d\}$. $P_i^j = 1$ means that the j-th input lies in the i-th interval, i.e., the interval $((i-1)/n, i/n]$. In this case $P_0^2 = P_0^4 = P_0^{12} = P_1^{15} = P_1^{16} = P_1^{18} = P_2^1 = P_2^3 = P_3^5 = P_3^6 = P_3^{13} = P_3^{17} = P_4^{10} = P_4^{11} = P_5^{14} = 1$. All other values are zero.*
- Q_ℓ^j *Specify the interval for each value x_j, where $j \in \{2d+1, \ldots, 3d, 6d+1, \ldots, 7d\}$. $Q_\ell^j = 1$ means that the j-th input lies in the ℓ-th interval, i.e., for $\ell = 1$ (resp., $\ell \geq 2$), x_j lies in $[0, \ell/m]$ (resp., $((\ell-1)/m, \ell/m]$). In this case, $Q_1^7 = Q_1^{19} = Q_5^{21} = Q_6^8 = Q_7^{20} = Q_9^9 = 1$, and all other values are zero.*
- P'_j *Assigns each position x_j an integer i if x_j belongs to the i-th interval, i.e., $P_i^j = 1$. $P'_1 = 2$, $P'_2 = 0$, $P'_3 = 2$, $P'_4 = 0$, $P'_5 = 3$, $P'_6 = 3$, $P'_{10} = 4$, $P'_{11} = 4$, $P'_{12} = 0$.*
- Q'_j *Assigns each value x_j an integer ℓ if x_j belongs to the ℓ-th interval, i.e., $Q_\ell^j = 1$. $Q'_7 = 1$, $Q'_8 = 6$, $Q'_9 = 9$, $Q'_{19} = 1$, $Q'_{20} = 7$, $Q'_{21} = 5$.*
- R_j *$R_j = 1$ if x_j, $1 \leq j \leq 2d$, is a non-zero and non-repeated position. For $3d+1 \leq j \leq 4d$, R_j is a weight from $\{d, d-1, \ldots, 1\}$ assigned to x_j in the descending order. In this case $R_1 = R_5 = 1$ for $1 \leq j \leq 2(3)$, whereas for $3(3)+1 \leq j \leq 4(3)$, $R_{10} = 3, R_{11} = 2, R_{12} = 1$.*
- R'_j *The variable $R'_j = 1$ if the position $x_j, 1 \leq j \leq 2d$ among the non-zero and non-repeated entries is at least $d+1$. Similarly, for $x_j, 3d+1 \leq j \leq 4d$, $R'_j = 1$ if the sum of the number of non-zero and non-repeated position entries before x_{3d+1} and weight R_j is at least $d+1$. In this case, $R'_{10} = R'_{11} = 1$. $R'_j = 0$ for $x_j, 1 \leq j \leq 2d$.*
- S_j *$S_j = B$ if x_j has index at least $d+1$ among the positions, i.e., $S_j = B$ if $R'_j = 1$ otherwise it is P'_j. In this case $S_{10} = S_{11} = B$. For all other values of j, $S_j = P'_j$*
- S'_j *If $S_j = B$ for $3d+1 \leq j \leq 4d$, then the corresponding bounds and values of x_j are also set to B, i.e, $S'_j = B$. In this case $S'_{13} = S'_{14} = S'_{16} = S'_{17} = S'_{19} = S'_{20} = B$.*

x'_j *Gives the preprocessed input for edit operations;* $x' = [2, 0, 2, 0, 3, 3, 1, 6, 9, B, B, 0, B, B, 1, B, B, 1, B, B, 5]$.

Apply deletion operations on padded $E(T)$ *to get* $E(T') = 3, 2, 4, 14, 12, 4, 14, 13, B, B$*, substitution operations to get* $E(T'') = 3, 2, 6, 16, 12, 4, 14, 13, B, B$*, and insertion operations to get* $E(T''') = B, B, B, B, 5, 3, 2, 6, 16, 12, 4, 14, 13, 15, B, B$*. Finally, obtain* $E(U) = 5, 3, 2, 6, 16, 12, 4, 14, 13, 15$ *by trimming* B*s as shown in Fig. 9.*

Examples of Code Execution

All codes are freely available at `https://github.com/MGANN-KU/TreeGen_ReLUNetworks`. An explanation of the program codes is given below.

The file `Finding_outward_edges.py` contains an implementation of the proposed generative ReLU to find the indices and labels of outward edges of the given inward edges in the Euler string.
Input:
`t` := Input Euler string of length `2n`
`m` := The size of the symbol set
`x` := The string of length `d` to identify inward edges
Output:
`y` := The Outward edges of `t` following `x`.
An example: `t = 3, 2, 7, 2, 4, 9, 7, 4, 9, 8,`
`d = 3,`
`m = 5,`
`x = 1, 3, 0,`
`y = 10, 7, 0.`

The file `TS_d.py` contains an implementation of TS_d-generative ReLU to generate Euler strings with a given Edit distance due to substitution.
Input:
`t` := Input Euler string of length `2n`
`d` := Edit distance
`m` := The size of the symbol set
`x` := The string of length `2d` to identify substitution operation
Output:
`u` := The Euler string obtained by applying substitution operation on `t` following `x` and has at most distance `2d`.
An example: `t = 3, 2, 7, 2, 4, 9, 7, 4, 9, 8,`
`d = 3,`
`m = 5,`
`x = 1, 3, 1, 5, 1, 2,`
`u = 5, 2, 7, 1, 4, 9, 6, 4, 9, 10.`

The file `TD_d.py` contains an implementation of TD_d-generative ReLU to generate Euler strings with a given edit distance due to deletion operation only.
Input:
`t` := Input Euler string of length `2n`
`d` := Edit distance
`m` := The size of the symbol set
`x` := The string of length `d` to identify deletion operation
Output:
`u` := The Euler string obtained by applying deletion operation and trimming B on `t` following `x` and has distance `2d`.
An example: `t = 3, 2, 7, 2, 4, 9, 7, 4, 9, 8,`
`d = 3,`

m = 5,
x = 1, 3, 1,
u = 2, 7, 4, 9, 4, 9.

The file TI_d.py contains an implementation of TI_d-generative ReLU to generate strings with a given edit distance due to insertion operation only.
Input:
t := Input Euler string of length 2n
d := Edit distance
m := The size of the symbol set
x := The string of length 4d to identify insertion operation
Output:
u := The string obtained by applying insertion operation on t following x and has distance 2d.
An example: t = 3, 2, 7, 2, 4, 9, 7, 4, 9, 8,
d = 4,
m = 5,
x = 1, 0, 3, 0, 2, 4, 1, 1, 3, 2, 5, 1, 4, 1, 3, 5,
u = 1, 6, 5, 3, 2, 7, 4, 2, 4, 9, 3, 8, 7, 4, 9, 9, 8, 10.

The file TE_d_unified.py contains an implementation of TE_d-generative ReLU to generate strings with a given edit distance due to deletion, substitution and insertion operations simultaneously.
Input:
t := Input Euler string of length 2n
d := Edit distance
m := The size of the symbol set
Δ := The small number
x := The string of length 7d to identify substitution, insertion and deletion operations
Output:
u := The string obtained by applying deletion, substitution, and insertion operations simultaneously on t following x and has at most distance 2d.
An example: t := 3, 2, 12, 2, 4, 14, 12, 4, 14, 13,
d := 3,
m := 10,
Δ := 0.01
x := 0.3, 0, 0.38, 0, 0.46, 0.55, 0, 0.6, 0.88, 0.66, 0.75, 0, 0.55,0.87, 0.03, 0.02, 0.45, 0.09, 0, 0.7, 0.5,
u := 5, 3, 2, 6, 16, 12, 4, 14, 13, 15.