GoTreeScape: Navigate and Explore the Tree Visualization Design Space

Guozheng Li and Xiaoru Yuan, *Senior Member, IEEE*

Abstract—Declarative grammar is becoming an increasingly important technique for understanding visualization design spaces. The GoTreeScape system presented in the paper allows users to navigate and explore the vast design space implied by GoTree, a declarative grammar for visualizing tree structures. To provide an overview of the design space, GoTreeScape, which is based on an encoder-decoder architecture, projects the tree visualizations onto a 2D landscape. Significantly, this landscape takes the relationships between different design features into account. GoTreeScape also includes an exploratory framework that allows top-down, bottom-up, and hybrid modes of exploration to support the inherently undirected nature of exploratory searches. Two case studies demonstrate the diversity with which GoTreeScape expands the universe of designed tree visualizations for users. The source code associated with GoTreeScape is available at https://github.com/bitvis2021/gotreescape.

Index Terms—Tree visualization, design space exploration, deep learning

1 Introduction

Researchers have proposed many declarative grammars for visualizations [1], [2], [3], [4], [5], [6]. These grammars build design spaces by decomposing visualizations into multiple different dimensions, each presenting different properties of a layout. Declarative grammars balance fine-grained design controls with the burden of constructing tree visualizations by specifying what to render. However, users may find it difficult to navigate and explore the design space implied by a grammar. Yet this is an important aspect of enlarging the set of design possibilities that are known to visualization designers—i.e., the known space—and also the solutions that the designers can actively consider—i.e., the consideration space [7].

Within the realm of information visualization, visualizing tree structures is a basic and fundamental task, with the literature offering hundreds of techniques for doing so [8]. Many software applications, programming libraries, and other techniques allow users to author tree visualizations, including general tools like D3 [9], Vega [2], and Tableau[1] as well as approaches tailored specifically for trees like GoTree/Tree Illustrator [4], [10] and the generative layout approach [11].

However, much previous research on authoring tree visualizations assumes that users have a clear target visualization in mind. Yet, in many cases, one's design objectives may only be loosely-specified, with the user finding themselves seeking a suitable solution from the design space. For example, a designer may want to visualize astronomical hierarchical data related to the solar system using a ring-shaped tree visualization, for a visual style consistent with the subject matter. Alternatively, perhaps the designer has a limited knowledge of all design options and does not know which tree visualizations might meet his/her requirements. S/he may not know whether a better tree visualization design exists nor how to choose the other design dimensions needed to reach an appropriate final solution. Yet, in general, supporting the exploratory design [12] of tree visualizations in such application scenarios is still an under-explored problem.

That said, there have been a few studies on exploratory design as well as exploratory visual analysis (EVA). When conducting an EVA, analysts have a vague hypothesis or an ill-defined task in mind. Similarly, exploratory design begins with loosely-specified design goals and proceeds in an opportunistic and serendipitous manner. These studies on exploratory design [13], [14], [15], [16], [17] and EVA [18], [19], [20], [21] mainly offer ways to explore a parametric space. Additionally, studies on EVA focus on changing the underlying data variables, that is *data* variations, while exploratory design generally involves tweaking the design parameters, i.e., *design* variations. What these studies do not address is how to support the exploratory design of tree visualizations. Overcoming this problem involves at least two challenges:

The first challenge is providing an overview of the tree visualization design space. This space is often extremely large, encoding both topological and node attributes with

1. https://www.tableau.com/

- *Guozheng Li was with the School of AI, Peking University, Beijing 100871, China. He is now with the School of Computer Science and Technology, Beijing Institute of Technology, Beijing 100811, China. E-mail: guozhg.li@gmail.com.*
- *Xiaoru Yuan is with the Laboratory of Machine Perception (Ministry of Education), School of AI, Peking University and National Engineering Laboratory for Big Data Analysis and Application, Peking University, Beijing 100871, China. E-mail: xiaoru.yuan@pku.edu.cn.*

Manuscript received 3 November 2021; revised 21 September 2022; accepted 7 October 2022. Date of publication 0 2022; date of current version 0 2022.
This work was supported in part by the National Key Research and Development Program of China under Grant 2021YFB3301502, in part by NSFC under Grant 61872013, and in part by the Beijing Institute of Technology Research Fund Program for Young Scholars.
(Corresponding author: Xiaoru Yuan.)
Recommended for acceptance by S.G. Kobourov.
This article has supplementary downloadable material available at https://doi.org/10.1109/TVCG.2022.3215070, provided by the authors.
Digital Object Identifier no. 10.1109/TVCG.2022.3215070

many visual channels. Ways of quantifying the similarity between different tree visualizations in a way that matches human perception are not necessarily obvious. For example, one could generate two congruent tree visualizations by swapping the layout-related design dimensions along the x and y-axes. Although the "edit distance" between these two visualizations' grammars may be large (because many design dimensions are different), the results would like be perceived as being extremely similar. Moreover, design dimensions will have different impacts on the visualization results. For example, changing a *Cartesian* coordinate system to a *polar* coordinate system influences both the relative positions and the shapes of nodes in the tree visualization, whereas changing the node type from *circle* to *triangle* only influences the shape of the node.

The second challenge involves how to provide a flexible approach to exploratory design, where users have the option to start from a loosely-specified goal and make subsequent decisions to identify a concrete solution. In some cases, the user might begin with a tentative design as a starting point and wish to confirm if better visualizations are available in the design space. In other cases, the user might begin with queries that partially restrict the set of possible solutions, or may even begin with no preconceived design and wish to freely explore. The decision-making process of the user can also be highly variable. It may be directed toward a clear goal; it may involve determining design choices for certain dimensions; or it may involve backtracking and starting over due to some new inspiration.

To address these challenges, we propose GoTreeScape, a system providing an overview of a land*Scape* of tree visualizations described by *GoTree* [4]. GoTreeScape allows users to control their exploratory design process while supporting wide variations in requirements. To project the set of possible tree visualizations onto a two-dimensional space, GoTreeScape uses a variational autoencoder (VAE) to map 62340 tree visualizations (as described by GoTree) onto a latent space, in which nearby points decode to similar tree visualizations. This training integrates domain expertise about what makes two tree visualizations look similar and also which GoTree design dimensions have a more significant impact on the tree visualization results than others (Section 4.2.2). To avoid excessive clutter, GoTreeScape displays landmarks in the design space, which are representative tree visualizations, and shows a density-based contour indicating other possible design choices rather than all discrete points. To enable flexible exploratory design, GoTreeScape incorporates an exploratory framework supporting top-down, bottom-up, and hybrid exploration modes. In addition, it allows for a data-oriented exploration of the design space where users upload their hierarchical data and can then generate all tree visualization results based on those data. Driven by the considerations distilled from existing studies on exploratory design and EVA [18], [19], [20], GoTreeScape visualizes the tree visualization design space through a landscape metaphor and supports navigation and exploration by users.

To evaluate the usability of GoTreeScape, we had one visualization designer and one visualization researcher apply the system to their own scenario of tree visualization design. These two case studies demonstrate the system's utility. The results not only show that GoTreeScape allows users to find desirable solutions but also that GoTreeScape expands the diversity of user-designed tree visualizations.

Our contributions include: (1) A novel approach to constructing a tree visualization design space as a holistic landscape; (2) An exploratory framework supporting varying user requirements and scenarios; and (3) A prototype system for navigating and exploring tree visualization design spaces.

2 RELATED WORK

This section reviews the literature on tree visualizations, and, particularly, tree visualization frameworks, as well as the literature on exploring design spaces.

2.1 Tree Visualization

Tree visualizations can be categorized into implicit and explicit techniques depending on how the parent-child relations in hierarchical data are visually represented. Explicit techniques emphasize topological structures by explicitly encoding parent-child relationships into the tree's visual elements, e.g., arcs [22], straight lines [23], and curves [24]. By contrast, implicit techniques are potentially more space-efficient because they encode the parent-child relations into relative positions between the nodes, e.g., containment [25] and adjacency [26]. Additionally, hybrid techniques that combine the advantages of two or more approaches have also been proposed [27]. Beyond novel tree visualizations, researchers have also proposed various ways of capturing and describing the vast design spaces in a unified way through *graphical* building blocks. Schulz et al. [8], for example, are collecting tree visualizations on treevis.net, with over 330 assembled to date. Exceeding the boundaries of a collection, treevis.net also classifies tree visualizations against three design criteria, namely dimensionality (2D, 3D, and hybrid), edge representation (explicit, implicit, and hybrid), and node alignment (radial, axis-parallel, or free). Li et al. [28] subsequently extended these three design criteria into 12 design features, and also constructed a phylogenetic tree to show evolutionary relationships. Similar to GoTreeScape, this study also supports exploring tree visualization designs. However, the phylogenetic tree comprises just 35 tree visualizations, while GoTreeScape allows users to explore and navigate a vast design space implied by a fine-grained declarative grammar.

Although useful for classifying design choices, the above design dimensions are not fine-grained enough to generate concrete tree visualizations. To overcome this problem, some researchers have looked to categorize all possible tree visualizations into subclasses. Others have proposed descriptive approaches to support the fine-grained specifications. For example, Schulz and Hadlak [29] proposed an approach to exploration based on presets that allows users to construct new designs by blending several existing visual representations. They feature five design dimensions: explicit/implicit, structure/attribute, aligned/cascaded, inclusion/adjacency, axis-parallel/radial and exemplify preset-based method on tree visualizations. As shown in Fig. 1, blending a *radial node-link layout* with a *nested squarified treemap* produces a *nested squarified pietree*. In terms of implicit tree visualizations,

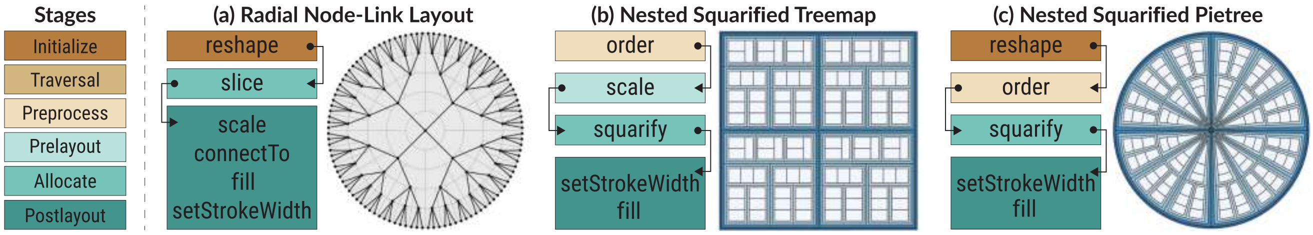

Fig. 1. Three typical tree visualizations and their step-wise creation process. A radial node-link layout (a) is an explicit tree visualization. A nested squarified treemap (b) and a nested squarified pietree (c) are both implicit tree visualizations. Users can construct (c) by setting (a) and (b) as presets during the exploration phase of the visualization design process.

Schulz et al. [30] divide the design space along four dimensions: dimensionality, node representation, edge representation, and layout. With *layout* containing more fine-grained parameters, such as *subdivision* and *packing*. Li et al. developed GoTree [4], a declarative grammar for tree visualizations, and Tree Illustrator [10], which is an interactive authoring tool for further reducing the burden of constructing visualizations imposed by GoTree. Spurred on by the capabilities of GoTree, we leveraged this application to construct the tree visualization design space for GoTreeScape.

Beyond *graphical* building blocks, such as visual elements and properties, some tree visualization frameworks use *functional* building blocks, i.e., operators. The generative layout approach [11] involves a construction pipeline with six stages for constructing implicit and explicit tree visualizations. The six stages include initialization, traversal, pre-process, pre-layout, allocate, and post-layout. Further, a set of operators is defined for each stage. Fig. 1 shows the step-wise creation process of three tree visualizations based on these operators. Many operator-based tree visualization frameworks also focus on the subcategory of the tree visualizations, especially for space-filling tree visualization layouts. For instance, Baudel and Broeksema [31] use five dimensions, namely, order, size, chunk, recurse, and phrase, to drive space-filling layouts. Existing studies [32], [33] also use operators to configure a hierarchical layout to visualize multivariate data.

Hence, overall, the current literature organizes the tree visualization design space and supports the rapid prototyping of tree visualizations, but it does not allow users to effectively perform open-ended explorations of the design space when the user's targets are not well-defined.

2.2 Design Space Exploration

The design space extracts preliminary building blocks from existing visualizations and builds a space for visualizing possible designs, both existing and novel, by assembling all possible combinations of the building blocks. Visualization design spaces can help guide users in the design process by supporting them to understand single visualizations and their relationships. [34], [35], [36]. For example, by examining existing implicit tree visualizations, Schulz, Hadlak, and Schumann [30] identified four independent building blocks: dimensionality, node representation, edge representation, and layout. These serve as axes for constructing the design space. Card et al. [34] structure the visualization design space by treating the data properties as an important aspect of representation. By contrast, Tory and Möller [37] provide a high-level taxonomy for a discrete or continuous visualization design, based on different display attributes. Design spaces for visualizing tree subcategories have also been structured, including composed visualizations [35], [38], timeline-based storytelling visualizations [39], biological data visualizations [40], and the tree visualizations explained in Section 2.1.

Volume rendering results are determined by various design dimensions, such as transfer functions and viewpoints. To search for the volume rendering results that meet one's analysis requirements, users need to explore a design space. Some provide an overview by calculating the differences between the visualization results and arranging them based on MDS projections [14], [41]. More specifically, Design Galleries [14] defines a distance metric within a parameter-based high-dimensional space, to ensure that the options displayed in the gallery differ from each other. In the transfer function map approach [41], a 2D representation of the transfer function feature space is built and the interpolations between the individual volume rendering results are explored. To organize the visual process of exploration for discovery, comparison, and analysis, Jankun-Kelly and Ma [13] propose solutions based on graphs [42] and a spreadsheet interface [13] so as to organize the volume rendering results. Additionally, tools like a palette-style volume visualization interface [43] and the intuitive WYSIWYG interactions [44] have been proposed to make the exploration process more user-friendly. What all these methods have in common is that they are designed to find a suitable parameter set for a given volume dataset. By contrast, Bolte and Bruckner [45] propose Vis-a-Vis to analyze the effect of one parameter set on different datasets with respect to both the graphical output and the source code. However, like volume rendering, generating volumetric geometry also involves a large set of parameters. Hence, Cupid [17] combines the abstract parameter space with the resulting geometric shapes in composite visualizations to help users understand the parameter sensitivities and identify invalid parameter settings.

Another scenario of design space exploration considers chart construction for multivariate and tabular data. The vast combinations of data variables, data transformations, and visual encodings can result in a large design space. Hence, existing studies focus on recommending possible visualizations, deriving insights from prior investigations, and guiding further explorations. To recommend visualizations, Voyager [19] allows users to choose the recommended charts according to statistical and perceptual measures in a mixed-initiative manner. Voyager2 [18] extends Voyager with wildcards and related views to allow open-ended exploration and targeted question answering. All these methods require users to actively participate to find appropriate visualizations, but visualization recommendations are another way of further improving efficiency. One typical example is Draco [46], which uses an optimization technique to find the best visual mapping approaches. Visualizations are specified based on answer set programming and modeling the knowledge from visualization designs as a collection of constraints.

Moreover, narrative visualizations require users to choose an order in which to present multiple visualizations instead of presenting the visualizations as independent individuals. In this vein, Hullman et al. [47] proposed a conceptual framework for identifying possible transitions in a visualization set. Here, the cost of transitions is optimized from the audience's perspective. Kim et al. proposed GraphScape [48], which builds a directed graph model of the visualization design space. GoTreeScape supports automated reasoning about the similarity and ordering of visualizations. Understanding the prior explorations is equally important for deriving insights and guiding further exploration. Chart Constellation [21] summarizes user-generated charts in a 2D space based on the similarities of four elements: chart encoding, keyword tagging, dimensional intersection, and aggregated pairwise. ChartSeer [20] includes a grammar-based encoder-decoder technique that provides a visual summary. However, it emphasizes informing users of the current EVA [49] state based on the charts created. It also decodes charts from the projection results for further exploration based on user interactions. In contrast to ChartSeer, GoTreeScape defines a weighted objective function based on the characteristics of the design features. In addition, it constructs an overview of the tree visualization design space implied by a fine-grained declarative grammar, and includes an exploratory framework to support user's design process.

It is worth noting that all of the above research studies that consider statistical charts are based on Vega-Lite [1], a grammar of graphics capable of expressing a variety of statistical charts. Significant differences exist between GoTree and Vega-Lite in terms of the expressiveness of tree visualizations. For example, at the time of this writing, Vega-Lite does support the authoring of tree visualizations. GoTree, however, is a declarative grammar designed specifically for visualizing tree structures that supports a wide range of tree visualizations.

Compared to existing works, GoTreeScape's point of difference is that it focuses on the design space of tree visualizations, helping users with the exploratory phase of their visualization design.

3 Overview of GoTreeScape

This section discusses the motivating design considerations of GoTreeScape, and then introduces an overview of our methods at a high abstraction level. The technical details are provided in Section 4.

3.1 Design Consideration

In the realm of tree visualization, the purpose of a declarative grammar is to define a design space in a fine-grained manner. Within this design space, tree visualizations can be regarded as combinations of arbitrary attributes from all design features. For example, GoTree [4] considers 49 design features. However, given combinatorial explosion, such a design space will contain an enormous number of visualizations, and browsing them all would imposes a significant cognitive burden on users. Thus, to help designers explore all possible options, we developed a set of considerations informed by the existing principles of exploratory data analysis [49], [50], visualization recommendation [20], and mixed-initiative systems [18], [19]. Moreover, given that not all principles from the above studies apply to tree visualization and not all principles cover the full gamut of what is needed in a tree visualization exploration schema, we also worked closely with visualization designers to refine these design considerations. The final set is summarized as follows:

D1: Show Design Variation Rather Than Data Variation. Design variation refers to the different forms of visually encoding of data, while data variation focuses on the different variables and transformations. In general, exploratory data analysis [18], [19] emphasizes data variation over design variation, while design space exploration pays more attention to design variation. Empirically, visualization designers always determine overall visual representations at first. For example, they decide whether the visual representations are consistent with the topic of their designs. The next step is then the visual encoding of the dataset. Many design features in a visualization grammar, e.g., *node width/height*, have several variations relating to the dataset. The proposed GoTreeScape collapses this space of options to a single tree visualization with default values.

D2: Prefer Fine-Tuning to Exhaustive Enumeration. Despite eliminating the data variation, enumerating the design features still produces a combinatorial explosion. However, not all design attributes have a significant impact on the visualization results; some only have a minor impact on the tree visualization results, such as *padding* between the elements. Other features are numerical, such as the *central angle* of a polar coordinate system, while others still are categorical with symmetrical options, e.g., the *alignment* between a parent and child can be either *left* or *right*. Further, some attribute combinations might be invalid with hierarchical data. Thus, to reduce combinatorial explosion, GoTreeScape rationalizes some features and their combinations and, instead of exhaustively enumerating every option, offers users options to fine-tune their selected visualization.

D3: Provide an Overview of the Tree Visualization Design Space. During exploratory design, users must be kept aware of what has been comprehensively explored and unexplored, and they must continuously determine subsequent exploration directions. An overview of the tree visualization design space provides a visual summary of this and present relationships among tree visualizations. Here, nearby points can be decoded into similar discrete tree visualizations in harmony with human cognition. In this way, such a visual summary benefits the exploratory design process. GoTreeScape considers the relationships between the design features and builds a landscape by employing a VAE technique based on GoTree's grammars in JSON format.

D4: Encourage Interactive Controls to Drive Exploration. Both exploratory design [12] and exploratory data analysis

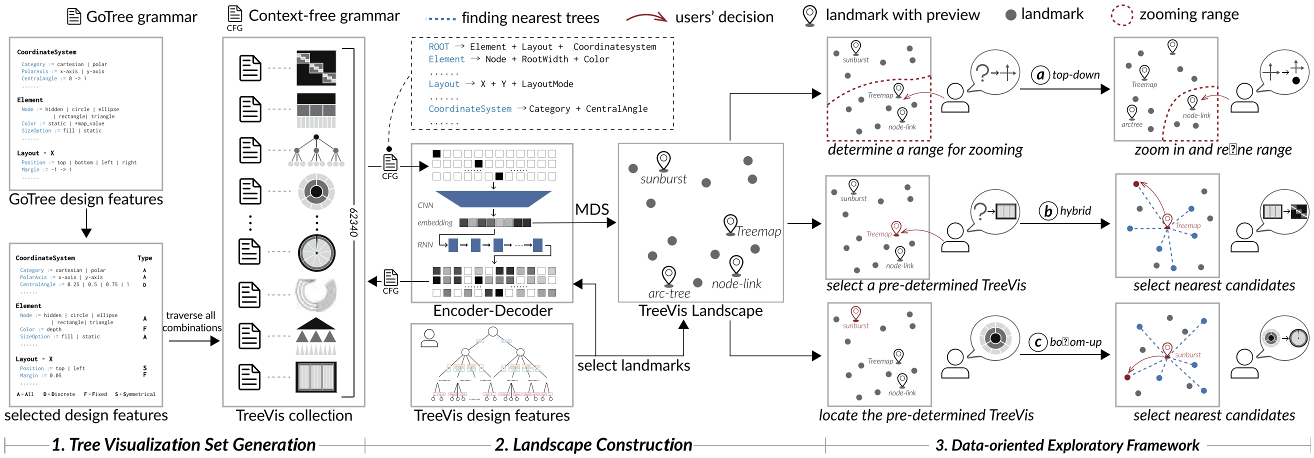

Fig. 2. The pipeline of GoTreeScape comprises three modules; (1) tree visualization set generation, (2) landscape construction, and (3) design space navigation and exploration, which is driven by a framework with top-down, bottom-up and hybrid modes.

are open-ended iterative processes. In the beginning, the user's design goals or analysis tasks may be vague or only loosely specified but, gradually and with exploration, they should become more and more concrete; they might even change completely. During their explorations, users will make decisions, such as determining which direction to pursue further, by assessing their current situation by using their own domain knowledge. To encourage such a dynamic exploration process, the system should always provide users with the interactive controls to indicate their intent and to drive their exploration. To this end, GoTreeScape provides users with density-based contours and landmarks as guides and offers an exploration framework consisting of top-down, bottom-up, and hybrid modes to flexibly adapt to a large range of user requirements.

3.2 System Overview

Guided by the above considerations, we introduce GoTreeScape. GoTreeScape comprises three parts: generating the collection of the tree visualizations; constructing the the design space landscape; and the framework for exploring. Fig. 2 illustrates the overall architecture of the proposed GoTreeScape.

Generating a collection of tree visualizations is the basis of navigating and exploring a design space. The visualization set generation is based on GoTree, which is a declarative grammar of tree visualizations. Compared to GoTree, Vega-Lite supports a wide range of statistical charts but, at the time of writing, cannot be used to author tree visualizations. Traversing all the combinations of the design features defined in GoTree would result in an enormous number of possible tree visualizations in the design space. Therefore, GoTreeScape simplifies the design features in three aspects rather than generating all possibilities. (1) Only combinations of the design features related to design variations are traversed (*D1*); (2) Design features that have a small impact on the final tree visualization results are removed and (3) Invalid combinations of the design features are removed based on domain expertise (*D2*).

Constructing a design space landscape provides users with an overview of the generated tree visualization collection. However, the similarities between the tree visualizations can often be difficult to quantify due to the various design features of the tree visualizations. GoTree decomposes tree visualizations into design features, and so GoTreeScape is based on an encoder-decoder architecture that computes a vector representation for each tree visualization. The decoder computes representations from GoTree in 2D space with a customized objective that considers the characteristics of the tree's layout (*D3*).

The exploratory framework allows users to explore and navigate based on the constructed landscape of the tree visualization design space (*D4*). It consists of top-down, bottom-up, and hybrid modes to account for the varying starting points of each user along with their design decisions during exploration. In top-down mode, users do not have any requirements for their target design, or perhaps they only have partial specifications. Partial specifications allow users to isolate a portion of the landscape from the beginning. As they explore, the correct design features are gradually determined with the help of landmarks. In bottom-up exploratory mode, users have a preliminary tree visualization design but want to explore some other alternatives in the design space. GoTreeScape will therefore recommend visualizations similar to their starting design and also at different levels of zoom. Finally, in hybrid mode, users can flexibly switch between top-down and bottom-up modes. For example, users can decide on a tree visualization during a top-down explorations and then use it as a starting point for a bottom-up exploration.

Based on this exploratory framework, we designed a prototype system to guide users during their exploratory process. The prototype includes density-based contours to inform users of what could be further explored, and also representative landmarks to inform users of the various design features of the tree visualizations. The system also provides a range of interaction options for users to indicate their intentions.

4 GoTreeScape System

This section presents the technical details of each part of the GoTreeScape architecture. Consistent with the system overview explained in Section 3.2, the following subsections introduce how the visualization set is generated, how the landscape is constructed, and the data-oriented exploratory framework. Note that the landscape is constructed independently of the hierarchical data, but the exploratory design framework takes the characteristics of the hierarchical data into consideration. In the last part of this section, we also discuss the design of the prototype.

4.1 Tree Visualization Set Generation

GoTreeScape uses GoTree to represent and manipulate tree visualizations. GoTree divides its 49 design features into three categories: visual elements, the coordinate system, and the layout. Each category consists of multiple fine-grained design features with categorical and numerical attribute values. For example, the attribute values for *NodeShape* and *LinkShape* in the visual element category are categorical, while the value of the *CentralAngle* in the polar coordinate system (within the coordinate system category) is numerical. Details of each design feature can be found in the supplemental material, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TVCG.2022.3215070. Traversing all design feature attributes will result in a massive collection of tree visualizations. So, in GoTreeScape, the visualization set that is generated is simplified against three criteria.

Design Features. GoTreeScape emphasizes design variations rather than data variations (*D1*). As a result, GoTreeScape does not traverse any design features that do not lead to new designs when generating the collection of tree visualizations. The eliminated design features fall into three main categories. The features in the first category, e.g., *NodeWidth*, are only related to the attributes of hierarchical data items. The second category contains features that only have a minor impact on the design, e.g., *Margin* and *Padding* between visual elements. The third category is independent of the visual representations, e.g., the position of *NodeLabel*.

Design Feature Attributes. In addition to the design features, the number of feature attributes is also a significant factor that determines the size of the tree visualization collection. Hence, GoTreeScape also makes the following two simplifications: (1) Any feature attributes that result in symmetrical tree visualizations are simplified. For example, setting the alignment of the parent-child relationship to “left” or “right” results in symmetrical tree visualizations. Therefore, GoTreeScape only takes one option from the symmetric feature values in the collection and leaves the other to a fine-tuning process. (2) Only representative discrete values are taken for the numerical feature attributes. For example, in GoTree, the *CentralAngle* of the polar coordinate system falls between 0 and 1. Zero indicates that the central angle of the polar coordinate system is $0°$, and one indicates that the central angle is $360°$. We set the central angles to 0.25 ($90°$), 0.5 ($180°$), 0.75 ($270°$), and 1 ($360°$) when generating the tree visualization collection.

Combinations of Design Feature Attributes. Some design feature combinations lead to invalid visual representations with hierarchical data due to severe overlaps with the visual elements or conflicts between the design features. Two typical examples follow: (1) When the relative positions between siblings along the x-axis and y-axis are both *aligned*, the nodes in the visualization overlap significantly making it difficult to differentiate them. (2) Some visual elements, such as, ellipses and triangles can conflict with the features of the layout and coordinate system. That is, the position and occupied space of the visual elements are calculated based on the layout and coordinate system, but these nodes cannot be appropriately visualized in the occupied space.

The design space can be represented using a hierarchical structure. The above three simplifications over the whole design space are shown in Fig. 7a, and the remaining design features and attributes are shown in Fig. 7b. After simplification, 62,340 tree visualizations remain in the collection. Details of the simplified configurations are given in the supplemental materials, available online.

4.2 Landscape Construction

To provide an overview of design space that shows a visual summary of the relationships between tree visualizations, an unsupervised encoder-decoder framework converts the tree visualizations to and from embedding vectors in the latent space. Specifically, the encoder maps the input samples to vectors in a low dimensional latent space, and then the decoder restores the vectors to the original space. Essentially, these embeddings constitute a representation of the target tree visualization design space.

The similarities between the tree visualizations can easily be measured based on the euclidean distance between the vectors of these representations. Furthermore, the relationships, clusters, and distribution of the tree visualization design space can be also derived from this latent space. To improve the readability of the landscape, the latent space is eventually reduced to a two-dimensional euclidean space, and landmarks are added to the landscape to guide the user’s exploration.

Our insights into the landscape design, which serve as the domain knowledge for the model design, are discussed next. We then discuss the VAE, highlighting its advantages over other dimensionality reduction techniques. Finally, we present more details on how the GoTree-based landscape is constructed.

4.2.1 Landscape Design Justification

One of the jobs of the overview is to help users learn the relative relationships between visualizations. One straightforward approach to accomplishing this goal is to directly display all tree visualization items and to use the distances between the items to encode their similarities. However, the underlying visualization set for constructing such an overview is too large for such a direct solution. Showing all visualizations in the collection would severely overwhelm users. Note that visualization design space exploration is different from recommendation, which is able to get a priority of different visualizations. For example, Draco [46] sorts the visualizations according to some criteria and shows them in a simple list. However, showing all tree visualizations without priority in a simple list is likely not optimal because users would need to check each tree visualization, making the exploration process tedious and time-consuming. To solve these challenges, we have turned to the visual and interactive properties of a landscape as a metaphor. More specifically, image looking at a map from a zoomed-out point of view where only representative landmarks, such as

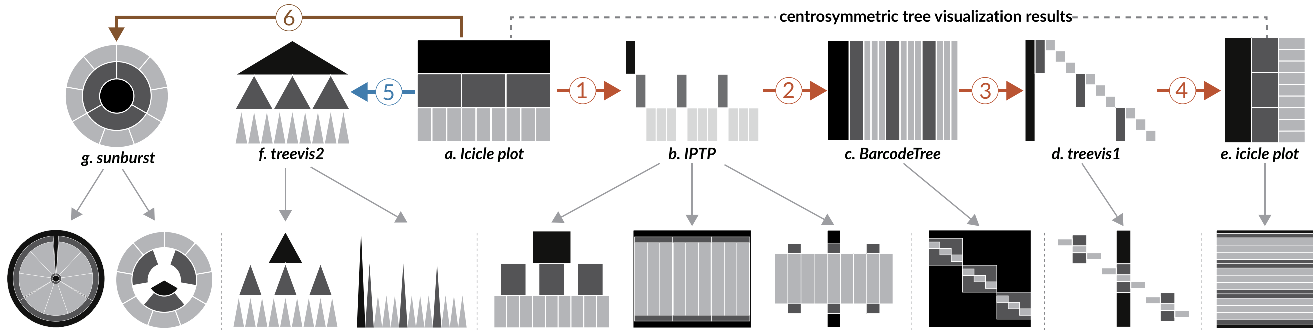

Fig. 3. The six tree visualizations. (b)-(g) in the first row are based on (a) an icicle plot by changing the layout-related design features (arrows in orange), the visual element-related design features (arrows in blue), and the coordinate system-related design features (arrows in brown). (1) Changing parent-child relation along the *x* axis to *juxtapose*. (2) Changing the parent-child relation along the *y* axis to *align*. (3) Changing the sibling relation along the *y* axis to *flatten*. (4) Changing the sibling relation along the *x* axis to *align*. (5) Changing the visual elements to *triangle*. (6) Changing coordinate system to *polar*. The visualizations in the second line are based on the results in the first row accordingly.

countries and their capitals, are visible. Then zoom in, and the inner states of a country begin to appear. In addition, when the user selects a specific target of interest, more targets belonging to the same category will be appear on the landscape. We propose an exploration tool that allows users to navigate the collection of tree visualizations in a similar way. A landscape was chosen as a visual metaphor for two reasons. First, the landscape metaphor was one of the first methods used by the information visualization community to visualize rich information that is not inherently spatial [51]. Second, existing studies have found that everyone intuitively understands landscapes [51] and generally learns to read maps in pre-school [52]. In addition, solving map-based analysis tasks requires little training. The remainder of this section introduces the specific techniques for building GoTreeScape. These techniques are invisible to ordinary users; all users need to do is to interact with the landscape overview.

4.2.2 Insight on Design Feature

This section explains our insights into the design features, which guided us in determining the weights of each feature when training the autoencoder. The model is used to map the tree visualizations to latent vectors. The design features in GoTree determine tree visualization results. However, by investigating the generated tree visualization collection, as explained in Section 4.1, we found that computing similarities between the tree visualizations based on euclidean distance was not consistent with human perception for the following two reasons:

First, different design features have a different magnitude of impact on the tree visualization results. In fact, we classified design features into four different categories according to the impact they have on the results. The design features associated with the coordinate system have the most significant impact. As shown in Fig. 3(6), changing the coordinate system attribute value from *Cartesian* to the *polar* influences the layouts (relative positions) of the tree visualizations. Since position is the most efficient visual channel for encoding data, layout-related design features have the second-most significant impact. As shown in Fig. 3(1)-(4), layout-related design features influence the tree visualization layout, including relative position and height/width of the visual elements. The third category consists of visual element-related design features (Fig. 3(5)). Features in this category only change the visual elements. In the fourth category, the design features only slightly adjust the layout. These features include attributes such as margins and paddings between the nodes.

Second, the similarities between tree visualizations do not necessarily correlate to the number of design features that have changed. GoTree is a declarative grammar defined along axes, which is a common way to design visualization grammars, such as ATOM [3] and Vega-lite [2]. However, with an axis-decomposed declarative grammar, the layout-related design features of two center-symmetric tree visualizations may be completely different. Fig. 3 shows an example. Starting from the icicle plot tree visualization in (a), the parent-child relation along the x axis is *include* and the sibling relation is *flatten*; along the y axis, the parent-child relation is *juxtapose* and the sibling relation is *align*). However, after swapping the design features along the x and y axis (Steps 1-4), the tree visualization changes to that shown in (e). Both (a) and (e) are icicle plot tree visualizations only with different orientations. However, the edit distance in terms of the design features of the two grammars is great.

Based on the above insights, we restructured the design space as shown in Fig. 7c, reordering the design features according to the level of impact they have on the tree visualizations. Further, the layout-related design features are grouped along the same axis. The weights of these design features decrease from top to bottom and are encoded in a vector W that is used to train an autoencoder, as described in Section 4.2.4.

4.2.3 VAE-Based Dimensionality Reduction

Landscape construction can be modeled as a dimensionality reduction task, which maps the tree visualization design space from a discrete space into a low-dimensional euclidean space. This section introduces the basis of the VAE, highlighting its advantages over other dimensionality reduction techniques. Unlike other traditional dimensionality reduction techniques, e.g., MDS [53], PCA [54], UMAP [55] and t-SNE [56], VAEs [57] are a type of generative model with a strong ability to represent data. VAEs assume that the input data has some sort of underlying probability distribution, such as a Gaussian distribution, and it projects data into the latent space in a generative modeling way.

Based on an encoder-decoder framework, the VAE uses variational inference to derive an evidence lower bound (ELBO) as the objective [57] given by:

$$\log p(x_i) \geq E_{\sim q_\theta(z|x_i)}[\log p_\phi(x_i|z)] - D_{KL}(q_\theta(z|x_i)||p(z)) \tag{1}$$

Note that the right-hand side of Eq. (1) is the core of the VAE, where $q_\theta(z|x_i)$ is the encoder that maps x_i into the latent variable z, and $p_\phi(x_i|z)$ is the decoder that reconstruct z from the input x_i. The first term in the ELBO represents the reconstruction log-likelihood, while the Kullback-Leibler (KL) term ensures the learned distribution. $q_\theta(z|x_i)$ is similar to the true prior distribution $p(z)$. Notably, the KL term reveals a fundamentally unique property that separates it from an ordinary autoencoder, that is, that the VAE not only reconstructs the inputs, it also learns a more coherent latent space in which nearby points decode to similar discrete outputs.

A Gaussian representation was chosen for the latent prior $p(z)$ and the approximate posterior $q_\theta(z|x_i)$ empirically. Finally, the VAE loss function is derived by considering the negative of the ELBO:

$$\mathcal{L} = D_{KL}(q_\theta(z|x_i)||\mathcal{N}(0,1)) - E_{\sim q_\theta(z|x_i)}[\log p_\phi(x_i|z)] \tag{2}$$

where the optimal parameters (θ^*, ϕ^*) are derived by minimizing $\mathcal{L}$:

$$(\theta^*, \phi^*) = argmin_{\theta,\phi}\mathcal{L}(\theta, \phi) \tag{3}$$

Thus, the GoTree-based landscape is constructed by customizing the neural network structure and the objective based on this VAE methodology.

4.2.4 GoTree-based Landscape Construction

The landscape construction approach is based on the declarative grammar of the tree visualizations. Generally, visualization images in bitmap format are the final results that users directly perceive. Therefore, the smaller the pixel-based distance between two bitmap images, the more similar the corresponding tree visualization results should be. However, after testing the landscape construction method based on tree visualization images, we found that the GoTree grammar captures inherent visualization features like radial versus angular or include versus juxtapose, and these would have to be tediously extracted from the resulting bitmaps using computer vision technique. Therefore, the landscape construction based on these grammatic expressions aligns by design with these features, breaking down the landscape into coherent and sensible regions implied by them – e.g., a region of radial visualizations versus a region of angular visualizations. Detailed results and explanations can be found in the supplemental material, available online.

Compared to the bitmap images, GoTree, which decomposes tree visualizations into design features, is a better input format and means that domain knowledge can be injected into landscape construction results. To input the grammar into the model, we used context-free grammar (CFG) inspired by the grammar-based variational autoencoder (GVAE) [58]. This linearizes the GoTree's grammar by mapping it into a set of rules, as shown in the dotted box in Fig. 2. The input/output of the encoder-decoder network is a set of one-hot feature vectors representing the rules extracted from CFG rules of the tree visualizations. Since the different tree visualizations do not have an identical number of rules, we carefully designed the feature vector to ensure that different tree visualizations share the same structure. As shown in Fig. 4, the length of its first dimension is the maximum number of rules extracted from individual tree visualizations' CFG, defined as m (35 in GoTreeScape). The second dimension indicates a padding rule (*Nothing→None*) and the deduplicated rules extracted from all tree visualizations' CFG, and its length is defined as n (60 in GoTreeScape). The GVAE model's structure is then refined based on the above tree visualization features, and a weighted reconstruction loss is introduced.

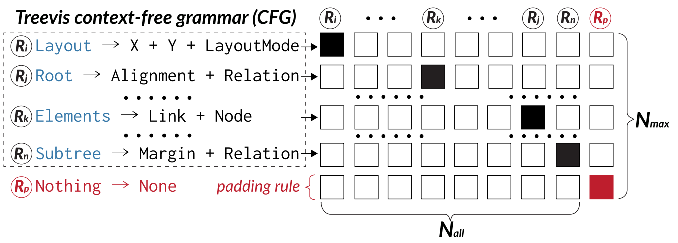

Fig. 4. The input to the encoder-decoder network transforms the rules into one-hot feature vectors. N_{all} indicates the number of rules extracted from the CFG of all tree visualizations. N_{max} indicates the maximum number of rules extracted from each CFG tree visualization.

$$\mathcal{L}_r = \frac{1}{n}\sum_{i=1}^{n} W^T \cdot \|(p_\phi(q_\theta(x_i)) - x_i)\|_2^2, \tag{4}$$

where $x_i \in \mathbb{R}^{n \times m}$ is the ith parsed tree in the generated training visualization set. Each rule is represented as a $n \times 1$ one-hot embedding and p_ϕ is an RNN decoder based on a GRU. Considering the repetitive and translationally invariant property of the input CFG strings, q_θ is designed as a 1D-CNN encoder, while W is an $n \times 1$ normalized weight vector, and W_i denotes the heuristic weight of the ith design feature given the design feature insights discussed in Section 4.2.2. The weights for generating the design space overview in Fig. 5 are 10000 for the coordinate system-related design features, 100 for the layout-related design features, and 1 for the visual element-related design features. With the help of the prior weight vector, the domain expertise concerning the importance of the design features can be preserved into the embeddings of the latent space. To enable users to explore and navigate the design space, the embedding results must be visualized in two-dimensional space. There are several ways this can be done. The first option is to learn a 20-dimensional latent space with the GVAE model and then project the embeddings in two-dimensional space using dimension reduction techniques. The other alternative is to learn a two-dimensional latent space directly, but this would have a lower accuracy, as shown in Table 1.

Fig. 5 shows the overview of tree visualization design space based on different methods. The first three columns are the projection results of the 20-dimensional latent vectors using the MDS [53], UMAP [55], and t-SNE [56] dimension reduction techniques. In terms of the parameters of the

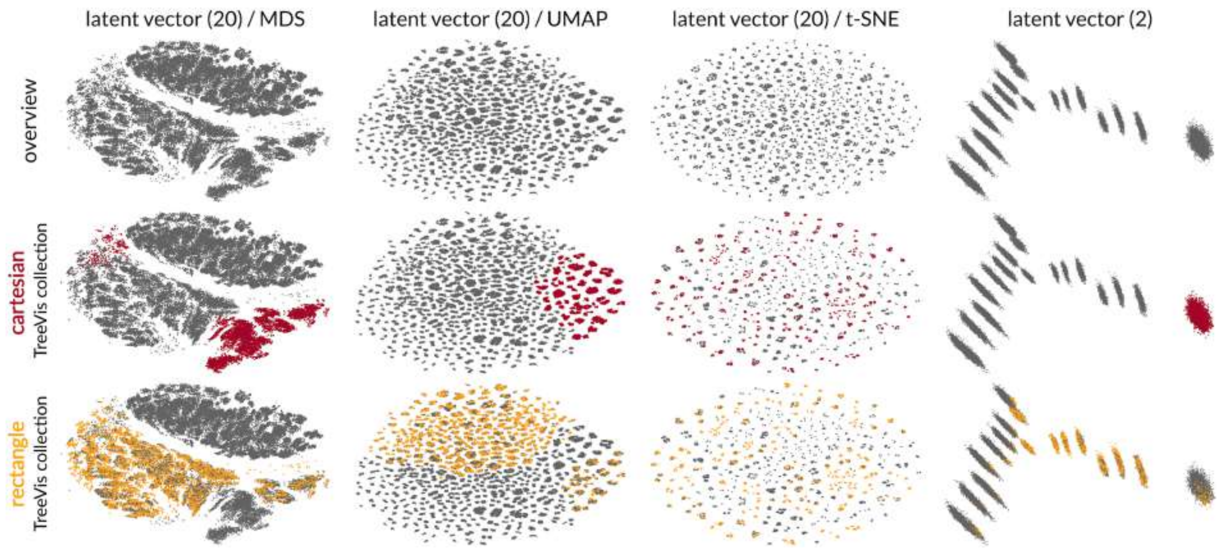

Fig. 5. Comparison of the projection results. Vertically, each of the four columns refer to the projection results derived from different techniques. The first three columns reflect 20-dimensional latent vectors, as computed by a grammar-based auto-encoder and projected them to 2D space using MDS, UMAP and t-SNE, respectively. The fourth column reflects 2-dimensional latent vectors. Horizontally, the projection results in the first row show the overview of tree visualization design space implied by GoTree. The second row highlights tree visualizations in the *Cartesian* coordinate system in red, and the third row highlights tree visualizations with *rectangular* visual element in orange.

UMAP technique, we set the *n_neighbors* (the number of neighbors) to 50 and the *min_dist* to 0.5. For t-SNE's parameters, we set the *perplexity* to 50. The fourth column shows the visualization results of the two-dimensional embeddings.

The second row and third row in Fig. 5 illustrate the tree visualizations with the Cartesian coordinate system and rectangular visual elements, respectively. The results show that the projection results with MDS and UMAP using a 20-dimensional latent space and a two-dimensional latent space (the first, second, and fourth columns) can better preserve the local feature characteristics. For example, most tree visualizations in the Cartesian coordinate system or with rectangular visual elements are adjacent in the landscape. However, the projection results for t-SNE (the third column) are not, because t-SNE performs much worse at preserving the global structure [59]. To evaluate the quality of the dimensionality reduction, we use a Jaccard index [60], which measures the dissimilarity between sample sets. First, we performed hierarchical clustering for tree visualizations in the landscape, with each cluster in the hierarchical clustering results (H) being denoted as c_i. Second, we extracted multiple tree visualization lists (denoted as l_k) by filtering some chosen design features (e.g., the Cartesian coordinate system and the rectangular visual elements). Then, the maximum Jaccard index for all clusters in the hierarchical clustering results is computed for each tree visualization list.

$$J_k = \max_{i=1,\ldots,n} \frac{|l_k \cap c_i|}{|l_k \cup c_i|}, \quad c_1, c_2, \ldots c_n \in H \tag{5}$$

Table 2 presents the results of 11 relatively important design features related to visual elements, the coordinate system and the layout. The full table can be found in the supplemental material, available online. From the results, we can see that the MDS, UMAP, and two-dimensional embedding results have larger values. Considering the accuracy of the GVAE model (Table 1) and the non-deterministic characteristic of UMAP techniques, GoTreeScape finally employs MDS projection method to compute the overview of tree visualization design space.

TABLE 1
Comparison of GVAE Autoencoder Accuracy

GVAE latent dimension	2	20	100
Accuracy	0.619	0.904	0.9292

4.2.5 Landscape Visual Guidance

For users to understand where they are situated in the constructed landscape, and to be able to decide on where to explore next, they need visual guidance. One such indicator provide in GoTreeScape is density-based contours, which show users the distributions of tree visualizations. The second is the representative tree visualizations across the landscape, which help users to understand whether optional tree visualizations within a certain range will meet their requirements. The third is the boundaries between tree visualization clusters, making the top-level structure visually distinctive. From these, users can decide whether to continue exploring at a finer granularity. Given that the above landscape construction

TABLE 2
Comparison of Dimensionality Reduction Techniques

design features	Latent(20) MDS	Latent(20) UMAP	Latent(20) t-SNE	Latent(2)
cartesian	0.89	0.78	0.50	0.91
polar	0.57	0.49	0.48	0.92
rectangle	0.51	0.44	0.41	0.64
circle	0.68	0.49	0.26	0.32
triangle	0.28	0.22	0.13	0.29
ellipse	0.22	0.15	0.07	0.23
y: include	0.23	0.15	0.16	0.17
y: juxtapose	0.43	0.40	0.26	0.49
x: within	0.31	0.30	0.28	0.34
x: align	0.33	0.33	0.24	0.47
x: flatten	0.44	0.40	0.29	0.48

method is designed to keep neighboring items relatively similar to the user's perception, it becomes possible to only show representative landmarks instead of each tree visualization in detail. To show these landmarks at different levels of zoom, hierarchical clustering is performed on the collection of tree visualizations and representative items from the clusters are computed at different clustering levels. Furthermore, GoTreeScape computes the cluster boundaries based on clustering centers using a Voronoi diagram [61]. Fig. 6 presents the landscape with the above three visual guides. According to Ceneda's [62] conceptual guidance framework, our visual guidances *orient* users towards regions that are worthwhile zooming into. This addresses the knowledge gaps pertaining to the target being unknown.

Algorithm 1. Landmark Selection Algorithm

Require:
- T indicates a tree visualization collection.
- F indicates the design feature list, which consists of tree visualization design features and each feature is denoted as f_i.
- W indicates the design feature weight list, which consists of the corresponding weight w_i of each design feature f_i.
- n indicate the amount number of the selected landmarks.

Ensure: - The selected landmark list L of T.

construct design feature hierarchy H based on F, each node h_i of H contains a design feature f_i. h_r denotes the root of H.
reorganize design feature hierarchy H according to W.
for $t \in T$ **do**
  Count_Feature(h_r, t) ▷traverse H and count for each t
end for
for $h_i \in$ Bottom_Up_Traversal(H) **do**
  Compute_Representative(h_i, n)
end for
$L \leftarrow h_r$.R[n] ▷R[n] of h_r is the selected landmarks
function Compute_Representativeh_i, n
  S_r indicates a selected representative tree. $S_r[i][j]$ indicates selected results from the first i children of h_i.
  S_i indicates the importance of corresponding representative trees in S_r.
  for $i \in (0, h_i$.children.length) **do**
    for $j \in (0, n+1)$ **do**
      k^*=arg max$_{\forall k \in (0,j)}$ S_i[i-1][j-k]
      + h_i.children[i].I[k] + $\sum_{l=j-k}^{j} \frac{w}{l} \times h_i.count$
      $S_i[i][j]$ = S_i[i-1][j-k^*] + h_i.children[i].I[k^*]
      + $\sum_{l=j-k^*}^{j} \frac{w}{l} \times h_i.count$
      $S_r[i][j]$ = S_r[i-1][j-k^*] + h_i.children[i].R[k^*]
    end for
  end for
  h_i.I = S_r[-1] ▷the last row
  h_i.R = S_i[-1] ▷the last row
end Function
function Count_Feature$h$, t
  h indicates a node of design feature hierarchy H.
  t indicates a tree visualization declarative grammar.
  if t.Match_Design_Feature(h) **then**
    $h.count$+=1
    for $h_{child} \in h$.children **do**
      Count_Feature(h_{child}, t)
    end for
  end if
end Function

Fig. 6. GoTreeScape with three visual guidance: density-based contour, representative tree visualizations, and boundaries between tree visualization clusters.

With the help of a declarative grammar, tree visualizations can be thought of as a combination of different design feature attributes. Therefore, the representative tree visualizations selected for display should span as many attributes and combinations of attributes as possible. Furthermore, different attribute values will have a different number of associated tree visualizations. Taking the *CoordinateSystem* attribute as an example, many fewer tree visualizations are associated with the value *Cartesian* than the value *polar*. This is because the polar coordinate system comes with many fine-grained design features, such as *PolarAxis* and *CentralAngle*. Therefore, if a random sampling method were to be used, the majority of the representative tree visualizations selected would be based on a polar coordinate system. Additionally, a random sampling technique assumes that each design feature has the same magnitude of impact on the tree visualization results. However, the opposite is true, as explained in Section 4.2.2. To fill this gap, we designed a dynamic programming algorithm (Algorithm 1) to select the most representative tree visualizations. The inputs to the algorithm are the collection of tree visualizations (T), the design feature list (F) with the corresponding weights (W) of the design features, and the number of the representative tree visualizations that should be selected (n). The algorithm then proceeds through the following four steps: (1) Construct a hierarchical data H for the design features based on F, where each node h of the hierarchy contains a design feature and a specific attribute value. (2) Reorganize the design feature hierarchy according to W. Initially, the design features are arranged in descending order of weight from heaviest to lightest. But, to avoid selecting symmetrical tree visualizations for the representative list, the hierarchy groups the layout-related design features along the horizontal and vertical axis together. The reorganized design feature hierarchy is shown in Fig. 7; (3) Compute the number of tree visualizations associated with different design features in the hierarchy, denoted as $h.count$, which is an important factor for computing the $importance$ of the representative items. (4) Traverse the hierarchy in a bottom-up manner and select the representative tree visualizations. For each node h, the dynamic programming algorithm defines the state $S_i[i][j]$ as the $importance$ of selecting the j most representative tree visualizations from the first i children, while $S_r[i][j]$ is defined as the selected tree visualization results corresponding to $S_i[i][j]$. Selecting one tree visualization as being representative of a design feature has a positive correlation with both the

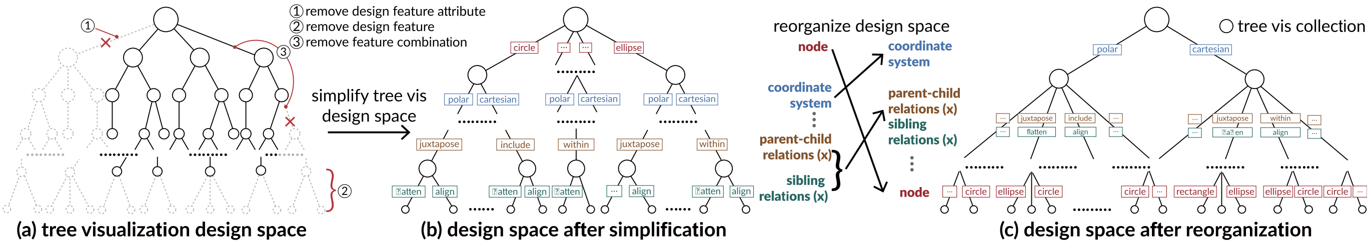

Fig. 7. The simplification and restructuring of the tree visualization design space. Three hierarchies represent the design space of tree visualizations. Within the hierarchy, each row refers to one design feature and each link indicates one specific design feature attribute. The left hierarchy displays the simplification for the whole design space, the middle one shows the remaining design features and the attributes of the design space after simplification, and the right one arranges the design features according to human perception.

$weight$ of the design features and the number of tree visualizations related to that design feature. It also has a negative correlation with the number of already-selected representative items l. As a result, the state transfer function is defined as follows:

$$S_i[i][j] = S_i[i-1][j-k] + h.children[i].I[k] + \sum_{l=j-k}^{j} \frac{w}{l} \times h.count \tag{6}$$

The number of design features in F as is defined as f, and the complexity of the algorithm for selecting representative tree visualizations is $\mathrm{O}(nf^2)$.

4.3 Data-Oriented Exploratory Framework

The *data-oriented* exploratory framework helps users to find the appropriate tree visualizations. As explained in Section 4.2, embeddings are learned from the visualization specifications, independent of any particular hierarchical data. However, various features of the hierarchical data are critical for determining tree visualizations, such as deep/shallow, large/small, balanced/unbalanced, and regular/irregular. As a result, GoTreeScape has to load the targeted hierarchical data and generate all visualization results based on them when presenting the constructed landscape to users. As explained in Section 4.2.5, the landscape provides a large number of tree visualization previews to guide users' explorations. However, too much hierarchical data will impose a huge rendering burden on the system. Additionally, the small display space of the preview panel will not be able to accommodate the visualization results. To solve this problem, GoTreeScape calculates a new derived attribute called the Strahler Number [63] for each node termed the Strahler number. The Strahler number serves as a measure of a node's importance according to the topological structure of the hierarchical data. Specifically, the central nodes have large values, while the peripheral nodes have low values. This means the complex hierarchical data at two different abstraction levels. The more simplified one serves as the underlying data of selected landmarks on the landscape. The other one is used as the underlying data for the preview panel. The benefit of this method is that the simplified results retain the key characteristics of the topological structure. Fig. 8 shows the sampling results from the Flare package[2] structure with different thresholds. With this sampling method, GoTreeScape allows the data-oriented exploration by showing the simplified hierarchical data visualization results in the preview panel instead of the original hierarchical data.

With the help of representative landmarks displayed on the landscape, users should be able to continuously make design decisions based on the results and accordingly provide feedback as input to interactively control the exploratory design process. The information determined by users about target tree visualizations in different application scenarios have significant differences. Before exploring the design space, users may not have any explicit requirements in mind. Alternately, they may have some loose ideas about design features, such as "the tree visualization should contain circular elements". Last, they may have a very fixed idea about the tree visualization type, e.g., it should be a "node-link diagram". To address each user's various requirements, GoTreeScape includes an exploratory framework that offers users three different exploration modes: top-down, bottom-up and hybrid exploration, as shown in Fig. 10. Note that these three exploration modes do not refer to users' patterns of zooming-in and out. They are motivated by the sensemaking models [64] from visual query systems. The top-down process is goal-oriented, where users gradually determine specific design features to concretize the target visualizations in their minds. By contrast, the bottom-up process is data-driven and initiated by a pre-determined tree visualization. Here, the GoTreeScape system "recommends" other tree visualizations as "stimuli" to drive users' explorations. In particular, these recommendations are not driven by a recommendation system in the data science sense of the word. Users still need to make navigation decisions as they move through a series of tree visualization landmarks in the landscape during exploration.

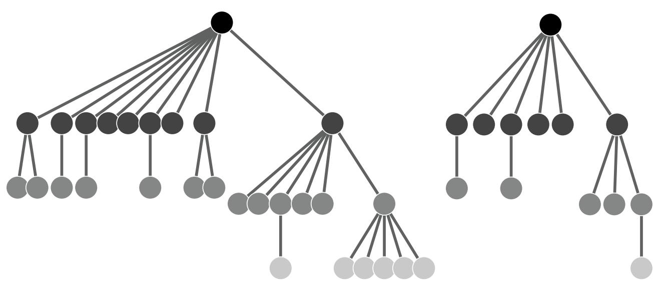

Fig. 8. The sampling results of the Flare package structure. There tare 258 nodes in the hierarchical data. (a) The threshold of the Strahler number is 3 and, after sampling, 29 nodes remain. (b) The threshold is 8 and the number of nodes after sampling is 13.

2. github.com/d3/d3-hierarchy/blob/main/test/data/flare.json

4.3.1 Top-down Mode

When users do not have any explicit requirements as they embark on building a tree visualization, their target visualizations might exist in any part of the entire landscape. This exploratory mode is defined as top-down and spans the entire design space as a starting point. As explained in Section 4.2.5, the landscape displays a series of tree visualization landmarks. Users can compare these tree visualization landmarks and then determine the design features they prefer to make their target more concrete. These landmarks displayed on the landscape also help users determine the range of the target tree visualizations, from which users can narrow down their scope of exploration. As the scope narrows and focuses, GoTreeScape displays more fine-grained landmarks to help users make further design decisions. By repeating the above exploration process, users eventually locate their target visualizations on the landscape. The red arrows in Fig. 10 show the users' exploration path in the top-down mode. Top-down exploration mode also supports the application scenarios where users only have partial design requirements. The difference is that the starting point of the exploration process is filtered to only show that part of the design space that meets the user's predetermined requirements instead of the entire landscape.

4.3.2 Bottom-up Mode

Prior to exploratory design, users might already have a predetermined tree visualization (denoted as p) that corresponds to a specific item on the landscape. Here, the user's goal is exploring whether there might be other tree visualizations that are more appropriate than the one in mind. In this exploration mode, users start with a pre-determined tree visualization and gradually expand their scope to the entire landscape. This is a bottom-up exploration process.

The blue arrows in Fig. 10 show the user's exploration path in bottom-up mode. The path consists of the two steps—The first is to locate p on the landscape. Here, the encoder module outlined in Section 4.2.4 transforms the user's input into a vector in the latent space. The item's position on the landscape is computed using the interpolation method used by Chartseer [20]. The second step is to find a collection of related tree visualizations from the landscape to help the user determine whether a tree visualization other than p better meets their requirements. To this end, GoTreeScape displays the top k tree visualizations that are most similar to p but from different clusters. As shown in Fig. 10, users can adjust the level interactively by selecting related tree visualizations. Hence, a tree visualization selected from the lower level could be more similar to p.

4.3.3 Hybrid Mode

The top-down and bottom-up modes are not isolated. Rather, users can flexibly switch between two different modes within their exploratory design process. For example, one user might start in top-down exploration mode and then once s/he finds a satisfying tree visualization, switches to the bottom-up mode to locate similar visualizations in the landscape at different levels as shown by the green arrows in Fig. 10). Alternatively, a user might have a tree visualization in mind and locate it in bottom-up mode. But when they find a tree visualization that does more to satisfy their requirements, they may switch to top-down exploration mode to for a more comprehensive exploration of the neighborhood.

4.4 User Interface and Interaction

Guided by all the above considerations, we designed a prototype system of GoTreeScape. The user interface consists of five interactively coordinated views. The main view is the landscape panel (Fig. 10a), which shows an overview of the tree visualization design space augmented by a small "bird's-eye" view as an orienting tool. A small rectangle within the overview shows the region viewable within the landscape. Visual guidance on the landscape consists of density-based contours and representative landmarks. After users upload their hierarchical data, GoTreeScape simplifies the data used to display the landmarks and the preview panel. The system further selects some landmarks to display in the corresponding visualization results while mapping other landmarks to circles. To help users make decisions, they can click on a landmark, which will show the visualization results in the preview panel (Fig. 10b). The right side of the preview panel provides a series of operations for the selected tree visualization, including switching to bottom-up mode based on the visualization, saving the visualization into the gallery, opening the visualization in Tree Illustrator, and checking the related tree visualizations after fine-tuning the parameters.

Users can flexibly adjust the displayed range of the landscape to suit their requirements. In top-down mode, users can decide the range of subsequent explorations interactively according to the landmarks. Additionally, users can zoom in to show the tree visualizations at a finer granularity or zoom out to change the determined design dimension. Our interface also supports users to filter for the exploratory design on the landscape panel. For example, the view will be updated according to the provided input query in Fig. 10e. Bottom-up mode includes a data uploading panel (Fig. 10d) that allows users to upload a tree visualization of GoTree grammar in JSON format. GoTreeScape also provides users with a collection of classic tree visualizations, as shown in Fig. 10c.

4.5 Implementation

GoTreeScape comprises a back-end exploration engine and a front-end user interface, with both being based on a pre-trained auto-encoder. The deep learning model was built using Tensorflow. Dimension reduction is handled by MDS [53], and the hierarchical clustering method used is from the Sklearn Python library. For the parameters of hierarchical clustering method, we set the distance metric as *euclidean* and the linkage criterion as *ward*. The front-end user interface uses D3 [9] based on scalable vector graphics (SVG). Specifically, we used the library provided by GoTree [4] to visualize the different trees. The source code for GoTreeScape is available at GitHub[3].

5 Case Studies

To demonstrate the effectiveness and usefulness of GoTreeScape, we invited one visualization researcher (VR) and one

3. https://github.com/bitvis/gotreescape

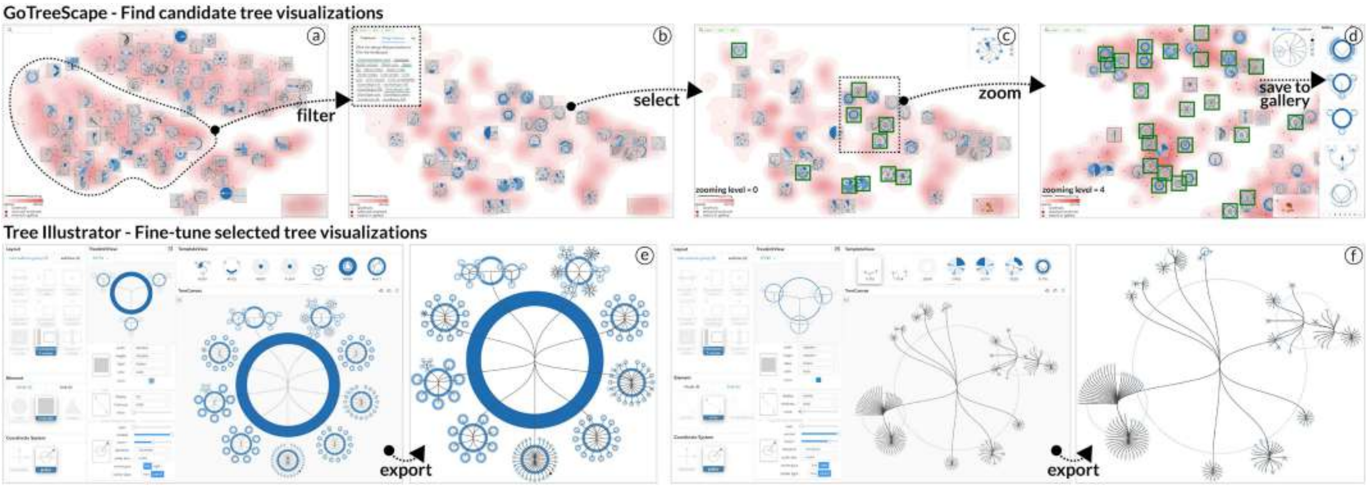

Fig. 9. The top-down exploration mode in the tree visualization design space. The top row shows the process of finding the candidate tree visualizations using GoTreeScape. (a) shows the entire landscape of the tree visualization design space. (b) shows the remaining landscape after filtering the tree visualizations to only show trees with rectangular nodes and using the polar coordinate system. (c) selecting the tree visualizations that meet a user's requirements. (d) zooming in on the lower levels of the landscape to select more tree visualizations. The selected tree visualizations are saved in a gallery. The bottom row shows the process of fine-tuning the selected tree visualizations. (e) and (f) show two fine-tuned visualization results from the user's uploading hierarchical data.

visualization designer (VD), who each had two to four years' experience in designing visualizations and visual analytic systems. They were given a brief introduction on how to use the prototype system, and talked through the interface designs and system functionalities. We then asked them to apply GoTreeScape into their own tree visualization design scenario. This section presents the workflows from these two use cases and concludes with the users' feedback on the system.

5.1 Case 1: Top-down mode

Our first user, VD, is a visualization designer that does not have a programming background. He needed to design a tree visualization to illustrate the reposting process in social media. A reposting tree is typical example of hierarchical data, where a node represents a message, and a link represents a repost. Further, this hierarchical data contained much information. For example, each node contained information about the reposted messages, such as its content and emotional attitude of the poster, as well as information about the authors, including their age, gender, and location.

VD's predetermined requirements for the visualization were that it should have a circular shape and be able to encode several attribute values alongside the nodes and links. This requirement meant he could filter out tree visualizations based on the Cartesian coordinate system (because they do not have a circular shape), and any tree visualizations with hidden nodes (because they cannot encode attribute values into the visual elements). The first step VD took was to upload his hierarchical data into the GoTreeScape system. The data had a depth of 5 and 264 nodes. At this point, the landscape showed many visualization previews, the underlying data of which is the hierarchical data after sampling based on the computation of Strahler number, as explained in Section 4.3. VD therefore adjusted the number and specific items of the tree visualization previews to be displayed on the landscape. From this, he learned that the tree visualizations with rectangular nodes had many design variations, so he filtered the landscape to only show tree visualizations with rectangular visual elements (see Fig. 9a). Next, he began to explore the remaining landscape (see Fig. 9b). He identified many tree visualizations that meet his requirements, saving each as he came across them to the gallery (see Fig. 9c). He continued to zoom into the landscape from the top level to the bottom level, putting any tree visualizations of particular interest in the center to check for additional related results (see Fig. 9d). Ultimately, VD decided on a candidate tree visualization collection that meets his initial requirements—a circular shape with rectangular nodes. VD perused his selected tree visualizations in the gallery and then switched to Tree Illustrator to fine-tune the results, as shown in the bottom row of Fig. 4.3. GoTreeScape helped VD to determine the parent-centric tree visualizations because each message during the reposting process needs to be analyzed as a center for

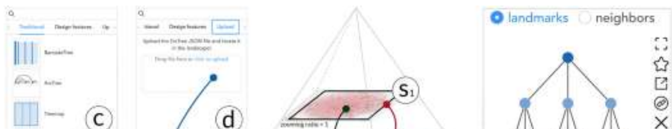

Fig. 10. GoTreeScape's exploratory framework. The framework consists of three modes, top-down (arrows in red), bottom-up (arrows in blue), and hybrid (arrows in green). GoTreeScape contains five panels, (a) the landscape panel, (b) the preview panel, (c) the traditional tree visualization panel, (d) the uploading panel, and (e) the filtering panel.

comparison. Figs. 9e and 9f show two circular-shaped tree visualization results. The difference is that the left one emphasizes the topology, because the subtrees are the same size, while the right one emphasizes the attribute values (the sizes of the subtrees are relative to their width).

Satisfied with his selected tree visualizations and is also inspired by the visualization results during the exploration process. VD mentioned that he planned to use the size of subtrees' circles to encode the underlying messages' impacts. Additionally, he would color the circles to encode positive or negative attitudes and arrange the subtrees in a clockwise direction according to the time sequence of the reposting behaviors.

5.2 Case 2: Bottom-up mode

Our second use case shows how GoTreeScape can guide users to explore novel tree visualization designs. VR mentioned that he always designs tree visualizations based on a collection of alternative options and further explores the design space according to different application scenarios. More specifically, when designing visual analytic systems, he would like a novel tree visualization technique as opposed to just applying known tree visualizations directly because existing tree visualizations are often not applicable to a specific problem at hand. The method therefore places novelty as a priority. To achieve this task, VR reproduced some of the existing tree visualizations in treevis.net [8] using GoTree. He then located and marked them on the GoTreeScape. Fig. 11a shows the landscape with labels for the existing tree visualizations. VR learned the distributions of the existing tree visualizations from the landscape, supporting further exploration for different scenarios. When looking to discover some novel tree visualizations, VR explored the upper-left corner of the landscape where there were with only few existing tree visualizations. The left part of Fig. 11a shows some inspiring tree visualizations found by VR using GoTreeScape. When looking to improve a tree visualization, VR first located the tree visualization in GoTreeScape. He then explored other possible candidates to find novel tree visualizations from the neighboring area through the bottom-up exploration mode. These tree visualizations can provide users with much inspiration and improve the efficiency with which they can explore novel ideas. Fig. 11b shows the landscape when an icicle plot tree visualization was set as the focus of the bottom-up exploration. Here, VR found some tree visualizations following an annual-ring shape in the landscape.

5.3 User Feedbacks

After they had used GoTreeScape, we conducted one-to-one 30-minute interviews with the two participants to collect their feedback. During the interview, the participants were encouraged to comment and ask questions on any aspect of GoTreeScape they felt was important. We answered their questions and made detailed records of their response. VD commented on the diversity of the tree visualizations displayed on the landscape: *"It is amazing to me that so many possible tree visualizations exist."*. VR was satisfied that GoTreeScape could provide so many tree visualization previews directly: *"I am impressed that [GoTreeScape] can provide me with tree visualization results directly so that I can judge the novelty of techniques more efficiently."* Another interesting finding was that users could not understand some of the tree visualizations during the exploratory design. For example, VD proposed the questions: *"I do not understand why [an unreasonable treevis] is a tree visualization."* The tree visualizations that VD did not understand fell into two main categories. The first category did not show the topological structure clearly. The second category contained some novel tree visualizations, and users were not sure about their benefits and application scenarios. This shows that GoTreeScape exposes users to know some very different tree forms (as well as some unreasonable tree visualizations) that fit the rules for encoding their hierarchical information. Even though these trees may not be an efficient form of visualization, it does give users knowledge that there are some stones unturned.

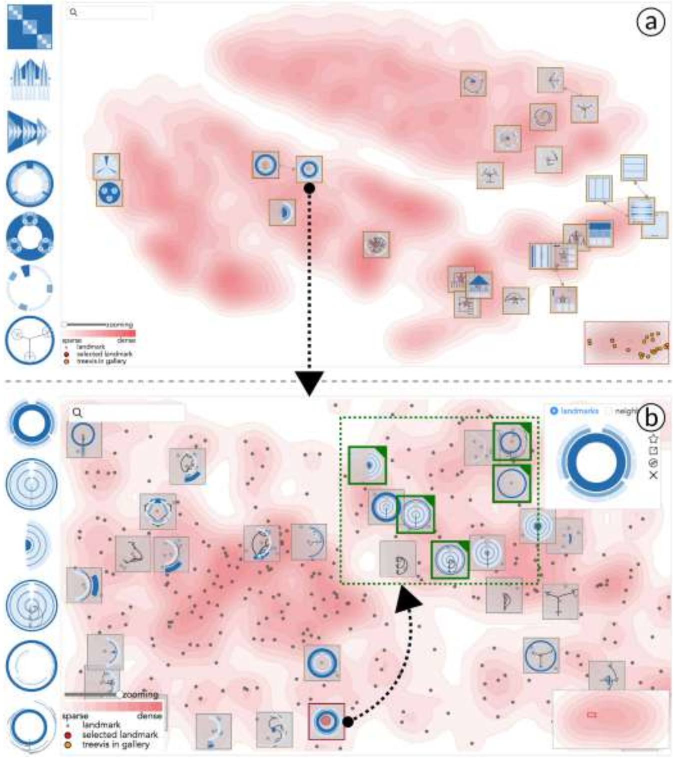

Fig. 11. Top: The distribution of existing tree visualizations in the GoTreeScape. The right part shows some inspiring tree visualizations identified by users. Bottom: The bottom-up exploration process start from the tree visualization highlighted with the red border. The right part shows some tree visualizations in annual ring form.

6 Discussion and Future work

The solutions competing with GoTreeScape include treevis.net [8], Tree Illustrator [4], and the phylogenetic tree-based method (PT) [28]. Given these competing solutions differ in the motivation, expressiveness, the availability of tutorials, and system prototypes, which involve many confounding variables, we did not conduct a quantitative user experiment to evaluate GoTreeScape. Rather, we compared GoTreeScape with the alternatives in four aspects: the number of tree visualizations; whether they differentiate different design features; whether an overview is provided; and whether an exploratory design framework is provided. The matrix of answers are shown in Table 3. As can be seen, GoTreeScape is the only method that meets every criteria as

TABLE 3
Comparison of Design Space Exploration Techniques

Techniques	Treevis.net	Phylogenetic-tree-based method	Tree Illustrator	GoTreeScape
Number of tree visualizations	333	35	countless	62340
Design feature differentiation	No	Yes	No	Yes
Show overview	No	Yes	No	Yes
Provide exploratory framework	No	Yes	No	Yes

well as offering a large number of tree visualizations. The PT method provides an overview based on a phylogenetic tree and allows users to specify the weights of design features dynamically. However, it only contains 35 tree visualizations. treevis.net has assembled 333 visualizations (at the time of this writing) and classifies them according to their dimensionality, representation, and alignment, but it does not provide users with an overview or a way to explore them. Users can select any tree visualizations existing in GoTreeScape using Tree Illustrator, but Tree Illustrator does not provide an overview and users need to gradually determine the design features without viewing the tree visualization results. As such, Tree Illustrator requires users to have a clear target in mind before they start building their visualization. By contrast, GoTreeScape allows users to directly select the satisfactory tree visualizations, after which they can continue to make fine-grained adjustments.

Although GoTreeScape employs domain expertise to filter the generated collection of tree visualizations, some unreasonable tree visualizations will still appear. These tree visualizations do place a cognitive burden on the users, hindering efficient exploration, because these tree visualizations are especially difficult to understand. However, from our case studies, we found that these unreasonable trees provided the users with inspirations during their exploration process. Hence, we plan to deploy this system online and track this activity within a community of users in the future. With more feedback, we can better estimate a good distribution of tree visualizations. Further, as more and more users participate, such estimations will derive a more intuitive exploratory design tool, creating a self-reinforcing system that becomes easier to use. By collecting the users' exploration paths, we might also be able to automatically recommend tree visualizations to users based on other users' previous decisions.

To construct an overview of tree visualization design space, we studied the generated tree visualization collection and extracted insights to guide the loss function and the model structure design. We assigned various weights to the design features according to the magnitude of impact that features would have on tree visualizations. The hypothesis is that the similarities between tree visualizations in terms of human cognition related to the impact of the visual channel. For example, changing the coordinate system always changes both node shapes and the layout of the tree visualizations. Therefore, the coordinate system design feature has the most significant impact on tree visualizations from the standpoint of human cognition. However, human cognition over different design features in the realm of tree visualizations is still an open question. The design features of some tree visualizations make a significant differences, but their visualization results are similar — for example, the triangle and sectors visual elements in the polar coordinate system. Therefore, when users make decisions about further exploration based on the GoTreeScape, they need to consider both a single tree visualization and the other tree visualizations in that context. We plan to design comprehensive user experiments to explore the relationships between human cognition and the tree visualizations' design features. Part of this will involve comparing the preferences of different users (e.g., data scientists and visual designers) when it comes to image-based and grammar-based landscape construction methods. In addition, we will explore the techniques to better realize the consistency between the grammar design and the visualization results. Keeping the efficiency of computing a layout in mind, we intend to use a data-independent techniques. Additionally, GoTreeScape makes the data-oriented exploration of the design space possible by allowing users to upload their hierarchical data from which all tree visualization results are generated. It would therefore be interesting to explore a data-dependent landscape construction method. We also plan to improve the constructed landscape in terms of the machine learning models. For example, we may be able to design the model's structure in a way that preserves the hierarchy of the design features better.

GoTreeScape is not targeted at the whole process of tree visualization design, such as the domain situation and data/task abstraction in the nested model by Munzner [7]. It only focuses on the step of exploring the design space so as to find a suitable tree visualization when users are not clear about their targeted visualizations or only have partial design features in mind. More specifically, GoTreeScape helps users understand the tree visualization design space. It helps them expand their known space and their consideration space. GoTreeScape uses the metaphor of contour-based map to present the tree visualization design space, where the contours of the landscape indicate the distribution of visualizations. In the future, we plan to explore the other map metaphors for design space visualizations. For example, a grid-based metaphor [65] might present more explicit boundaries between different clusters and avoids the overlapping between representative landmarks on the landscape. In addition, it would also be interesting to conduct user experiments to compare the effectiveness of different map metaphors for presenting visualization design spaces. Lastly, the methods proposed in this work could be modified for use as a way to explore the design space of other visualization subcategories with a declarative grammar, for example, ATOM [3] (for unit visualizations), multiclass density maps [5], and so on. In the future, we plan to use the techniques in GoTreeScape for other visualizations. With the increasing number of declarative grammars proposed in visualization research communities, it may also be worthwhile designing a general framework for visualization design space exploration based on declarative grammars.

7 Conclusion

In this paper, we presented GoTreeScape, a system that helps users to navigate and explore the tree visualization design space implied by a fine-grained declarative grammar. GoTreeScape comprises three parts: visualization set generation, landscape construction, and an exploration framework. An encoder-decoder architecture is used to project tree visualizations into a two-dimensional landscape. We employ domain expertise to simplify the visualization set and guide the model design. To address user's varying requirements and scenarios, GoTreeScape provides an exploration framework with top-down, bottom-up, and hybrid modes within GoTreeScape. We applied GoTreeScape to several tree visualization design scenarios within two case studies to demonstrate its usability. The results show that GoTreeScape can expand the diversity of constructed tree visualizations.

Acknowledgments

The authors thank the anonymous reviewers for their helpful comments and valuable feedback.

References

[1] A. Satyanarayan, D. Moritz, K. Wongsuphasawat, and J. Heer, "Vega-Lite: A grammar of interactive graphics," *IEEE Trans. Vis. Comput. Graph.*, vol. 23, no. 1, pp. 341–350, Jan. 2017.
[2] A. Satyanarayan and J. Heer, "Lyra: An interactive visualization design environment," *Comput. Graph. Forum*, vol. 33, no. 3, pp. 351–360, 2014.
[3] D. Park, S. M. Drucker, R. Fernandez, and N. Elmqvist, "Atom: A grammar for unit visualizations," *IEEE Trans. Vis. Comput. Graph.*, vol. 24, no. 12, pp. 3032–3043, Dec. 2018.
[4] G. Li, M. Tian, Q. Xu, M. J. McGuffin, and X. Yuan, "GoTree: A grammar of tree visualizations," in *Proc. ACM Conf. Hum. Factors Comput. Syst.*, 2020, pp. 1–13.
[5] J. Jo, F. Vernier, P. Dragicevic, and J. Fekete, "A declarative rendering model for multiclass density maps," *IEEE Trans. Vis. Comput. Graph.*, vol. 25, no. 1, pp. 470–480, Jan. 2019.
[6] L. Wilkinson, *The Grammar of Graph.*. Berlin, Germany: Springer, 2005.
[7] T. Munzner, *Visualization Analysis and Design*. Boca Raton, FL, USA: CRC Press, 2014.
[8] H. Schulz, "Treevis.net: A tree visualization reference," *IEEE Comput. Graph. Appl.*, vol. 31, no. 6, pp. 11–15, Nov.-Dec. 2011.
[9] M. Bostock, V. Ogievetsky, and J. Heer, "D^3: Data-driven documents," *IEEE Trans. Vis. Comput. Graph.*, vol. 17, no. 12, pp. 2301–2309, Dec. 2011.
[10] G. Li, M. Tian, Q. Xu, M. J. McGuffin, and X. Yuan, "Tree Illustrator: Interactive construction of tree visualizations," in *Proc. Extended Abstacts ACM Conf. Hum. Factors Comput. Syst.*, 2020, pp. 1–4.
[11] H. Schulz, Z. Akbar, and F. Maurer, "A generative layout approach for rooted tree drawings," in *Proc. IEEE Pacific Visualization Symp.*, 2013, pp. 225–232.
[12] J. O. Talton, D. Gibson, L. Yang, P. Hanrahan, and V. Koltun, "Exploratory modeling with collaborative design spaces," *ACM Trans. Graph.*, vol. 28, no. 5, pp. 1–10, Dec. 2009.
[13] T. J. Jankun-Kelly and K.-L. Ma, "A spreadsheet interface for visualization exploration," in *Proc. IEEE Visualization*, 2000, pp. 69–76.
[14] J. Marks et al., "Design galleries: A general approach to setting parameters for computer graphics and animation," in *Proc. Conf. Comput. Graph. Interactive Techn.*, 1997, pp. 389–400.
[15] T. J. Jankun-Kelly and Kwan-LiuMa, "Visualization exploration and encapsulation via a spreadsheet-like interface," *IEEE Trans. Vis. Comput. Graph.*, vol. 7, no. 3, pp. 275–287, Mar. 2001.
[16] E. H. Chi, P. Barry, J. Riedl, and J. Konstan, "A spreadsheet approach to information visualization," in *Proc. VIZ: Visualization Conf. Inf. Visualization Symp. Parallel Rendering Symp.*, 1997, pp. 17–24.
[17] M. Beham, W. Herzner, M. E. Gröller, and J. Kehrer, "Cupid: Cluster-based exploration of geometry generators with parallel coordinates and radial trees," *IEEE Trans. Vis. Comput. Graph.*, vol. 20, no. 12, pp. 1693–1702, Dec. 2014.
[18] K. Wongsuphasawat et al., "Voyager 2: Augmenting visual analysis with partial view specifications," in *Proc. ACM Conf. Hum. Factors Comput. Syst.*, 2017, pp. 2648–2659.
[19] K. Wongsuphasawat, D. Moritz, A. Anand, J. D. Mackinlay, B. Howe, and J. Heer, "Voyager: Exploratory analysis via faceted browsing of visualization recommendations," *IEEE Trans. Vis. Comput. Graph.*, vol. 22, no. 1, pp. 649–658, Jan. 2016.
[20] J. Zhao, M. Fan, and M. Feng, "Chartseer: Interactive steering exploratory visual analysis with machine intelligence," *IEEE Trans. Vis. Comput. Graph.*, vol. 28, no. 3, pp. 1500–1513, Mar. 2021.
[21] S. Xu, C. Bryan, J. K. Li, J. Zhao, and K. Ma, "Chart constellations: Effective chart summarization for collaborative and multi-user analyses," *Comput. Graph. Forum*, vol. 37, no. 3, pp. 75–86, 2018.
[22] B. Kerr, "THREAD ARCS: An email thread visualization," in *Proc. IEEE Symp. Informat. Visualization*, 2003, pp. 211–218.
[23] T. Munzner, F. Guimbretière, S. Tasiran, L. Zhang, and Y. Zhou, "TreeJuxtaposer: Scalable tree comparison using focus+context with guaranteed visibility," *ACM Trans. Graph.*, vol. 22, no. 3, pp. 453–462, Mar. 2003.
[24] F. Block, M. S. Horn, B. C. Phillips, J. Diamond, E. M. Evans, and C. Shen, "The DeepTree exhibit: Visualizing the tree of life to facilitate informal learning," *IEEE Trans. Vis. Comput. Graph.*, vol. 18, no. 12, pp. 2789–2798, Dec. 2012.
[25] B. Johnson and B. Shneiderman, "Tree maps: A space-filling approach to the visualization of hierarchical information structures," in *Proc. IEEE Visualization*, 1991, pp. 284–291.
[26] G. Li et al., "BarcodeTree: Scalable comparison of multiple hierarchies," *IEEE Trans. Vis. Comput. Graph.*, vol. 26, no. 1, pp. 1022–1032, Jan. 2020.
[27] S. Zhao, M. J. McGuffin, and M. H. Chignell, "Elastic hierarchies: Combining treemaps and node-link diagrams," in *Proc. IEEE Symp. Inf. Vis.*, 2005, pp. 57–64.
[28] S. Li et al., "Exploring hierarchical visualization designs using phylogenetic trees," in *Visualization and Data Analysis*. Bellingham, WA, USA: SPIE, 2015, pp. 68–81.
[29] H.-J. Schulz and S. Hadlak, "Preset-based generation and exploration of visualization designs," *J. Vis. Lang. Comput.*, vol. 31, pp. 9–29, 2015.
[30] H. Schulz, S. Hadlak, and H. Schumann, "The design space of implicit hierarchy visualization: A survey," *IEEE Trans. Vis. Comput. Graph.*, vol. 17, no. 4, pp. 393–411, Apr. 2011.
[31] T. Baudel and B. Broeksema, "Capturing the design space of sequential space-filling layouts," *IEEE Trans. Vis. Comput. Graph.*, vol. 18, no. 12, pp. 2593–2602, Dec. 2012.
[32] S. MacNeil and N. Elmqvist, "Visualization mosaics for multivariate visual exploration," *Comput. Graph. Forum*, vol. 32, no. 6, pp. 38–50, 2013.
[33] A. Slingsby, J. Dykes, and J. Wood, "Configuring hierarchical layouts to address research questions," *IEEE Trans. Vis. Comput. Graph.*, vol. 15, no. 6, pp. 977–984, Nov./Dec. 2009.
[34] S. K. Card and J. Mackinlay, "The structure of the information visualization design space," in *Proc. VIZ: Visualization Conf. Informat. Visualization Symp. Parallel Rendering Symp.*, 1997, pp. 92–99.
[35] W. Javed and N. Elmqvist, "Exploring the design space of composite visualization," in *Proc. IEEE Pacific Visualization Symp.*, 2012, pp. 1–8.
[36] J. F. Rodrigues, A. J. M. Traina, M. C. F. de Oliveira, and C. Traina, "The spatial-perceptual design space: A new comprehension for data visualization," *Inf. Visualization*, vol. 6, no. 4, pp. 261–279, 2007.
[37] M. Tory and T. Möller, "Rethinking visualization: A high-level taxonomy," in *Proc. IEEE Symp. Inf. Visualization*, 2004, pp. 151–158.
[38] Y. S. Kristiansen and S. Bruckner, "Visception: An interactive visual framework for nested visualization design," *Comput. Graph.*, vol. 92, pp. 13–27, 2020.
[39] M. Brehmer, B. Lee, B. Bach, N. H. Riche, and T. Munzner, "Timelines revisited: A design space and considerations for expressive storytelling," *IEEE Trans. Vis. Comput. Graph.*, vol. 23, no. 9, pp. 2151–2164, Sep. 2017.
[40] A. Kerren, K. Kucher, Y.-F. Li, and F. Schreiber, "Biovis explorer: A visual guide for biological data visualization techniques," *PLoS One*, vol. 12, no. 11, pp. 1–14, 11 2017.

[41] H. Guo, W. Li, and X. Yuan, "Transfer function map," in *Proc. IEEE Pacific Visualization Symp.*, 2014, pp. 262–266.
[42] Kwan-LiuMa, "Image graphs-a novel approach to visual data exploration," in *Proc. IEEE Visualization*, 1999, pp. 81–88.
[43] Y. Wu, A. Xu, M. Chan, H. Qu, and P. Guo, "Palette-style volume visualization," in *Proc. IEEE Int. Symp. Volume Graph.*, 2007, pp. 33–40.
[44] H. Guo, N. Mao, and X. Yuan, "WYSIWYG (what you see is what you get) volume visualization," *IEEE Trans. Vis. Comput. Graph.*, vol. 17, no. 12, pp. 2106–2114, Dec. 2011.
[45] F. Bolte and S. Bruckner, "Vis-a-Vis: Visual exploration of visualization source code evolution," *IEEE Trans. Vis. Comput. Graph.*, vol. 27, no. 7, pp. 3153–3167, Jul. 2021.
[46] D. Moritz et al., "Formalizing visualization design knowledge as constraints: Actionable and extensible models in draco," *IEEE Trans. Vis. Comput. Graph.*, vol. 25, no. 1, pp. 438–448, Jan. 2019.
[47] J. Hullman, S. M. Drucker, N. H. Riche, B. Lee, D. Fisher, and E. Adar, "A deeper understanding of sequence in narrative visualization," *IEEE Trans. Vis. Comput. Graph.*, vol. 19, no. 12, pp. 2406–2415, Dec. 2013.
[48] Y. Kim, K. Wongsuphasawat, J. Hullman, and J. Heer, "Graphscape: A model for automated reasoning about visualization similarity and sequencing," in *Proc. ACM Conf. Hum. Factors Comput. Syst.*, 2017, pp. 2628–2638.
[49] J. Tukey, *Exploratory Data Analysis*, vol. 2. Noida, UP, India: Pearson, 1977.
[50] T. J. Jankun-Kelly, K. Ma, and M. Gertz, "A model and framework for visualization exploration," *IEEE Trans. Vis. Comput. Graph.*, vol. 13, no. 2, pp. 357–369, Mar./Apr. 2007.
[51] S. I. Fabrikant, D. R. Montello, and D. M. Mark, "The natural landscape metaphor in information visualization: The role of commonsense geomorphology," *J. Amer. Soc. Informat. Sci. Technol.*, vol. 61, no. 2, pp. 253–270, 2010.
[52] M. Blades et al., "A cross-cultural study of young children's mapping abilities," *Trans. Inst. Brit. Geographers*, vol. 23, no. 2, pp. 269–277, 1998.
[53] J. B. Kruskal, *Multidimensional Scaling*. Newbury Park, CA, USA: Sage, 1978.
[54] I. T. Jolliffe, *Principal Component Analysis and Factor Analysis*. Berlin, Germany: Springer, 1986, pp. 115–128.
[55] L. McInnes, J. Healy, and J. Melville, "UMAP: Uniform manifold approximation and projection for dimension reduction," 2018, *arXiv:1802.03426*.
[56] L. Van der Maaten and G. Hinton, "Visualizing data using T-SNE," *J. Mach. Learn. Res.*, vol. 9, no. 86, pp. 2579–2605, 2008.
[57] D. P. Kingma and M. Welling, "An introduction to variational autoencoders," *Found. Trends Mach. Learn.*, vol. 12, no. 4, pp. 307–392, 2019.
[58] M. J. Kusner, B. Paige, and J. M. Hernández-Lobato, "Grammar variational autoencoder," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 1945–1954.
[59] D. Kobak and P. Berens, "The art of using t-SNE for single-cell transcriptomics," *Nat. Commun.*, vol. 10, no. 1, pp. 1–14, 2019.
[60] R. Real and J. M. Vargas, "The probabilistic basis of jaccard's index of similarity," *Systematic Biol.*, vol. 45, no. 3, pp. 380–385, 1996.
[61] F. Aurenhammer, "Voronoi diagrams—a survey of a fundamental geometric data structure," *ACM Comput. Surv.*, vol. 23, no. 3, pp. 345–405, 1991.
[62] D. Ceneda et al., "Characterizing guidance in visual analytics," *IEEE Trans. Vis. Comput. Graph.*, vol. 23, no. 1, pp. 111–120, Jan. 2017.
[63] D. Auber, "Using strahler numbers for real time visual exploration of huge graphs," *J. WSCG Int. Conf. Comput. Vis. Graph.*, vol. 10, no. 1/3, pp. 56–69, 2002.
[64] D. J.-L. Lee, J. Lee, T. Siddiqui, J. Kim, K. Karahalios, and A. Parameswaran, "You can't always sketch what you want: Understanding sensemaking in visual query systems," *IEEE Trans. Vis. Comput. Graph.*, vol. 26, no. 1, pp. 1267–1277, Jan. 2020.
[65] W. Meulemans, M. Sondag, and B. Speckmann, "A simple pipeline for coherent grid maps," *IEEE Trans. Vis. Comput. Graph.*, vol. 27, no. 2, pp. 1236–1246, Feb. 2021.

Guozheng Li received the PhD degree in computer science from the school of EECS, Peking University, in 2021. He is currently an Assistant Professor with the School of Computer Science and Technology, Beijing Institute of Technology, Beijing. His major research interests include information visualization, especially hierarchical data visualization and visualization authoring.

Xiaoru Yuan (Senior Member, IEEE) received the BS degree in chemistry and the BA degree in law from Peking University, in 1997 and 1998 respectively. In 2005 and 2006, the MS degree in computer engineering and the PhD degree in computer science with the University of Minnesota, Twin Cities. He is now a professor with Peking University with the Laboratory of Machine Perception (MOE). His primary research interests lie in the field of scientific visualization, information visualization and visual analytics with an emphasis on large data visualization, high dimensional data visualization, graph visualization and novel visualization user interface.