TimeSeriesMaker: Interactive Time Series Composition in No Time

Franziska Becker* Tanja Blascheck†

University of Stuttgart, Germany

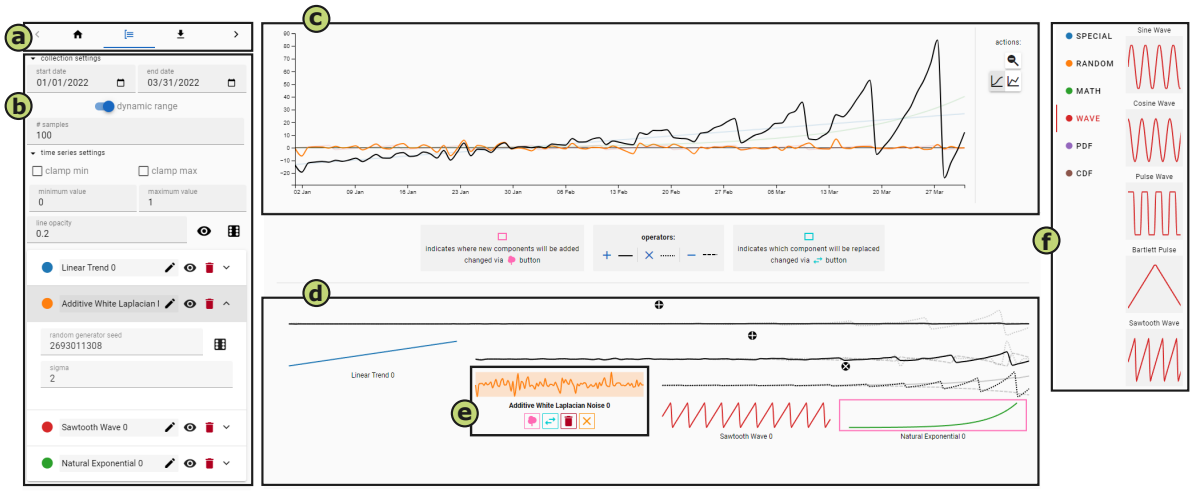

Figure 1: The compositor page of **T**ime**S**eries**M**aker showing (a) page tabs, (b) the list of *components* for a *time series* where analysts can change their parameters, (c) a line chart showing the final result and all *components*, (d) the compositor tree visualization, (e) highlighted *component* with action buttons and (f) the component picker with small previews for each *component* option.

Abstract

Time**S**eries**M**aker is an open-source application to visually compose time series data in an intuitive and shareable manner. Visualization researchers often use time series data in studies about perceptual or cognitive phenomena and many other contexts. However, finding or generating time series data that fits a given scenario is not always easy. Using a component-based architecture, **T**ime**S**eries**M**aker allows analysts to compose time series data with complex patterns by combining different components, such as noise, a linear trend or a seasonal pattern. An interactive compositor tree of these components lets analysts explore their combinations using different operators. We support reproducibility and transparency by including functionalities that allow analysts to export and share their configuration, which others can use to reload and modify the same time series. In a qualitative online study with visualization researchers, we found that our approach enables them to create a time series based on an example image or their own requirements. However, system usability could be further improved when interacting with the compositor tree. **T**ime**S**eries**M**aker can be found here: https://unistuttgart-visus.github.io/time-series-maker/.

Index Terms: Human-centered computing—Visualization systems and tools; Human-centered computing—Visualization; Information systems—Open source software

1 Introduction

Time series data is everywhere—on financial markets, in simulations, lab experiments, population changes, interaction sequences, weather recordings and more. Visualizations of such time series data can range from simple and well-known visualizations to complex handcrafted visualizations for special analysis scenarios. Because of their ubiquity, time series visualizations can be an attractive option to use as experiment stimuli. If researchers aim to study perceptual or cognitive phenomena, they often want to reach a large and diverse audience, so familiar visualizations are well-suited for such scenarios. Appropriate stimuli may need to exhibit specific perceptual or mathematical requirements. Similarly, studies that evaluate visualization designs for time series data may also use generated data so they can test data with different characteristics and complexity. On the most basic level, desired characteristics for time series data often include some portion of noise or randomness to obfuscate other patterns or to appear more realistic. Other demands can include the existence of a visible peak, a certain periodicity or relations between different time series. With **T**ime**S**eries**M**aker (cf. Fig. 1), we developed an open-source visualization application that supports analysts in constructing time series data. It focuses on a visual composition using components, so analysts can explore different options to see which ones satisfy their needs. With the help of an interactive *compositor tree*, they can iteratively build up their time series and explore the space of combinations by quickly swapping components and trying out different operators to combine them. We evaluated our approach in a qualitative online study with four researchers who have experience with visualization and provided some insight into their experiences with time series data and its generation. **T**ime**S**eries**M**aker is available at https://unistuttgart-visus.github.io/time-series-maker/ and the source code can be found at https://github.com/UniStuttgart-VISUS/time-series-maker.

2 Related Work

In the context of scientific studies, researchers may either generate data or use existing (open-source) datasets as input for stimuli. We consider related work that either studies perceptual phenomena *using*

*e-mail: franziska.becker@vis.uni-stuttgart.de
†e-mail: tanja.blascheck@vis.uni-stuttgart.de

time series data or evaluates a visualization design for time series data. Few works (e. g., [5, 16]) consider only real-world datasets, which has the advantage of providing external validity but may come at the cost of lower diversity and less control over data characteristics. Depending on the research question and tasks, data may need to exhibit specific patterns or fulfill certain mathematical requirements. In their experiments about aggregation in time series visualization, Albers et al. [4] generated time series data which had to fit several constraints, for example, noise, different difficulty levels and decorrelation requirements. The authors generated their data by solving an optimization problem or manually adjusting a signal. Similarly, Song and Szafir [18] used constraint-based optimization to adjust the mean differences of an initial signal created by five different noise levels. Wu et al. [23] wrote a data generation algorithm to create time series with different difficulty levels and to fit the tasks they investigated, such as extrema identification or value comparison. For extrema identification, they used a mixture of pseudo-random number generation, noise, smoothing using cubic b-splines and manual tuning to generate data with varying difficulty levels that also reflect the semantics of a specific real-world dataset. Other works generate time series data by using a random walk [9, 13, 15] or common statistical functions like normally distributed random numbers in Matlab [24]. Thudt et al. [19] looked at the readability of three stacked graph visualizations and generated visualizations both from two real-world datasets and by generating data using source code[1] that Byron and Wattenberg [8] wrote to visualize streamgraphs. They state that they used Byron's generator to create time series with varying temporal patterns, though this is not described in more detail. A look at the source code suggests that a fixed number of bumps is most likely created to generate the time series data. For SineStreams, Bu et al. [7] similarly employed Byron's data generation code, but they supplemented this with a larger number of real-world datasets.

Regarding visualizations of time series data, Aigner et al. [1–3] provided a thorough overview in three related works. They view time series data in terms of its temporal structure, related data and representation. Time can be linear, cyclical or branching; its related data can exhibit different characteristics like being abstract or geographic, being uni- or multivariate and require different abstraction levels to be visualized effectively. The authors emphasize that parameters for visualization techniques and interaction functionalities have high importance when analyzing time series data in a visualization context. While our usage scenario is not limited to any particular type of time series data, we focus on the most common type—linear time series data. Visualizations of such data include simple line or bar charts, but also small multiples [20] and circular layouts [22] as well as stacked visualizations as seen in the theme river [12] and stream graph [8] approaches. For time series data with periodic patterns, Van Wijk et al. [21] analyzed clustering and calendar-based visualizations that allow finding recurring patterns. Weber et al. [22] visualized time series on spirals, which are suitable for larger time series and better at showing data periodicity. To efficiently explore different periodicities in time series data, Franke and Koch [10] employed dense pixel-based visualizations.

3 Design

For the design of **T**ime**S**eries**M**aker, we started with our own use case: creating a time series with specific *visible* patterns to generate stimuli for an online study. Based on this use case and related works with a similar context, we came to the following requirements:

R1 Iterative composition of different time series.

R2 Complete control of all generation parameters.

R3 Inclusion of random factors, like noise.

R4 Visualization of the time series and its constituting elements.

R5 Making the time series generation shareable and replicable.

By iteratively composing the time series from a set of simple time series (**R1**), analysts can construct complex patterns and simultaneously retain a record of the elements it consists of. A similar strategy is often used when generating data with scripts, making it possible to adjust only those parts that need tweaking. Including random factors (**R3**) lets analysts construct time series data that more closely resembles real data or obfuscate patterns from other components. In addition, most related works we discussed previously include some form of randomness in their data generation process. While analysts may have an idea or mental image of what their time series should *look* like, they may not know exactly how this can be achieved through time series composition. Modifying any parameter (**R2**) and immediately seeing how this change affects the resulting time series (**R4**) allows for rapid exploration of the space of existing time series. Since a research context demands transparency and replicability, respective functionalities are included in our approach. In practice, many researchers share either the data or the algorithms used to generate their data. Accessing such data can be more or less difficult, so we include options to share both the resulting data as well as the settings that are used to generate it (**R5**).

3.1 Time Series Representation

Time**S**eries**M**aker employs four types of objects to represent time series: the *time series collection*, *time series*, *compositor tree* and *component*. The *time series collection* consists of an arbitrary number of *time series*, which in turn consists of *components* that each contains their own time series. For the collection, analysts can specify the time range, the number of samples and the drawing range for all time series that belong to it. Each *time series* has a name and a group of *components*, which are composed by the *compositor tree* to form the final time series. In case analysts want to restrict the value range of its data, they can define an upper and lower bound.

Components *Components* are divided into six different categories: `SPECIAL`, `RANDOM`, `MATH`, `WAVE`, `PDF` (probability density function) and `CDF` (cumulative probability density function). The stdlib[2] framework partly inspired these categories and has a rich supply of math functions, random generators, statistical functions, as well as other utilities for scientific computing. Existing knowledge about patterns in time series data also informed our choice of *components*. Time series data can be viewed in terms of their patterns over time. Common patterns include trends, outliers and cycles or are based on seasonality (see e. g., Hyndman and Athanasopoulos [14]). A trend is an overall persisting increase or decrease, like a linear line with a positive slope—though a trend need not be linear. Outliers represent values, unlike all others, for example, a sudden spike in price in a particular year. The imagery often invoked for such cases is a valley or hill. Cyclical patterns denote fluctuations without a specific period, whereas seasonal patterns are fluctuations with a fixed period, often related to calendar units like weeks or months. In the following, we list exemplary *components* for each category available in **T**ime**S**eries**M**aker. `SPECIAL` is the largest category and contains specific patterns not easily created via other means, such as outliers and periodic patterns based on days, weeks or months. Random functions can be found in the `RANDOM` category, including different types of noise or random numbers of varying distribution. When a *time series* has one or more *components* from the `RANDOM` category, analysts can specify the number of instances for the *time series*. The *time series* is then replicated the chosen number of times with a unique random seed for each random *component* (cf. Fig. 2). The `MATH` category contains common mathematical functions like exponential, logarithmic or trigonometric functions.

[1] https://github.com/leebyron/streamgraph

[2] https://github.com/stdlib-js/stdlib

It partially overlaps with the WAVE category, where analysts may choose among different waves like a sine, cosine, pulse, bartlett pulse or sawtooth wave. Finally, the PDF and CDF categories contain a few (cumulative) probability distribution functions, respectively. In total, **T**ime**S**eries**M**aker features 50 different *components*, each with adjustable parameters. For example, a sawtooth wave has a period, an amplitude, an offset and a scale that defines its shape. In addition, all *components* from the random category have a seed that is fed to their random generator.

Compositor Tree The *compositor tree* of a *time series* is a binary tree that determines how the different *components* of a *time series* are combined. Leaf nodes contain the *components*, while intermediate nodes represent one of three operators: addition (⊕), subtraction (⊖) or multiplication (⊗). When generating the data for a *time series*, the tree is parsed from bottom to top, combining *components* using the respective operator from their parent node. Leaf nodes can be swapped and operators can be switched between, thereby allowing analysts to create complex combinations of simple time series patterns.

3.2 Visualization and Interaction Design

Time**S**eries**M**aker consists of five views, available in different tabs: home (🏠), composition (☰), export (⬇), import (⬆) and help (?). Analysts can switch between tabs using an icon menu in the main panel on the left side of the page, as shown in Fig. 1a. Each tab follows the same layout but provides different functionalities. In the main panel on the left side, it always displays the page tabs and any available settings, e. g. the number of samples for the *time series collection*. The middle section of the page displays any available visualizations, like a line chart depicting the different *components* of a single *time series*. On the right side, we reserve space for an additional side panel which is only used when analysts are in the composition tab.

In the export tab (⬇), analysts can download their *time series collection* or *time series*, either only as *settings* or as *data*. If settings are exported, a JSON file is created which contains all information that is required to generate the respective time series in **T**ime**S**eries**M**aker (**R5**). If data is exported, we generate a CSV file in which each time series corresponds to one row and each column is a sample point. For both cases, analysts are shown a preview of what their files look like. Exported settings can be loaded into the application in the import tab (⬆), allowing analysts to modify previous creations (**R5**). The help tab (?) features a collection of short videos that briefly demonstrate different system functionalities.

In a typical workflow, analysts start in the home tab (🏠), which acts as a hub to manage the *time series collection*. The main panel displays the collection's settings and lists all of its *time series*, while the center area contains a line chart that visualizes the *time series*, each with its respective color (cf. Fig. 2). The line chart can be zoomed and panned using common scroll and drag interactions. Analysts can also perform actions on each *time series*: rename, duplicate, delete or re-roll random seeds. Finally, they can select a *time series* to modify it in the composition tab. Upon selecting a *time series*, analysts are redirected to the composition tab (☰). It has the same overall layout as the home page, but is restricted to a single *time series*. Here, analysts can modify the selected *time series* by adjusting its *components* and *compositor tree*. The main panel lists all *components* and their parameters, as shown in Fig. 1b. Each *component* in the list can be unfolded to view and modify its parameters (**R2**) and its name can be edited by clicking the pencil icon. At the center of the page, the line chart depicts the chosen *time series* and its *components*, each colored by their category (cf. Fig. 1c). It acts as a constant overview, showing all the constituting elements of the *time series* in a single chart (**R4**). Any unfolded *components* are highlighted in the line chart, which temporarily reduces the drawing opacity of all the other lines. In the side panel on the right, the component picker gives an overview of all available *components* (cf. Fig. 1f). It has vertical tabs for each category and displays a small preview line chart to illustrate how each *component* looks like with the default parameters (**R3**). Hovering over a preview displays a tooltip with a general description of the *component* and a list of its parameters with their allowed value range. For example, if a parameter may only be a positive integer in the range from 1 to 31, then this is communicated in a common mathematical notation as follows: $[x \in [1,31] \mid x \in \mathbb{Z}]$. When one of the previews is clicked on, the respective *component* is added to the *compositor tree* (**R1**).

Right below the central line chart, analysts can find the *compositor tree* (cf. Fig. 1 d & e). The *compositor tree* visualization is based on an icicle plot where each node is a small line chart without axes. For leaf nodes, these charts depict the respective *component's* time series (**R4**), which can be swapped via simple drag-and-drop. Intermediate nodes visualize the result of applying any of the three composition operators to its children. Each operator has a different line style it is associated with, as depicted in Fig. 3. The line for the active operator is drawn with full opacity whereas others are drawn with reduced opacity. Analysts can switch between operators either by clicking on the corresponding line or by clicking on the operator icon (⊕, ⊗ or ⊖) above the node which opens a small pop up (cf. Fig. 3). Initially, we did not visualize the result each operator produces when it is applied. However, preliminary tests showed that operators were mostly ignored and analysts struggled to understand how they function without further instruction. Therefore, we opted to visualize any intermediate results and alternative options, so analysts understand how the final time series comes to be. When adding a *component*, the *compositor tree* is expanded at the currently active leaf node, which is highlighted with a pink box around it (cf Fig. 1d). Essentially, the active leaf node is replaced with an operator node with two children: the new node and the active leaf node. Analysts can change this behavior by hovering over a leaf node, which shows a set of four action buttons (cf. Fig. 1e). Basic actions let analysts delete the node in question and hide the action buttons. The tree button can be used to mark the current node as active (i.e. where future nodes should be added). Finally, analysts may also switch to *replacement mode* via the arrow button. In this mode, the chosen node is replaced with the selected *component* instead of adding a new node. This functionality gives analysts more control over the

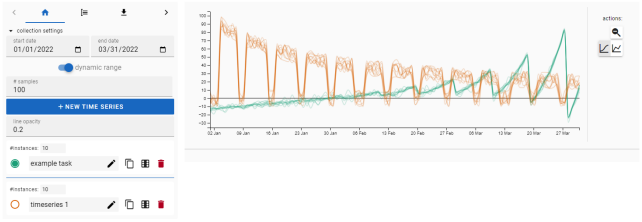

Figure 2: View of the home tab (🏠) showing a *time series collection* with two *time series* that each have 10 instances.

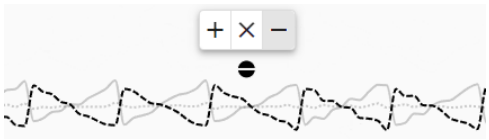

Figure 3: Cleaned close-up of an operator node in the compositor tree, showing a line with a unique line style for each operator. When clicking on the operator indicator, a button choice pops up which analysts can use to change the operator.

construction of their *time series* and provides an avenue for rapid exploration because they can try out replacing a specific *component*, for example, to compare similar types of *components*. When switching to *replacement mode*, the pink highlight shown previously is hidden and a cyan-colored box is drawn instead, indicating which node will be replaced. We opted for the boxed highlights so analysts do not have to perform long drags between the component picker and the correct position in the *compositor tree*. This also lets analysts add new *components* without having to *plan* where in the tree this *component* should go.

3.3 Usage Scenario

In the following, we describe a short usage scenario to build a simple time series. Accompanying screenshots for this scenario can be found in the supplemental material. Anna wants to build a time series with an upward trend that also includes some randomness. She starts out on the home page, where she clicks on the circle for the available *time series* to select it. The application automatically redirects her to the composition tab, which presents her with an empty *time series*. On the right side of the page, Anna sees the component picker with previews for different *components*. She hovers over the previews, which lets her read a brief description for each *component*. By clicking on the respective preview, she adds a linear trend *component* to her *time series*. This action updates the line chart and *compositor tree* visualization. The root node of the *compositor tree* has the default operator (addition) and one child—the linear trend *component*. Anna returns to the component picker and chooses the `RANDOM` category. She considers the different options and adds a random uniform *component*. Now the root of the *compositor tree* has two children, the linear trend and random *components*, which are combined using addition. Content with her result, Anna switches to the export page and downloads the time series she created.

4 Online Study

We conducted a qualitative online study to evaluate how well analysts can navigate **T**ime**S**eries**M**aker with little training, how they rate its usability and which additional requirements they may have for such a system. Our main target analysts are researchers with visualization expertise, therefore we distributed a link to our study at our own and neighboring institutes via email. The study consisted of a questionnaire with a consent form, an example recreation task and an open recreation task. The questionnaire was created using Google Forms while the tasks were performed with our system that can be used in any browser. In the questionnaire, we collected demographic information, self-reported visualization expertise and participants' experience with time series and data generation. Our application includes a short interactive tutorial that highlights different interface elements, together with a brief explanation of their functionality. The tutorial automatically walks through a few steps of creating a time series and can be aborted or restarted on demand. For the example recreation task, we showed analysts an image of a time series we created with **T**ime**S**eries**M**aker, though only the final result, not its *compositor tree*. For the open task, we first asked participants to describe a time series they have generated or seen previously. Then we asked them to recreate the same time series they described in the previous question using our application. To complete both tasks, participants needed to upload a JSON file, which they had to export from **T**ime**S**eries**M**aker and then upload using a cloud link provided in the task description. Their answer should contain the filename they used for the upload. We analyzed the results they submitted for both tasks by importing these files into **T**ime**S**eries**M**aker, which allowed us to see which *components* they made use of. Finally, we asked questions about participants' opinions regarding the usability of the application, based in part on questions from the application usability scale by Brooke [6].

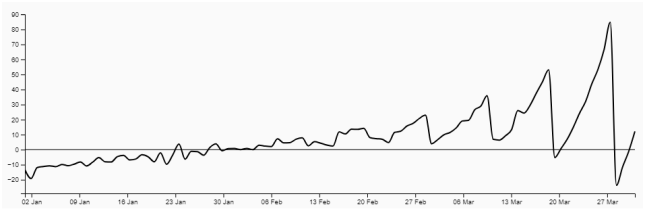

Figure 4: Image given to participants in the example recreation task.

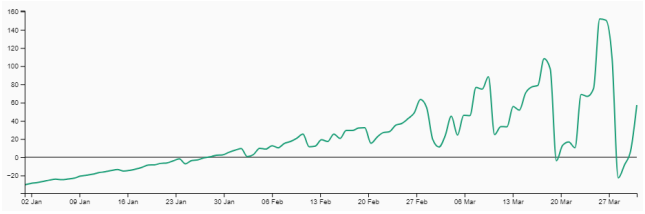

Figure 5: The example recreation task solution for participant P4, in which *components* were not combined correctly but the linear trend intersects the *x*-axis at a similar position.

4.1 Example Recreation Task

The first task aims to familiarize participants with the different features of **T**ime**S**eries**M**aker and simultaneously test their ability to use them. Fig. 4 depicts the image shown to participants for this task and Fig. 1d shows its corresponding *compositor tree*. We created a simple *time series*, consisting of only four components with minor adjustments to their default parameters. However, it uses both addition and multiplication operators as well as *components* from different categories. Specifically, the *time series* consists of a linear trend, additive white uniform noise, a sawtooth wave and a natural exponential function. The latter two *components* are multiplied and then added to the former. To get the correct result, the linear trend needs to be moved so it crosses the *x*-axis at the approximately right position and the exponential function needs to be adjusted so it does not *swallow* all other components.

4.2 Participants

A total of four participants took part in our study. All participants are in the age range of 25 to 34, with one female and three male participants. Everyone rated their visualization expertise as high (P1, P2) or very high (P3, P4) and all of them have previously worked with time series data. Three participants (P2, P3, P4) have generated time series data before and they all reported using Python for the generation, while P2 also reported using Julia, R and Matlab. When asked about characteristics of the data they generated, participant P2 said their data includes *"noise, dependencies, autocorrelation, complex experimental design dependencies, physical model, equation application."* For the other two participants (P3, P4) who previously generated data, the characteristics were simpler. They reported that their generation included true randomness, outliers, peaks and specific frequencies. All participants have previously used other data sources, either openly available or from a project, though only one (P2) employed synthetic data. Real-world data has been used by all participants and three of them (P2, P3, P4) have used common toy datasets. When asked for which usage scenarios they had employed these datasets, all participants reported using them for applications (i.e., for prototyping and testing), while three (P2, P3, P4) also used them for research studies. We also asked participants how easy it was for them to access and use these datasets. Responses included "Hard" (P1), "Average" (P3) and "Easy" (P2, P4) while the ease of use was uniformly rated as "Easy."

4.3 Results

In the following, we discuss the results of our online study. We start with participants' solutions to the example recreation task; then we describe the characteristics of the time series generated in the open recreation task. Lastly, we detail the usability ratings and free-text responses for **T**ime**S**eries**M**aker.

4.3.1 Example Recreation Task

All participants successfully recreated the time series in the example recreation task, though the likeness to the solution varied across participants. One participant combined all *components* using the correct operators, whereas the three other participants made the same mistake: Instead of multiplying the exponential function with the sawtooth wave and then adding the noise and trend, they added up the sawtooth and noise first, which they then multiplied with an exponential function. While the resulting shape is similar, it is not identical, as can be seen in Fig. 5. Multiplying the noise function with the exponential function means that noise values are *less* pronounced at the start of the time series and simultaneously *more* pronounced towards its end. Two participants did not adjust the default parameters for the linear trend, even though the solution shows the time series intersecting with the x-axis earlier than the default value. All but one participant also did not modify *components* to fit the y-value range of the solution. This may be due to the task description, which stated that the overall shape of the time series is more important than specific values.

4.3.2 Open Recreation Task

For the open recreation task, the complexity of participants' time series varied. Two participants recreated their time series using only two *components*, a combination of noise and a specific pattern (outlier for P3 and sine wave for P4). The other two participants both used eight *components* and employed exactly one random *component* each. Fig. 6 shows the *time series* participants P2 created based on the description they gave beforehand: *"dampened oscillator, 1/f spectrum noise."*

4.3.3 Usability

After the two tasks were completed, participants were asked four questions taken from the SUS questionnaire [6], the results of which are displayed in Fig. 7. Overall, participants found the application well-integrated but cumbersome to use. When asked whether the system was easy to use, answers come to an average of 3 ("Neither Agree nor Disagree"). When asked how much participants felt they needed to learn, answers varied more, ranging from 1 ("Strongly Disagree") to 4 ("Agree"). To get more insight into their usability rating, participants had to answer two free-text questions asking them how the application **helped** or **hindered** them in accomplishing the two tasks. For features that **helped**, all participants named the variety of *components* as a useful feature and two participants further stated that they liked the preview and *compositor tree* functionalities. One participant said that they found the tutorial *"great."* Regarding what **hindered** them, three out of four participants said they would want to rework or expand the compositor tree interactions. Specifically, participants wanted more drag-and-drop interactions, like allowing them to add *components* in that manner and being able to reorder and modify complete subtrees. The response from participant P3 captures this well: *"I think the drag-and-drop reordering is nice. It would be good to also be able to drag new components onto existing ones to create a subtree there. This would make creation easier if the mental image of the pattern does not match the visualization's structure."* Two participants mentioned specific *components* they would like to have added to **T**ime**S**eries**M**aker, like a random walk, cumulative sum or more seasonal patterns. Finally, one participant said that they would additionally like to see the mathematical formula that describes the data generation for a *component*.

5 Discussion

Participants in our online study were able to recreate both a given time series from an image and one they chose themselves with little training. Looking at the time series they created, it seems that participants with *simpler* demands are already well-supported by **T**ime**S**eries**M**aker, while those with advanced requirements need more options regarding *components*, parameters and choice of operators. While participants liked the variety of *components* and the concept of the *compositor tree* to compose a *time series*, they found interacting with the tree and modifying it cumbersome—though they found the various system functions to be well-integrated. In particular, participants wanted more options to adjust the *compositor tree* to their needs, for example, moving whole subtrees and extending the tree with drag-and-drop interactions. Single *components* can be swapped via drag-and-drop, but dragging to add *components* was not implemented, as we considered it too inconvenient. In that case, analysts would need to cover long distances between the component picker and the tree. However, feedback strongly indicates that analysts would prefer drag-and-drop for a larger number of interactions. Overall, interactions in **T**ime**S**eries**M**aker can be improved to provide better usability but the concept of time series composition using a *compositor tree* seems promising to visually construct time series without the need for programming.

Participants mentioned wanting an even larger variety of *components*, particularly for random *components*. Another direction to provide more options to analysts is to add other operations that allow for higher complexity. For example, function composition (i.e., using the output of one function as the input of another) opens

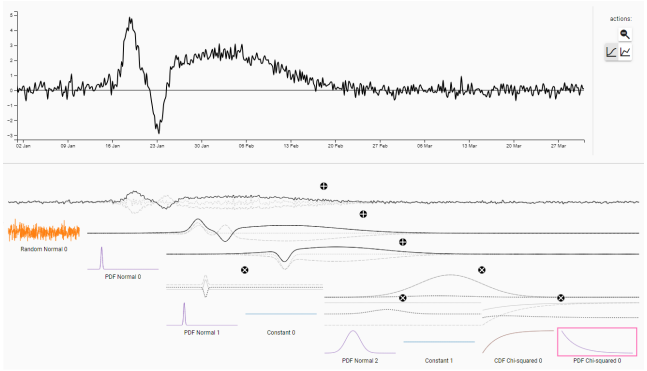

Figure 6: The line chart (top) and *compositor tree* (bottom) for the solution to the open recreation task solution participant P2 submitted.

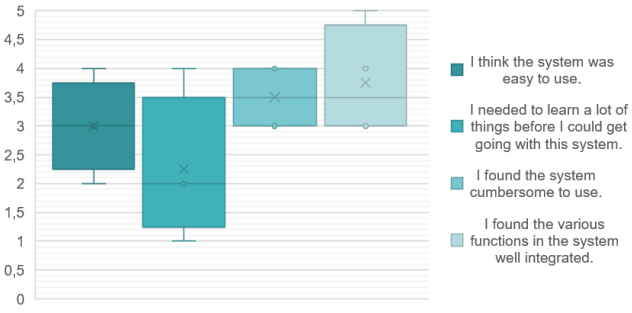

Figure 7: Participant answers for the four usability questions about **T**ime**S**eries**M**aker. Answers were given on a scale from 1 to 5 with the labels "Strongly Disagree" and "Strongly Agree."

up many new combinations. Other functionalities we considered but did not incorporate include different sampling strategies, additional visualizations and a division operator. Regarding the latter, we hesitated to include division as it introduces invalid values that create discontinuities, though this may be desirable for some analysts. In its current state, **T**ime**S**eries**M**aker samples its given time range uniformly, but other sampling strategies could be added to provide more flexibility. For the visualization, we chose a common and simple one: the line chart. It is easy to understand and requires little to no training, but does not perform as well as other methods for certain tasks like finding periodic patterns (see e. g., [17, 22]). Including additional visualizations could be interesting for analysts who want to validate other characteristics of their data, but may require more training that could deter analysts who *just* want to generate simpler time series and feel overwhelmed by many options. We saw in related works that some researchers have specific requirements they achieve using methods like constraint optimization. **T**ime**S**eries**M**aker does not provide any means to specify constraints for a *component*, relations between different *components* or even different *time series*. How to easily define and visualize such constraints is a challenge for future work to investigate, although it could certainly provide the complexity advanced analysts are still missing. In terms of scalability, the *compositor tree* visualization works well for up to ten *components* at full HD resolution. However, scalability suffers for larger numbers, as each *component* requires a minimum size to depict its time series. There are two obvious routes to address this issue: hiding parts of the *compositor tree* on demand or implementing a focus+context approach [11]. Both strategies come with their own challenges. Hiding parts of the *compositor tree* means that the overview is lost and interactions with hidden parts of the tree require more effort. Focus+context approaches could introduce complexity that may worsen the experience for those with less visualization experience.

6 Conclusion

We present **T**ime**S**eries**M**aker, an open-source application to visually compose time series data and share it with others. Analysts can explore the space of possible time series combinations using a *compositor tree* that allows them to switch between different constellations of components. By visualizing both the intermediate results and their building blocks, analysts can allocate their resources to finding the right combinations and parameters. The time series created with **T**ime**S**eries**M**aker can be exported to share with others, either as raw data or as a settings file which can be re-imported to be analyzed in more detail or modified to fit a different use case. In an online study with four researchers, we found that participants liked our approach and could successfully use it to build (and share) different time series, but participants found the way of interacting with the compositor tree somewhat cumbersome. Our approach provides a basis for future work to investigate how complex requirements for advanced use cases could be integrated in an intuitive manner that does not deter analysts with simpler demands.

References

[1] W. Aigner, S. Miksch, W. Müller, H. Schumann, and C. Tominski. Visualizing time-oriented data — a systematic view. *Computers & Graphics*, 31(3):401–409, 2007. doi: 10/dxs9t9

[2] W. Aigner, S. Miksch, W. Müller, H. Schumann, and C. Tominski. Visual methods for analyzing time-oriented data. *IEEE Trans. Vis. Comput. Graphics*, 14(1):47–60, 2008. doi: 10/c2qm3s

[3] W. Aigner, S. Miksch, H. Schumann, and C. Tominski. *Visualization of Time-Oriented Data*. Springer, second ed., 2023. doi: 10/mf9f

[4] D. Albers, M. Correll, and M. Gleicher. Task-driven evaluation of aggregation in time series visualization. In *Proc. SIGCHI Conf. Human Factors in Comput. Systems*, pp. 551—-560. ACM, New York, NY, USA, 2014. doi: 10/gg2c55

[5] M. Brehmer, B. Lee, P. Isenberg, and E. K. Choe. A comparative evaluation of animation and small multiples for trend visualization on mobile phones. *IEEE Trans. Vis. Comput. Graphics*, 26(1):364–374, 2020. doi: 10/gh52ss

[6] J. Brooke. SUS: a 'quick and dirty' usability scale. In *Usability evaluation in industry*, chap. 21, pp. 189–194. Taylor & Francis, 1996.

[7] C. Bu, Q. Zhang, Q. Wang, J. Zhang, M. Sedlmair, O. Deussen, and Y. Wang. SineStream: Improving the readability of streamgraphs by minimizing sine illusion effects. *IEEE Trans. Vis. Comput. Graphics*, 27(2):1634–1643, 2021. doi: 10/ghv58n

[8] L. Byron and M. Wattenberg. Stacked graphs – geometry & aesthetics. *IEEE Trans. Vis. Comput. Graphics*, 14(6):1245–1252, 2008. doi: 10/dq8747

[9] M. Franke, M. Knabben, J. Lang, S. Koch, and T. Blascheck. A comparative study of visualizations for multiple time series. In *Proc. 17th Int. Joint Conf. on Comput. Vision, Imaging and Comput. Graphics Theory and Applications, VISIGRAPP 2022, Volume 3: IVAPP*, pp. 103–112. SciTePress, 2022. doi: 10/mf9c

[10] M. Franke and S. Koch. Compact phase histograms for guided exploration of periodicity. In *Proc. IEEE Vis. and Visual Analytics*, pp. 191–195. IEEE Computer Society Press, 2023. doi: 10/mf9d

[11] H. Hauser. Generalizing focus+context visualization. In *Scientific Visualization: The Visual Extraction of Knowledge from Data*, pp. 305–327. Springer, Berlin, Heidelberg, 2006. doi: 10/dh5vbj

[12] S. Havre, B. Hetzler, and L. Nowell. ThemeRiver: visualizing theme changes over time. In *Proc. IEEE Symp. Inf. Vis.*, pp. 115–123. IEEE Computer Society Press, 2000. doi: 10/c9hv3q

[13] J. Heer, N. Kong, and M. Agrawala. Sizing the horizon: The effects of chart size and layering on the graphical perception of time series visualizations. In *Proc. SIGCHI Conf. Human Factors in Comput. Systems*, pp. 1303—1312. ACM, New York, NY, USA, 2009. doi: 10/br9tpv

[14] R. Hyndman and G. Athanasopoulos. *Forecasting: Principles and Practice*. OTexts, third ed., 2021.

[15] W. Javed, B. McDonnel, and N. Elmqvist. Graphical perception of multiple time series. *IEEE Trans. Vis. Comput. Graphics*, 16(6):927–934, 2010. doi: 10/dvzcst

[16] B. Saket, A. Endert, and C. Demiralp. Task-based effectiveness of basic visualizations. *IEEE Trans. Vis. Comput. Graphics*, 25(7):2505–2512, 2019. doi: 10/gfw9xc

[17] M. Sips, P. Köthur, A. Unger, H.-C. Hege, and D. Dransch. A visual analytics approach to multiscale exploration of environmental time series. *IEEE Trans. Vis. Comput. Graphics*, 18(12):2899–2907, 2012. doi: 10/f4ft6c

[18] H. Song and D. Szafir. Where's my data? evaluating visualizations with missing data. *IEEE Trans. Vis. Comput. Graphics*, 25(1):914–924, 2019. doi: 10/gqnj3m

[19] A. Thudt, J. Walny, C. Perin, F. Rajabiyazdi, L. MacDonald, D. Vardeleon, S. Greenberg, and S. Carpendale. Assessing the readability of stacked graphs. In *Proc. of Graphics Interface Conf.* Victoria, Canada, 06 2016. doi: 10/gtgz8d

[20] E. Tufte. *The visual display of quantitative information*, vol. 2. Graphics press Cheshire, CT, 2001.

[21] J. Van Wijk and E. Van Selow. Cluster and calendar based visualization of time series data. In *Proc. IEEE Symp. Inf. Vis.*, pp. 4–9. IEEE Computer Society Press, 1999. doi: 10/bvpbs7

[22] M. Weber, M. Alexa, and W. Muller. Visualizing time-series on spirals. In *Proc. IEEE Symp. on Inf. Vis.*, pp. 7–13. IEEE Computer Society Press, 2001. doi: 10/drgbqf

[23] K. Wu, E. Petersen, T. Ahmad, D. Burlinson, S. Tanis, and D. Szafir. Understanding data accessibility for people with intellectual and developmental disabilities. In *Proc. SIGCHI Conf. Human Factors in Comput. Systems*, pp. 1–16. ACM, New York, NY, USA, 2021. doi: 10/gksmw6

[24] C. Xiong, J. Shapiro, J. Hullman, and S. Franconeri. Illusion of causality in visualized data. *IEEE Trans. Vis. Comput. Graphics*, 26(1):853–862, 2020. doi: 10/gh52s5