Automating Animated Transitions in Statistical Graphics

Younghoon Kim

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2021

Reading Committee:

Jefrrey Heer, Chair

Jessica Hullman

Katharina Reinecke

Daehyun Kim

Program Authorized to Offer Degree:
Paul G. Allen School of Computer Science & Engineering

University of Washington

Abstract

Automating Animated Transitions
in Statistical Graphics

Younghoon Kim

Chair of the Supervisory Committee:
Full Professor Jefrrey Heer
Paul G. Allen School of Computer Science & Engineering

People use statistical graphics (e.g., bar or line charts) to analyze data or convey insights from data. As data often involves many variables and aspects, it is inevitable to use more than one graphic. By transitioning among the charts, people relate findings across them and build up a holistic understanding of data. To facilitate this process, visualization researchers have studied effective methods for analyzing and conveying transitions, typically via animation. Indeed, animated transitions have been employed not only in research but also in public media, such as video news, slideshows, and interactive articles.

Yet, authoring animations of transitions requires significant effort: authors need to 1) understand the semantics of changes across the charts and organize them in an easy-to-follow way, 2) consider many animation designs with varied timing and techniques (e.g., staging), and 3) implement animations by manually specifying low-level details, such as selecting visualization components and assigning animation parameters.

I address these challenges by automating animation designs for a given transition. I have contributed empirical studies for a deeper understanding of animated transition designs (**Aggregate Animations**), formal representations and computational models for transitions (**GraphScape**) and animations (**Gemini**), and recommender systems for staged animation designs (**Gemini**, **Gemini2**). I evaluate the representations and models through user studies

and assess how they align with users' preferences and perceptions.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENTS

This dissertation is an accumulated work of my last six years with incredible people and communities. Personally, the six years of this journey are more than learning research; I am from a country where graduate students are forbidden to call professors by their given names. Despite these cultural challenges, during the process I have learned great wisdom while unexpectedly enjoying every moment. I want to give acknowledgments to all the people who helped my academic journey.

First and foremost, I give my special thanks to my advisor, Jeffrey Heer. Working with Jeff is always fun and inspiring. Needless to say, he taught me how to do research; He kept me on track and showed me a bigger picture that ensured I was motivated on my topics. Despite my less than perfect English communication skills, he understood my thoughts with his immeasurable patience and helped me elaborate the ideas professionally through papers, figures, and presentations. It was also a pleasure to talk about non-research topics with Jeff. Every conversation with him was warm and fun, which helped me get used to a new culture and settled in Seattle.

Next, I thank my collaborators, co-workers, and colleagues. Kanit "Ham" Wongsuphasawat helped me from the start of this journey; not only making (delightfully) fun of me to be connected with more people easily but also give invaluable advice and feedback on my research. I also appreciate all collaborators and co-authors, Jessica Hullman and Michael Correll, who worked on remarkable projects together. I could expand my understanding and methodologies on research from each of their unique perspectives. I thank all UW IDL alumni and members I have met. Even when they were busy with their work, they always shared tons of great feedback and

experiences. I feel very privileged that I can talk and discuss with these people. It is always inspiring to take excellent courses in UW and talk with UW colleagues in seminars. The inclusive feeling from the UW community helped me to sustain my work-life balance. I also thank summer internship mentors at Apple (Zach Nation) and Microsoft Research (Dave Brown and Steven Drucker). Working with these mentors in two companies was a tremendously exciting experience, which refreshed me to think about what really matters when visualizing data.

Finally, I appreciate my family in Korea and New Jersey for giving endless mental support. I also thank my Seattle friends who made happy moments with me. Due to these people, I could enjoy the whole process. Thanks to my wife, Kyoungah, and my cute dog, Gemini, for their love and support.

Chapter 1

INTRODUCTION

People use multiple statistical graphics to understand data and communicate their findings, especially when there are many interesting relationships in data. For example, people create various charts during exploratory data analysis by using visualization tools [66, 76, 77] and recommender systems [46, 71]. Interactive articles, videos, and slide shows [33, 65] convey data-related findings by using charts.

Using multiple charts involves *transitions* as people move their attention from one chart to another. Through transitions, people relate charts, drill down their exploration, or keep track of data-driven stories. Thus, to facilitate such data analysis and communication across multiple charts, transitions must be carefully presented so people can easily connect charts.

Animation techniques have been used to make transitions easy-to-follow. Visualization researchers have studied the effectiveness of animated transitions in statistical graphics [34, 38, 49, 55] and animation strategies (e.g., staggering, grouping and staging) [16, 22, 30, 73]. However, employing animated transitions requires significant effort: authors should design animations by considering varied design factors (e.g., keyframes, staging, staggering, timing) and then must implement details such as selecting visualization components and properties and assigning animation parameters.

Automating animation designs can reduce this significant effort. However, to achieve automation, two things are required: (1) comprehensive animation design guidelines for varied transitions, and (2) formal models with which machines can represent, enumerate, and evaluate transition designs. With such models, machines can systematically compose animation designs for given transitions and recommend ones based on their evaluations.

My dissertation addresses these challenges to *facilitate transitions between statistical*

graphics by automatically animating the transitions. Towards this goal, I completed four collaborative projects. I contribute empirical design knowledge for animating transitions [39] as well as new models and automation systems for animating transitions between statistical graphics [40, 41].

This thesis document begins by reviewing background and related work. From Chapter 3 to Chapter 6, I convey each of my projects. Finally, Chapter 7 concludes with directions for future work.

1.1 Contribution

My dissertation makes the following contributions:

First, this dissertation contributes programmatic models and frameworks that let machines systematically reason about transitions and their animation designs between common statistical charts. Using these models, machines can search over a combinatorial space of animated transition designs and recommend ones for effective communication. As a result, the models make transitions easier to understand using computing power.

Secondly, this dissertation provides design knowledge for animating and organizing transitions between statistical graphics. I conducted multiple human-subject studies to investigate various transitions and animation designs that are either carefully hand-crafted or recommended by heuristic score functions. These empirical studies provide data points for honing design guidelines for transitions and animation. Moreover, the study results can inform computational model to better evaluate designs and allow a deeper understanding of how human perception and cognition work on varied designs.

Lastly, yet importantly, my dissertation actively encourages follow-up work on animation design, transition reasoning, and recommendations by making related resources public. All formal models and systems are open-sourced. Future researchers and practitioners can use the contributed artifacts for their purposes, such as developing new recommender systems or authoring tools. Evaluation studies are also provided as supplementary material of published papers to be re-examined or used to corroborate future empirical studies.

1.2 Prior Publications and Authorship

My dissertation presents four published (or submitted) collaborative works which I mainly led as a primary researcher. In all four pieces, I collaborated with Jeffrey Heer as my advisor. Kanit Wongsuphasawat and Jessica Hullman co-authored GraphScape (CHI 2017), and Michael Correll collaborated for the aggregate animation project (Eurovis 2019) as a co-author. Only Gemini (VIS 2020) and Gemini^2 (submitted to VIS 2021) are completed without co-authors other than Jeffrey Heer. I use the first-person plural in each corresponding chapter to indicate the collaboration.

Chapter 2
RELATED WORK

I automate animated transition designs between statistical graphics by extending studies of animated transitions, visualization recommender systems, and computational models that systematically represent visualization, transitions, and animations.

2.1 Animated Transitions

Animated transitions between visualizations have been employed to convey changes between the start and end states of a transition; animations help people construct mental models of spatial information [8], make decisions [28], and track spatial transitions [56, 57].

Researchers have also investigated the effectiveness of animated transitions. Tversky et al. [70] raised skepticism about its efficacy. They found that prior studies were biased in favor of animation, as the animated and static conditions were unequal in their information content and interactivity. Robertson et al. compared timestep animations of scatter plots to static visualizations with traces [55]. They found that subjects preferred the animations over the static depictions though the animations resulted in lower accuracy. In a presentation context, animation resulted in faster response times. More recently, animations performed better than small multiple encodings in value comparison tasks [49]. Hypothetical outcome plots, which describe probability distributions by animating possible outcomes, outperformed static charts with aggregate uncertainty values (e.g., error bars) in inferring trends and statistical value (e.g., means and probability) [34, 38].

These evaluations led to design principles for effective animated transitions on top of existing general animation guidelines. Disney animators Frank Thomas and Ollie Johnston provided 12 cartoon animation principles [26], which inspired researchers to propose anima-

tion guidelines for user interfaces [15]. In a general context of animated transitions between information visualizations, Tversky et al. proposed two principles: *Congruence*, "the structure and content of the external representation should correspond to the desired structure and content of the internal representation.", and *Apprehension*, "the structure and content of the external representation should be readily and accurately perceived and comprehended" [70]. Heer & Robertson [30] grounded Tversky et al.'s principles into specific guidelines for statistical graphics.

In this dissertation work, I automate animated transitions between statistical graphics, having observed effectiveness by employing prior works' guidelines and conducting extra human-subject studies. I evaluate the effectiveness of carefully hand-crafted and machine-generated animation designs.

Animated transition techniques and strategies have also been developed. Heer & Robertson investigated staging, which divides changes into multiple steps [30]. They found that staged animations mostly performed better and were favored over non-staged animations in object tracking and value change estimation tasks but can also be too fast and confusing, inducing errors. Chevalier et al. [16] studied staggering techniques, which assign a temporal gap between consecutive element changes to avoid occlusion. They found staggering has no significant performance improvement in multiple object tracking. Temporal distortions (e.g., slow-in slow-out, fast-in fast-out) are also studied, and slow-in slow-out outperformed other distortions [22] in object tracking. Researchers also have developed bundling strategies that modify trajectories of sets of data points to make them seem to move together [23, 73]. My thesis lets animation authors explore these previously mentioned animation techniques, especially focusing on staging to convey complex changes in a logical way. However, inclusion of techniques such as trajectory-related strategies is left for future work, as they are only applicable to transitions within scatter plots.

2.2 Visualization Recommender Systems

Automating visualization designs has been a long-standing topic for visualization researchers. Mackinlay's APT [45] automates visualization designs in a greedy way: it encodes higher priority data fields to more effective visual variables based on perception studies [17] while excluding those that violate expressiveness principles. Subsequent recommendation engines improved flexibility by employing heuristics [46], or hand-tuned score functions [76, 77]. Scoring functions for the recommender systems can be more robust by knowledge-based models that automatically fit the scoring functions to reflect a given knowledge base (e.g., prior perception studies, heuristics) by constraint programming [42, 47]. The Gemini [40] and Gemini^2's recommender systems use a heuristic scoring model to evaluate enumerated animated transition designs by rules and functions reflecting prior animation studies. GraphScape [41] uses a mixed approach to assess transitions and sequences that compute edit operation costs from heuristic constraints through linear programming.

2.3 Formal Representation for Visualizations

Machines can compute about entities when they are formally represented. For example, 3D printers can print out a sphere by taking a corresponding 3D model of the sphere as input. This thesis provides models enabling machines to compute animations and transitions between visualizations. My thesis leverages existing visualization grammars [62, 63] to create a formal model and grammar for transitions and animations. In this section, I review the work related to the formal representations of visualizations.

Visualizations have been formally represented through declarative grammars [10, 62, 63, 66, 74, 75]. They provide several benefits. First, they allow machines to reason about visualization specifications, such as enumerating or searching visualizations. In addition, users can focus more on visual designs (specifications) by saving time for actual implementation, which compilers of the grammars do instead. Moreover, specified designs can be more robust (e.g., supporting various sizes of data and platforms) because the compilers can optimize visual-

ization workflow depending on given environments. Motivated by these advantages, Gemini provides a declarative grammar for animated transitions. Both Gemini and GraphScape are implemented with the Vega [63] and Vega-Lite [62] grammars to leverage their declarative representation. Although these works employ specific grammars, the underlying models are based on Wilkinson's Grammar of Graphics [75] (abstractions of data, marks, encodings, and guide components) so that this work can be generalized to other languages using the Grammar of Graphics, such as ggplot2 [74].

2.4 Authoring Animated Transitions

A range of tools helps users create animated transitions, each making different trade-offs between *ease-of-use* and *expressiveness*. Focusing on ease-of-use, DataClip [3] provides pre-defined animation templates, and gganimate [58] lets users animate existing ggplot charts by appending a few lines of codes. Thus, users can easily get animated graphics. However, their finite animation templates and pre-defined static visual encoding limit expressiveness. In contrast, professional animation tools, such as D3 [11] or Adobe After Effects, allow greater expressiveness but require significant effort, including writing program code to calculate and assign values for low-level components, diminishing ease of use. Between these extremes, researchers have developed declarative grammars for animated transitions, such as Canis [27]. It helps users avoid imperative programming while focusing on specifying animation designs. However, Canis still requires users to make low-level selections using W3C Selector syntax [20] (like D3). More recently, Data Animator [68] enables keyframe animation authoring with a user-friendly graphic user interface (GUI); users can import graphics from Data Illustrator [43] as keyframes and configure animation designs using timing parameters and data mapping between adjacent keyframes. Like Data Animator, this thesis targets to let users set keyframes by importing Vega-Lite visualizations. Unlike Data Animator, I focuses on helping initiate the authoring process via its recommendations instead of a GUI for a general authoring experience.

Chapter 3

DESIGNING ANIMATED TRANSITIONS TO CONVEY AGGREGATE OPERATIONS

Automating animated transitions between statistical graphics requires a substantial understanding of effective animation design. To gain such design knowledge, I investigated and evaluated animation designs for a common yet underserved class of data transformations: aggregate operations. This work was done with Michael Correll and Jeffrey Heer and published at Eurovis 2019.

3.1 Introduction

Animation can help viewers track changes and stay oriented across transitions between related statistical graphics [8, 28, 30, 57], with research to-date primarily focused on transitions in response to filtering, time steps, changing variables, or adjusting visual encodings [16, 22, 30, 55, 73]. However, visual analysis regularly involves summarizing groups of data using aggregation operations such as count, sum, and average. Sarikaya et al. [2] find that aggregation is applied in 74% of the summary visualizations in their survey. Though prior work has depicted count aggregation using a metaphor of sedimentation [36], we lack a more comprehensive treatment of statistical aggregates common to visualization practice.

Visualizations of aggregated data typically indicate the operations performed via axis titles (e.g., "Sum of Profit" or "Average Delay"). While helpful, viewers might overlook such titles or may be unfamiliar with the operations being performed. In this work, we seek to design and evaluate animated transitions that convey aggregation operations, with the goal of reducing ambiguity and imparting a better intuition for the semantics of each operation. Towards this aim, we consider eight common aggregation operations, including

both point estimates and measures of spread: *count*, *sum*, maximum (*max*), minimum (*min*), arithmetic mean (*average*), *median*, standard deviation (*stdev*), and interquartile range (*iqr*).

We first contribute novel animation designs for aggregation operations over univariate distributions, transitioning from unaggregated to aggregated dot plots. For each operation we present our design rationale, considering factors of the target concept, staging, axis scale changes, and staggering. We discuss design choices according to these factors and arrive at two staged animation designs for each operation: an *elaborate* design intended to provide a complete impression of the operation, and a simplified *basic* design with fewer animation stages.

Next, we present results from a controlled experiment evaluating these animation designs. We compare against two baseline conditions: *static* (non-animated) transitions and *interpolated* transitions that linearly interpolate between start and end states. Through an initial pilot study on Mechanical Turk, we identify which aggregation operations are most likely to be confused with each other. We then evaluate the four transition conditions with undergraduate students from a visualization course, ensuring subject familiarity with visual analysis tools such as Tableau. We ask subjects to perform binary identification tasks in which they indicate whether or not a presented transition matches a provided operation name.

Our study results indicate that staged animated transitions can improve subject's ability to correctly identify aggregation operations, though sometimes with longer response times than with static transitions. We highlight operations that benefit from *elaborate* staged transitions (such as disambiguating among *average* and *median*, or *stdev* and *iqr*) as well as those that do not (*max* and *min*). Through an analysis of participants' rankings, we find a consistent preference for staged animation designs over the baseline *static* and *interpolated* transitions. From participants' textual responses, we identify the visual features that subjects report using to make their judgments and how they vary across transition types.

Informed by our findings, we describe extensions of our design rationales to more complex charts, such as transitions to depict the construction of box plots, histograms, and means

and confidence intervals calculated via bootstrapping. We conclude with a discussion of implications and areas for future work.

3.2 Related Work

We seek to craft animated transitions of aggregation operations that are uniquely identifiable by depicting the logic of the specific operation performed. In this regard, our design goal is similar to that of *algorithm visualization*, which conveys logical processes to aid understanding. A meta-analysis of algorithm visualization in educational contexts [35] finds that they are "effective insofar as they enable students to construct their own understandings of algorithms through a process of active learning." Though we do not test educational benefits in this work, we allow subjects to play a transition as many times as they like to actively learn what is presented.

Transitions induced by aggregation operations may require more complicated designs. For unit visualizations, animated transitions conserve a one-to-one mapping between data and visual elements [51]. In contrast, aggregation transitions require visual conventions to convey many-to-one mappings of data points to an aggregate value.

Some prior projects have used animation to depict numerical aggregation. Gapminder Trendalyzer [60] uses bubble charts to representing the aggregated values (averages and sums) for geographical regions such as continents and countries. When drilling down to a more fine-grained level of detail (*e.g.,* from continent to country), each bubble is subdivided into smaller units. In a related vein, visual sedimentation [36] depicts incoming data streams and their accumulated values by employing the metaphor of a physical sedimentation process. Over time, marks representing individual data points become part of aggregate "strata" of accumulated data. Seeing Theory [21] presents interactive visualizations for basic probability and statistics concepts, including animated calculations of the mean and variance of sampled numbers. In this work, we design and evaluate animated transitions for eight common aggregation functions to transition a 1D dot plot to an aggregate value.

3.3 Animated Transition Design

We consider eight aggregation operations common to visual data analysis: *count*, *sum*, maximum (*max*), minimum (*min*), arithmetic mean (*average*), *median*, sample standard deviation (*stdev*), and interquartile range (*iqr*). In the case of *stdev*, we focus on showing the range of [-1, +1] standard deviation from the average, rather than a point estimate. For *iqr*, we calculate the quartiles using the R7 method (as used by R and Excel) [53]. Our primary goal is to design transitions for which viewers can accurately identify which aggregation operation is being performed.

We relied on Heer & Robertson's [30] guidelines for adhering to Tversky et al.'s [70] Congruence and Apprehension principles when designing our animations. We also considered four high-level design factors along which our designs might vary: *target concept*, *staging*, *axis scale*, and *staggering*. We explain our design choices for each of these factors below. Our design process began by prototyping a number of candidate designs and gathering informal feedback from members of our laboratory to inform iterative design. The candidate designs with brief comments are available in our supplement material. With the exception of *max* and *min* — which lend themselves to simpler designs — we arrived at two staged animations per operation: a *staged elaborate* version, corresponding to our full original design, and a *staged basic* version, which collapses some of the stages of the elaborate version.

3.3.1 Guiding Design Factors

Target Concept. The animations should concretely represent the concept we wish to convey. While one could imagine animations with arbitrary signifiers that would accomplish the task of disambiguation (for instance, points could turn green only when averaging, or blue only when summing), such designs would fail to convey the semantic content of the operation being undertaken. Instead, we wanted our transitions to convey at least some of the mathematical character of the operation being conveyed.

Staging. Rather than a single, complex transition, animations can be divided into a

sequence of simpler sub-transitions. Both the choice of sub-transition keyframes and timing — including pauses between stages — are important considerations. Given a short duration, an excessive number of stages may result in too many changes in rapid succession for viewers to reliably follow [30]. On the other hand, if there are insufficient stages and one simply interpolates from start to end, the target concept may not be adequately conveyed. Appropriate pauses can add emphasis and prompt consideration of keyframes, while helping avoid an overwhelming experience. Following Heer & Robertson [30], we separate major visual changes (*e.g.,* axis transitions or the rearrangement of the data points) into stages with pauses. We combine minor adjustments to color and position into single stages to reduce the total number of stages and so the complexity of the final animation.

Axis Scales. Animations should minimize disruptive changes to the axis scale. Previous research has found that changes of scale can complicate perception of animated transitions, particularly when both axis scales and data points are changing simultaneously [30]. Accordingly, in our designs we seek to minimize or, if possible, avoid changes of axis scale and always use separate stages for changes to axis scale and changes to the visualized data. To ensure fair comparisons in our evaluation, we also apply this consideration to our baseline condition that otherwise performs simple linear interpolation.

Staggering. An animation should limit the number of objects that are moving simultaneously. With the exception of *min* and *max*, aggregate operations involve the simultaneous combination of multiple data points. Animations for a single output aggregate value can thus involve the movement of many input points. A natural question, then, is how to stagger or stage such movements, while still ensuring sufficient similarity of movement to give rise to perceptual grouping (the Gestalt principle of "common fate" [50]). We employ different strategies for different aggregation operations, as described below. In addition, aggregates may be calculated separately over multiple groups. With group-by aggregation, we might animate all the groups at once or animate a single group first to ensure a single focus of attention, then synchronously animate the other groups to complete the transition.

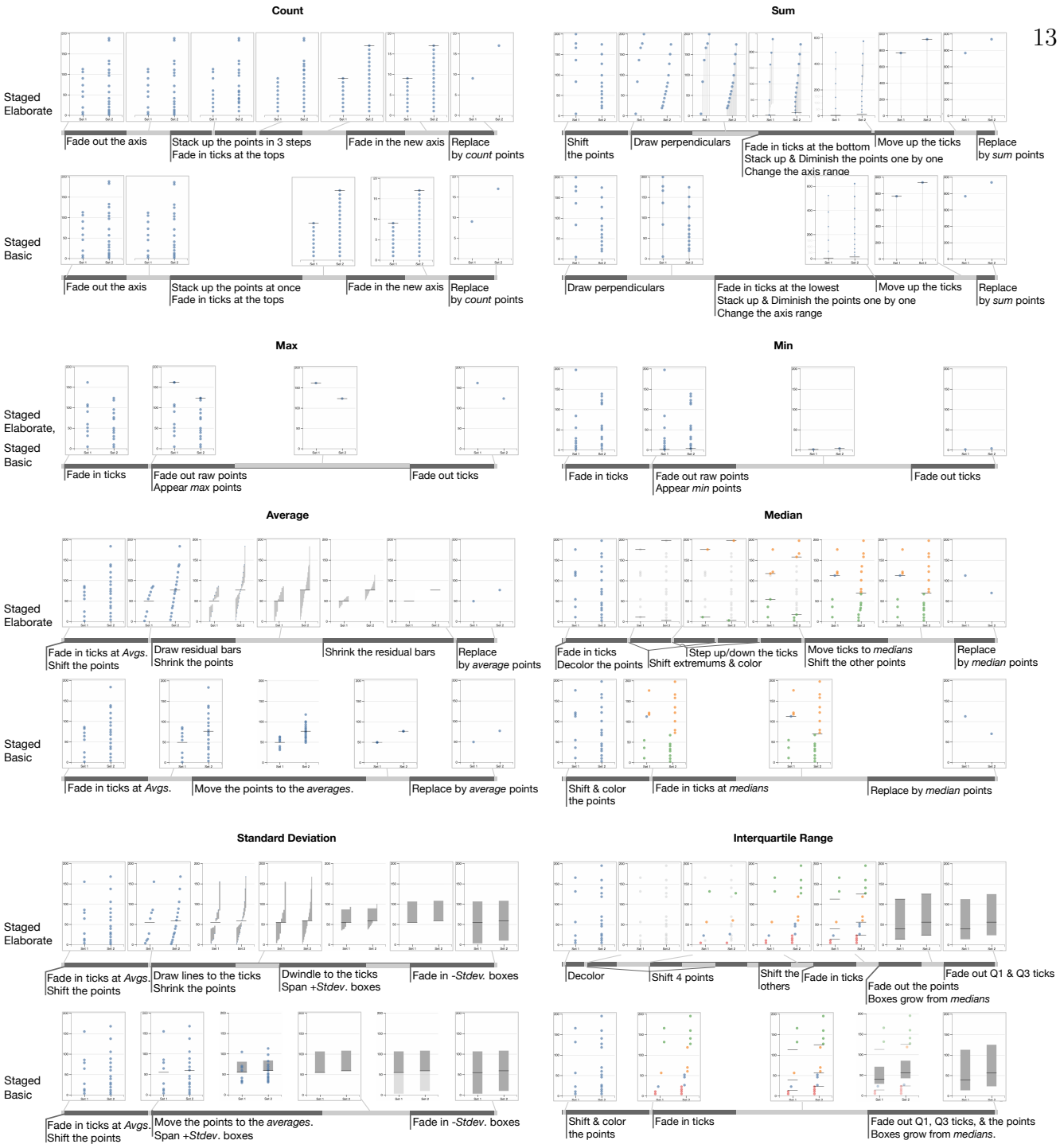


Figure 3.1: Designs of animated transitions for 8 aggregate operations with captured frames. Frames with border lines are key frames so that changes happen between them. Each grey line under each series of the frames is a timeline of the animation. The dark grey spans indicate the moves or changes of the graphic objects, and the light grey spans indicate pauses.

3.3.2 Designs

We now describe our rationales and animation designs for each aggregation operation. Our descriptions focus first on the staged *elaborate* transition, then discuss how we simplify it to produce a staged *basic* version. All transition designs are intended to be comfortably presented over a duration of 2 seconds. Illustrations and specific details on keyframes and relative timing are depicted in Figure 3.1.

Count. The *count* operation tallies the number of data records in a group. To convey this, we consecutively stack the data points, such that the final height of the stack reflects the result. The points are spaced uniformly to emphasize that each point makes an equal contribution. We separate changes to the axis and to the points by stages: the axis fades out first, the data points are accumulated without the axis, then the new axis fade in. The goal here is to avoid misinterpretation of the point positions. We subdivide the incremental stacking into three sub-stages, such that groups with more points take longer to stack, reinforcing the larger count. For the *basic* version, we instead stack all the points in one stage.

Sum. A *sum* aggregate adds up the values of the data points. To convey an accumulation, our animation design stacks up the values. To emphasize the specific values and differentiate from *count*, we first horizontally offset the points and augment them with line segments whose length encodes the data value. We then stack the lines and grow the axis scale incrementally. We do not fade out the axis, as the final axis represents the same semantic unit, just on a different scale. We also display a supplemental tick that appears at the bottom point and moves to the top as a guide. The *basic* version reduces the visual changes by skipping the horizontal offset.

Max & Min. Maximum and minimum operations have arguably the simplest target concept: they filter the data to only the highest or lowest value. Accordingly, it is sufficient for the animation to convey which extremal point is being selected and fade out the others. Doing so does not require axis scale changes or staggering. As a result of this simplicity, there

is no difference between the *basic* and *elaborate* versions for these operations. We perform only minimal staging: we first introduce a tick element to annotate the extremal value, fade out the non-extremal points, then remove the tick.

Average. The arithmetic mean or *average* is a common measure of central tendency for a distribution. One option for depicting an average could be to show how it is calculated, for example, by first animating the sum of all values followed by division by the number of records. However, this is not the only means of calculating averages (*c.f.,* online methods) and would induce potentially jarring changes of the axis scale. Instead, we convey the property that an average is the value at which the residuals (deviations from the mean) sum to zero. Movements are staggered to arrive at the average from above and below simultaneously to signify this symmetry. The *basic* design, in contrast, does not illustrate the residuals.

Median. The *median* is a value that bisects a distribution, such that half the values lie below the median and half lie above, providing a more robust measure of central tendency. Our animation depicts this process by visually segmenting a distribution into lower and upper halves, and then highlighting the median value between the two. The difference between *elaborate* and *basic* designs is the staggering when bisecting. The *elaborate* transition counts each point from the top and bottom synchronously with incremental speed to find the median; the *basic* transition divides them at once.

Stdev. The *standard deviation* is a measure showing the degree of the dispersion. Its mathematical definition involves the square root of the sum of squared residual values. This calculation can be depicted geometrically using square shapes to convey *variance*, as done in Seeing Theory [21]. Nevertheless, we abandoned this method as the two consecutive conversions (points to residual lines to variance components) was too involved and distracting. Instead, we abstract these details by starting with residual lines (initially the same as for the *average* transition) and then collapse them to form the upper range $[\mu, \mu + \sigma]$. We then fade in the lower range ($[\mu - \sigma, \mu]$) to signify that these two spans are symmetric. The *basic* version skips illustrating the residuals.

IQR. The *interquartile range* is the interval from the 25th percentile to the 75th percentile

(i.e., spanning the two inner quartiles), indicating where the central half of the data values are distributed. To emphasize the quartile boundaries, we use a similar animation as for *median*, but dividing into four groups instead of the two. After the separation into quartiles, the points fade out and a range grows from the median to inner quartile boundaries. The *basic* transition collapses stages to visualize the quartile separation all at once.

3.4 Evaluation: Identifying Operations from Transitions

To evaluate our animation designs, we conduct two controlled experiments. The first, a pilot study on Mechanical Turk, was designed to assess the parameters of our identification task, and determine which aggregate measures were difficult to disambiguate. The second, a study with undergraduate computer science students in a visualization tasks, assesses both the quantitative performance and subjective preference of students for using different transition designs to identify aggregate operations. Materials, including stimuli and data tables, are available in the supplemental material and online at `https://github.com/uwdata/aggregate-animation-data`.

3.4.1 Stimuli

We compare our staged animation designs with two baseline conditions: *static* and *interpolated* transitions. Static transitions are composed of two static graphics without any animation. Interpolated transitions linearly interpolate marks in the unaggregated charts to their final values. The data points move to the positions of their aggregated values in *count*, *sum*, and *average* operations, and simply fade out in the other operations. Axes fade out and in for *count*, while axis scales change incrementally for *sum*.

All transitions are implemented using D3.js [11] and initiated by clicking a "play" button. For static transitions, the unaggregated chart simply replaces the aggregated chart. Otherwise, the transition plays for 2 seconds. Upon completion, the play button changes to a "reset" button that, if clicked, swaps back to the unaggregated chart and allows subjects to replay the transition.

Each stimulus includes two groups of one dimensional quantitative values, randomly sampled from two different log normal distributions with distinct *average* and *median* values:

$$X_1 \sim \exp(\mathcal{N}(\mu = 5, \sigma = 3))$$

$$X_2 \sim \exp(\mathcal{N}(\mu = 5, \sigma = 4))$$

We sample differing numbers of data points ($N_1 \in \{6, 7, 8, 9\}$, $N_2 \in \{12, 13, ..., 17\}$) to ensure distinct *count* values. We also constrain the sampled distributions such that $\mu_{sample} > \sigma_{sample}$, ensuring the *stdev* range does not cross zero.

3.4.2 Pilot Study

To inform subsequent evaluation, we first conducted a pilot study using Amazon Mechanical Turk. While Mechanical Turk provides a convenient platform, we were concerned that this subject population was unlikely to match the expertise of people who regularly conduct analyses or consume analysis results. Nevertheless, we wanted to get an initial sense of subjects' familiarity with and confusion among aggregation operations. To do so, we employed both *comprehension* and *identification* tasks.

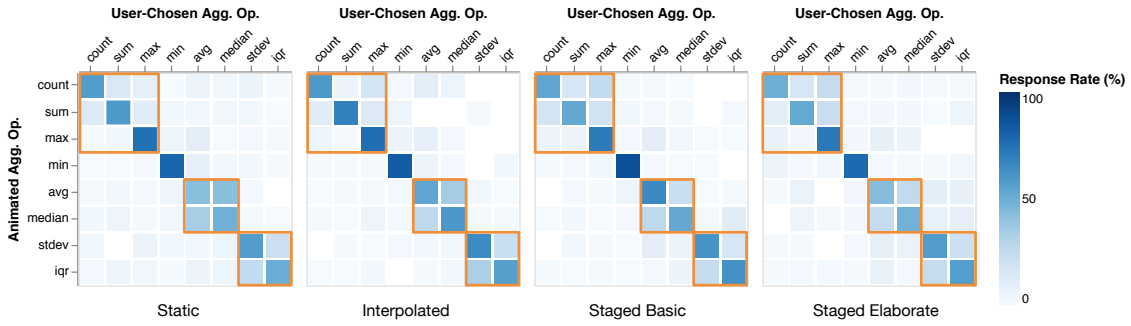


Figure 3.2: Confusion matrices for pilot study judgments of transition / operation correspondences. Values along the diagonal indicate correct responses, all others are incorrect. For each transition style, we consistently find three groups of aggregate operations that are confused with each other as shown in orange squares; $\{count, sum, max\}, \{average, median\}, and \{stdev, iqr\}$.

In a comprehension task, we first show an example transition of an aggregate operation and then show an unaggregated chart alongside four aggregated charts. The subject is asked to select the aggregated chart that correctly depicts the result of applying the aggregation operation to the unaggregated data. The three incorrect choices show aggregate values that differ by 10%, 20%, or 30% from the correct value. The primary purpose of this task is to introduce the aggregation operations and spur critical engagement from participants. After a block of comprehension tasks, participants completed a block of identification tasks, in which they are shown an aggregate transition and asked to identify which operation was performed. For both task types, participants are required to play a transition at least once before responding, but are then free to replay it as much as they like.

After completing an initial survey, participants completed 32 comprehension tasks followed by 32 identification tasks. In each case, participants saw four replications for each of eight aggregation operations, presented in randomized order. The transition style was assigned as a between-subjects variable, so a single participant saw a consistent style throughout their session.

We recruited roughly 30 participants per transition style (31 for *elaborate*) on Amazon Mechanical Turk. Based on participant self-reports, we filtered out subjects who reported color vision deficiencies or unfamiliarity with the concept of an *average*. Participants were paid $1.45 US dollars. This amount was selected using internal pre-tests indicating a 12 minute session. However, the actual average completion time was 22 minutes (for $3.95 per hour).

Figure 3.2 shows the collected responses for the eight-way identification task. From the responses we see a consistent pattern of confusion across the transition styles, with three recurring groups prone to misidentification: $\{count, sum, max\}$, $\{average, median\}$, and $\{stdev, iqr\}$. These groups correspond to commonalities among the operations: the *count*, *sum*, and *max* operations all visually select a "high" value (though in some cases only after initial transitions); the *average* and *median* both involve central tendencies; and *stdev* and *iqr* both depict ranges rather than point estimates.

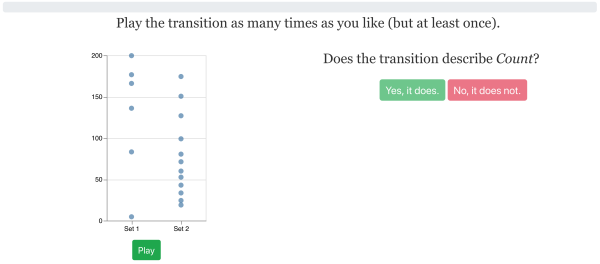


Figure 3.3: Identification task interface. Subjects view a chart transition and then indicate whether or not it matches the named aggregation operation.

Figure 3.4: Rank task interface. Subjects view each transition type for a given aggregation operation, rank order them by preference, and provide a textual rationale for their choices.

3.4.3 Main Experiment

Based on our pilot study, we developed a modified identification task, shown in Figure 3.3. To simplify the task and streamline the analysis, we adopt a two-alternative forced choice variant: given a transition and the name of an aggregation operation, the subject must indicate whether or not the presented transition matches the purported aggregate operation. Following the confusion matrix in Figure 3.2, we choose our alternatives (actual vs. purported operation) from among four clusters of "similar" operations: {*count*, *sum*, *max*}, {*min*}, {*average*, *median*}, and {*stdev*, *iqr*}, resulting in $(3 \times 3) + (1 \times 1) + (2 \times 2) + (2 \times 2) = 18$ questions. The aggregate operation is a within-subjects factor, while the transition type is between-subjects — each participant saw a consistent transition style. Subjects completed four replications for each transition / operation pair, presented in randomized order, for a total of 72 stimuli.

In addition, subjects performed a *ranking* task (Figure 3.4) that presents all transition styles for each aggregate operation and asks subjects to rank them in order of subjective preference. We then ask subjects to provide the rationale for their rank judgments using a free-form textbox and require they write more than ten words. For each of the eight ranking questions, participants could play the transitions as many times as they liked, but were required to play each transition at least once before assigning ranks.

To ensure sufficient familiarity with aggregate operations and visual analysis, we recruited computer science undergraduate students in a visualization course. All students were familiar with common visual encodings and aggregate statistics, and had experience using visualization tools including Tableau and D3. Participants were compensated with extra credit in their course.

We recruited a total of 84 subjects, and they participated through a website remotely. We filtered out four who reported color vision deficiencies and another three whose identification accuracy was below 50% (worse than chance). We also excluded two responses with completion times of more than an hour. In total, we analyzed data from 77 participants (33 female, 42 male, 2 other) distributed nearly uniformly over transition styles (20 static, 19 interpolated, 18 staged basic, and 20 elaborate).

3.4.4 Experiment Results

We analyze accuracy, completion time, and transition play count as performance measures for our identification task. We then examine participants' transition rankings and text rationales to assess user preference and visual strategies. We first investigate overall differences across transition styles and aggregate operations, then examine results for each operation in detail.

We use hierarchical Bayesian models to analyze the three performance metrics. We use weakly-regularizing priors (default choices by the `brms` library in R) and use Bernoulli, shifted log, and Poisson functions as response distributions for accuracy, completion time, and play count, respectively. Each model takes the form:

$$response \sim transition * operation + mismatch$$
$$+ order + (1|subject)$$

That is, we include the *transition* type, the named aggregation *operation*, and their interaction as fixed effects. In addition, the term *mismatch* is a binary indicator that is true when the transition and purported operation do not match, which we empirically observe leads to higher error — that is, accuracy was higher when the transition did in fact signify the aggregate operation being asked about. The *order* term is the index of the stimulus presentation order and accounts for a mild learning effect. Finally, the $(1|subject)$ term indicates a hierarchical (or *random effects*) term to capture varying performance intercepts per subject.

Qualitative responses from the ranking task were independently coded by two authors. The authors confirmed each others' codes and resolved conflicts via discussion.

Overall Performance

Across transition types, *static* transitions unsurprisingly exhibit higher play count rates across all aggregation operations (see Figure 3.5), though the strength of this effect diminishes for the simpler *max* and *min* operations. On average, subjects viewed animated transitions only once, but tended to replay *static* transitions one or more times. In terms of rankings (Figure 3.6), participants consistently prefer the staged (*basic* and *elaborate*) transitions, followed by *interpolated*, and finally *static* transitions.

Across aggregate operations, *max* and *min* exhibit the highest identification accuracy, followed by the two accumulations (*count*, *sum*), then the two central tendencies (*average*, *median*), and finally the two measures of variability (*standard deviation*, *interquartile range*) (Figure 3.5).

Comparisons of accuracy and completion times for transition styles pooled across aggregation operations should be cautious because there is the strong effect of aggregation

operation type, and transition designs are very different across the operations. Instead, we compare these performance measures for transition types separately for each aggregation operation below. Figures 3.5 and 3.6 present performance and ranking results by aggregation operation.

Count Performance

For *count* aggregation, all transition styles perform similarly in terms of accuracy. In terms of completion time, the *static* and *elaborate* transitions are faster ($\Delta_\mu \sim 1$), but with 95% credible intervals that overlap those of the *interpolated* and *basic* transitions. As shown in Figure 3.6, subjects prefer the *basic* transition, followed by the *elaborate* transition. Participants who selected the staged *basic* transition specifically praise the visual metaphor of stacked points: *"we can see the number of dots lined up nicely making it obvious there was a count"*, *"really conveys the idea of counting obects[sic] by placing points equidistant from each other signifying that we don't care about their actual value."* The *elaborate* transition is the next most preferred, though its complexity is mentioned as detracting from its effect: *"similar to the [basic], but has some useless and meaningless animation"*; *"Staged animation is confusing when all the dots look the same and move along the same line."*

Sum Performance

For the *sum* aggregation, all transitions perform similarly in terms of accuracy and completion time. The staged *elaborate* transition is the most preferred, followed by the *interpolated* transition. Participants note that the horizontal offset followed by the accumulation of lines in the staged *elaborate* transition clearly conveyed the notion of sum: *"The lines in the transition I ranked first were important in my mind because it showed that it wasn't just a count, it was summing the individual distances from 0."*; *"The way each data point had a line made it easy to see they all had contribution to the ending point. This made it clear that it was a sum."*

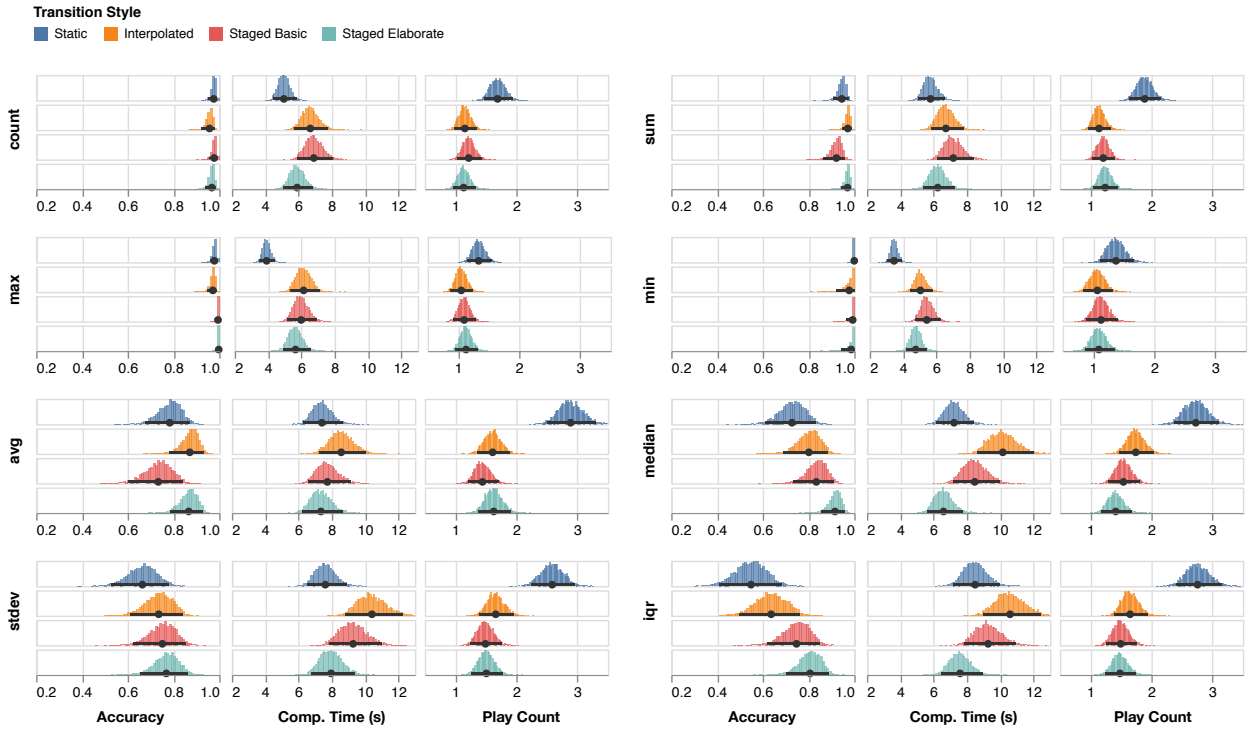


Figure 3.5: Posterior probability distributions of expected accuracy, completion time, and play count of each transition style within each aggregate operation. Overlaid points indicate means, and horizontal lines indicate 95% quantile credible intervals.

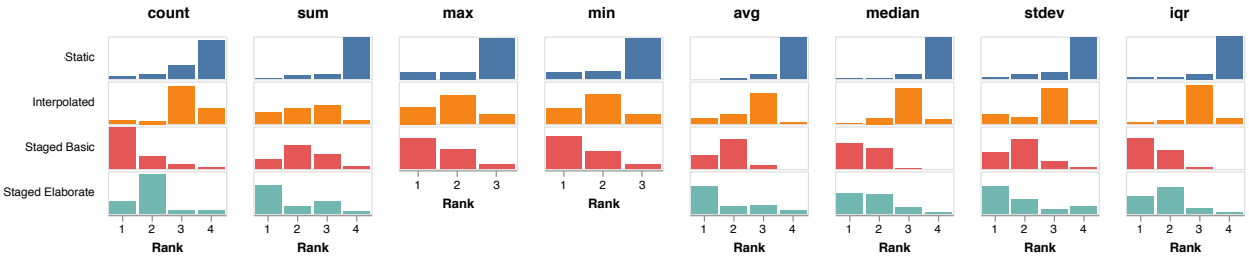


Figure 3.6: Rankings of participants' preferred transition style across aggregate operation.

However, those who prefer the *interpolated* transition thought that the collapse of the points to the aggregate value was a good visual metaphor for summation: *"the collapsing transition... made it obvious to me that it was summing"*; *"it seems to be merging the values*

which is how I usually visualize the summation."

Max and Min Performance

For the *max* and *min* operations, participants in the *elaborate* and *basic* condition saw the same staged animation. Since these two populations saw different animations for all other operations, the comparisons were slightly different, explaining the differing posterior distributions of identification task performance in Figure 3.5. Subjects were only asked to rank three animations for these operations, hence the three options in Figure 3.6. Completion time was faster for the *static* transitions ($\Delta_\mu > 1.3$), with 95% credible intervals that do not overlap any of the other transitions. Accuracy was high (mean of greater than 95%) across all conditions, although for *max*, the staged transition outperformed the *interpolated* transition.

While participants prefer the *staged* transition most often, many prefer the *interpolated* transition. The rationales reveal two conflicting opinions. Those who favor the *staged* transition find the tick annotations helpful: *"Showing the line as a new feature draws your eye to the maximum before getting rid of the data which makes it first place."*; *"Clear selection of top value by using a thick line while all points were visible"*. Those who favor the *interpolated* transition were skeptical of the utility of the animated elements in the *staged* transition, and feel that the aggregate operation is easy enough to identify without additional aids: *"I felt that no animation was necessary to show that the topmost element is the maximum of each set."*, *"Same rationale as the maximum, minimum is an easy concept to present and doesn't need that much animation."*

3.4.5 Average and Median Performance

For *average* and *median* operations, the staged *elaborate* transition meets or exceeds the others in terms of accuracy and completion time. For the *median* operation, the 95% credible interval for the accuracy of the *elaborate* transition is fully separated from that of the *static* transition ($\Delta_\mu = 0.19$). The staged *elaborate* transition is the most preferred for the *average* operation, whereas the *basic* version is most preferred for *median*. For participants who

prefer the staged *elaborate* transition for the *average* specifically mention the gray residuals as matching their conception of the target concept, helping with explainability and interpretability: *"for average, it really helps to see the 'area under the curve', to emphasize that selected line really is an average, supported by equal shaded area (by lines)"*; *"The explicit vertical lines show the different weightings by value, which is quite compelling."*

For the *median* operation, participants who prefer the staged *basic* transition like the use of color, but felt that the additional elements in the *elaborate* transition (such as the removal of points one by one), are distracting: *"The colors were incredible helpful. Of the two colored ones, the simpler one was easier to understand at a glance."*, *"The use of color to delineate elements on either side of the median is really helpful, but in this case the additional animations don't provide additional clarity. Perhaps here, color is *so* useful, that the additional steps seem a superfluous distraction."* That said, others did appreciate the per-element animation in the staged *elaborate* transition for the *median* operation, as it lined up with their internal conception of the target concept: *"The first place one is great here – does median the way i think about median, taking off values from each end of the sorted values."*, *"I like the counting motion in [the staged elaborate transition] which is how I usually visualize finding a median."*

3.4.6 Standard Deviation and Interquartile Range Performance

For *stdev* and *iqr*, the staged *elaborate* transition performs best in terms of accuracy and completion time. It has overlapping 95% credible intervals with the other conditions, with an exception for the *iqr* operation, where it exhibits higher, non-overlapping accuracy relative to the *static* transition ($\Delta_\mu = 0.26$). The staged *elaborate* transition is the most preferred for the *stdev* operation. Participants note that its depiction of the residuals of the average matches their internal target concept: *"they were more clearly calculating an average and then showing the same size range on either side of the average line."*; *"The fourth graphic's discretization makes average very clear, and the standard deviation logically follows."*

The staged *basic* transition is more often preferred for the *iqr* operation, with the staged

elaborate as the second most preferred. People who favor the *basic* transition appreciate the use of color to identify quartiles in both staged animations as a way of clearly communicating the inter-quartile range, but find the *elaborate* transition needlessly ornate: *"The 4 colors made it really easy to understand that it was splitting it into 4 quarters"*; *"I thought the [staged elaborate transition] added too much animation without it being helpful."*; *"[the staged elaborate transition] just was constant movement and made the transition a little more confusing."*

3.5 Discussion

We now discuss the effectiveness of animated transitions for aggregation operations, emphasize the importance of conveying a target concept, and extend our transition designs to additional operations.

3.5.1 Transition Effectiveness varies by Aggregation Operation

The observed effectiveness of the tested transitions strongly depends on the particular aggregation operation. The *max* and *min* operations are accurately and quickly identified regardless of transition type: while animation is preferred, it does not appear to provide performance benefits. For *count* and *sum* operations the performance distributions for each transition type overlap, but with *elaborate* and *static* exhibiting slightly shorter response times ($\sim$ 1 sec.). Participants' text responses express appreciation for design differences that disambiguate the two: equidistant stacking for *count* and summation lines for *sum*. Meanwhile, staged *elaborate* transitions exhibit benefits for conveying central tendencies (*average*, *median*) and measures of spread (*stdev*, *iqr*). In these cases, participants value the depiction of residuals (for *average* and *stdev*) and quantile segmentation (for *median* and *iqr*) to convey the target concepts.

3.5.2 Static Transitions Benefit from Replay

Unlike the animated transitions, static transitions were typically played more than once. This result suggests that participants used the replay feature to aid identification, for example by further comparing start and end states. However, in some real-world scenarios (such as presentations) it may not be feasible for viewers to replay a transition. Our results in favor of animation may thus be conservative, showing static transitions in a stronger light than may actually apply. In addition, we intentionally cue participants attention up front. In more realistic solutions, viewers may not attend to features of the start and end states in sufficient detail prior to a change of view. These observations, in conjunction with the equivalent or better performance of animated transitions, strengthen the argument for the use of appropriately-designed animations.

3.5.3 Target Concept Congruence Drives Preferences?

Our results provide evidence that user preferences are impacted by how the concept depicted by an animation relates to participants' understanding. As seen in the results and quotes above, subject rationales explain differences among ranking choices in terms of how well the animation aligns with their mental model of the aggregation operation. For the *sum* operation, the *interpolated* transition is ranked highly by some participants, who report interpreting the (upward) coalescing of points as a summation. For the *stdev* operation, multiple participants state that they prefer the *interpolated* transition because it introduces the lower and upper ranges simultaneously, whereas the staged animations show the construction for one side first, then extend it to the other. Echoing Tversky et al. [70], it appears that *congruence* between prior mental models and the animation design correlates with user preference.

3.5.4 Preferences and Potential Novelty Effects

For the most part, subject preferences either align with the performance results or at least do not contradict them. We observe a consistent preference for the staged animated transitions, then *interpolated* transitions, and lastly *static* transitions. However, we acknowledge that our results may also suffer from a novelty effect. For example, with prolonged use, viewers might prefer shorter or less elaborate transitions. Some rationales specifically acknowledge potential novelty issues: *"I like the last one with the fancy animation, but I do think the second from the left is enough for most cases"*; *"I am not sure what interquartile range with median is, but the 1st one I choose looks [like the] more complicated animation so I like it more."*; *"I like more [feature-]packed animations."*

3.5.5 Reusing Animation Designs for Additional Operations

We focused on our eight aggregation operations because they are commonly used and address ambiguities that we believe are likely to arise in practice. That said, we also look upon our animations as reusable "component" designs that can be combined with other animations to convey more complex operations. As shown in the top of Figure 3.7, we can animate the construction of a box plot by extending our *iqr* animation. A histogram can be created by applying a *count* animation, where the histogram bins define the groups (Figure 3.7, middle). As a more complex example, bootstrapped means and confidence intervals can be depicted using *average* and *stdev* animations: a sampling plus *average* animation can be applied repeatedly, followed by a *stdev* animation to form a parametric confidence interval over the bootstrapped means (Figure 3.7, bottom).

3.5.6 Limitations

Of course, our work also exhibits a number of limitations. Our examples focus on relatively small datasets, involving aggregation of only a dozen or so points per class. The designs we propose may fail to be comprehensible or suffer rendering performance degradation as the

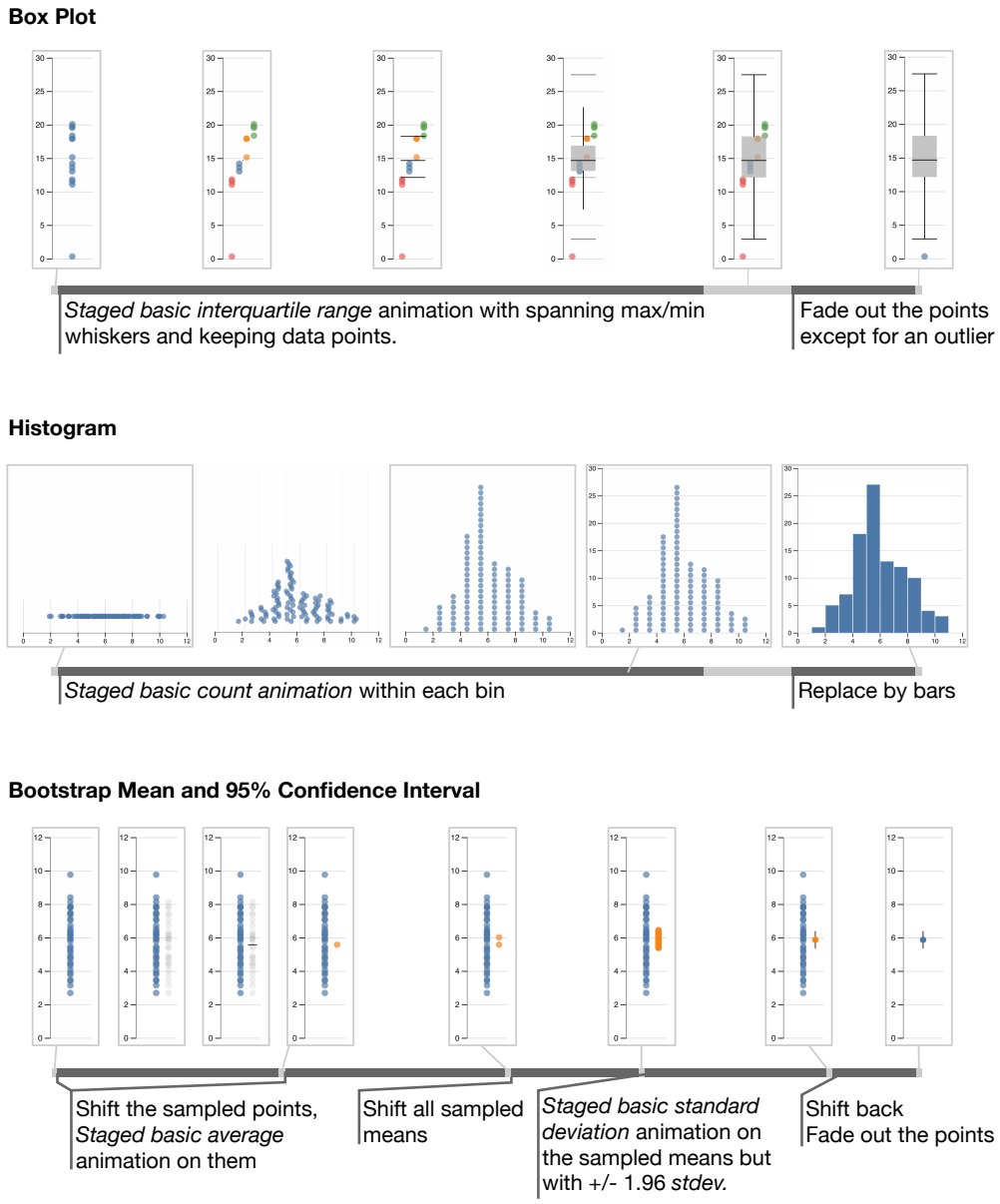


Figure 3.7: Animated transition designs for more complex aggregate operations, from a univariate dot plot to a box plot, histogram, or bootstrapped mean and confidence interval.

number of points increases. That said, we note that this critique is also applicable to existing animation techniques in the literature. Sub-sampling may be one means to address scalability issues, but requires future study and potential refinement. We also focus only on transitions that aggregate univariate data in the form of dot plots. As such we do not explicitly address

animation designs concerning other mark types, multivariate plots, or deaggregation (*e.g.,* drill-down). Nevertheless, variants of our presented techniques are applicable in other cases; for instance stacking animations for *count* or *sum* can be applied to other mark types.

This work primarily seeks to support unambiguous identification of aggregation operations, assuming basic statistical literacy on the part of the viewer. To be clear, we do not make any claims regarding the educational value of our animation designs for *learning* about aggregation operations. We leave investigation of potential pedagogical uses to future work. We expect that additional concerns, including longer playback, pauses with descriptive annotations, as well as accompanying text and graphics, might play a role. Moreover, for such cases *animation* itself may not be particularly valuable (c.f. Tversky et al. [70]), but the choice of keyframes underlying our elaborate designs might prove a useful starting point for either animations or static multi-panel explanations.

3.6 Conclusion & Future Work

We presented animation designs for eight common aggregate operations and conducted a controlled experiment to assess their effectiveness for identifying what operation is being performed. We found that our staged animated transition designs were able to meet or exceed the accuracy of static and simpler interpolated transitions for identifying measures of central tendency (average, median) and spread (standard deviation, interquartile range). In other cases (count, sum, maximum, minimum) static and animated transitions fared similarly. Participants generally prefer our staged animation designs, but in particular prefer those transitions that are congruent with their individual mental model of the operation. Our results extend existing design treatments of animated transitions and provide new evidence of animation effectiveness and how it varies with the specific operation depicted.

Still, many challenges remain. Our proposed animation designs only concern aggregate operations for position encodings of univariate data. Future work might consider techniques for other visual variables and multi-dimensional displays. In addition, techniques and effectiveness studies for large data volumes remain an open problem. For example, our *count*

and *sum* animations are not effective with many data points, leading to overplotting when stacked. Regarding evaluation, we focus on using animation to better identify which operation is being performed, and measured accuracy, completion time, and play count alongside subject-reported rankings and rationales. Eye-tracking or other physiological measures might enable a more detailed account of participants' strategies and experienced difficulties. Moreover, future design and evaluation work is needed to consider other measures, such as memorability, and other uses of animation, such as helping to teach aggregation operations to an unfamiliar audience.

Chapter 4

GRAPHSCAPE: A MODEL FOR AUTOMATED REASONING ABOUT VISUALIZATION SIMILARITY AND SEQUENCING

As we previously saw, staged animations have a great potential to convey complex changes. A next step for automating animated transitions is enabling machines to reason about the various types of changes that may occur within a transition. I present a formal model, GraphScape, to support transition analysis. This work was done with Kanit Wongsuphasawat, Jessica Hullman, and Jeffrey Heer and published at CHI 2017.

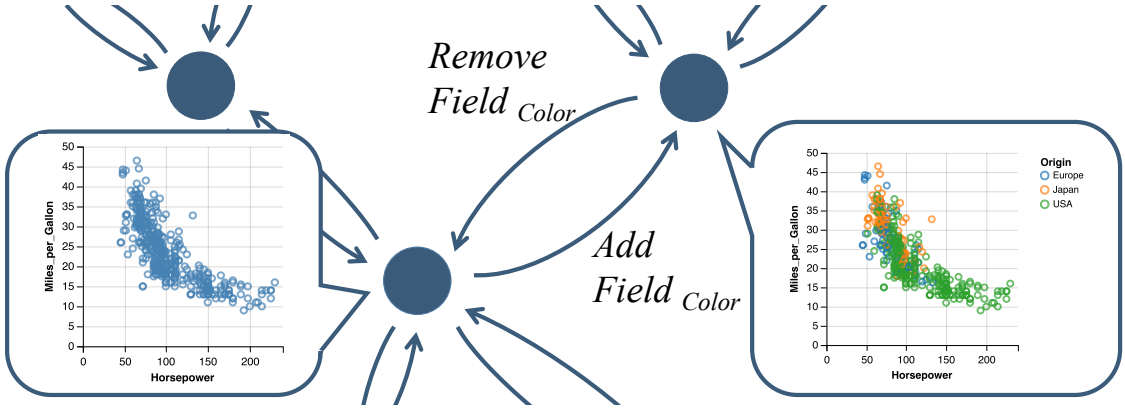


Figure 4.1: A GraphScape model. Nodes represent Vega-Lite chart specifications, edges represent edit modifications between charts. Edge weights encode estimated transition costs between charts.

4.1 Introduction

Data visualizations such as statistical charts are often viewed in sequence. Users of visualization tools may transition among plots during exploratory analysis [29, 30], view sets of

charts suggested by a recommender [71, 76], or author a presentation [33, 65] to communicate findings. To fully grasp the implications of data, visualization viewers must scrutinize not only individual charts, but also the relationships between them.

Though automated design support for *individual* charts is a long-standing topic of visualization research [45, 46, 76], less attention has been paid to modeling visualization *sequences*. In the area of narrative visualization [65], Hullman et al. [33] conduct an initial investigation of automated sequencing. They envision a graph model in which visualizations are nodes, edges represent transitions between charts, and graph distances can encode the "cost" of transitioning from one chart to another. Such formal models can provide a framework for reasoning about a visualization design space, including applications such as automatic sequence recommendation.

Here, we build on this concept to present *GraphScape*, a directed graph structure and sequence cost function for modeling relationships among charts. GraphScape supports automated sequencing, elaboration of paths between visualizations, and recommendation of visualization design alternatives. We have implemented GraphScape in terms of Vega-Lite [62], a grammar of graphics capable of expressing a variety of statistical charts. We offer four primary research contributions:

First, we define a **directed graph model** of the visualization design space, in which nodes are Vega-Lite specifications and edges are edit operations between charts. This model enables search over a visualization design space via graph traversal.

Second, we introduce a method for **estimating transition costs** between visualizations in terms of weighted GraphScape paths. We generate a principled ranking among edit operations, and encode these rankings in a linear program whose solution provides cost estimates for all atomic edit operations.

Third, we formulate a **sequence cost function** that balances pairwise transition costs and global sequence analysis. GraphScape's cost model attempts to minimize edit distances while also rewarding consistent orderings for grouped subsequences and filter terms (e.g., to promote chronological order).

Fourth, we present **results from two human-subjects experiments**. A formative study asks students in a graduate visualization course to author sequences for a set of visualizations. Subjects' sequence designs accord with our ranking of edit operations and reveal the need to promote *subsequence consistency* in addition to minimizing pairwise transition costs. The second study asks workers on Amazon's Mechanical Turk to rate a set of sequences in terms of how well they convey the data in a clear and logical manner. We find that subject preferences strongly correlate with GraphScape's cost model.

Finally, we demonstrate applications of GraphScape: recommending sequences, elaborating transition paths, and suggesting design alternatives (e.g., given an input chart, find the "nearest" designs that are scalable to larger data volumes). We conclude with a discussion of future research directions.

4.2 Related Work

GraphScape extends prior work on modeling visualization state spaces.

4.2.1 Modeling Visualization State Spaces

Multiple projects have proposed models of visualization state spaces, often in order to analyze user exploration. The P-Set [37] and Image Graph [44] models formally represent a visualization using a parameter vector, and treat user exploration as a graph structure in which edges correspond to parameter value changes. To support bookmarking and sharing, the sense.us [31] collaborative analysis environment similarly models each state of an interactive visualization as a parameter set. In these models, the parameter sets may be idiosyncratic and vary across visualization types. In contrast, GraphScape provides a generative model for reasoning about a space of charts expressible within a grammar of graphics.

VisTrails [12, 64] analyzes visualization workflows in order to track provenance and automatically suggest operations based on prior activity. Tableau graphical histories [29] record user activity, representing states as specifications in the VizQL language. User explorations form a tree structure in which each state is a node. This structure can be visualized to aid

revisitation or analyze usage patterns. GraphScape similarly models relationships among visualization states in a graph structure, but GraphScape edges encode edit operations to transition from one state to another, not observed user behavior.

Research on animated transitions [30] examines the relationship between statistical graphics to design animations that convey *semantic* changes between charts, including changes to data transformations and encoding channels. We consider semantics to rank transitions relative to the degree that they modify a chart's meaning. As we later demonstrate, GraphScape can elaborate paths between visualization states, for example to generate staged animated transition plans.

The most relevant prior work is Hullman et al.'s study of visualization sequence [33] for narrative visualization. Similar to GraphScape, Hullman et al. model the visualization state space as a directed graph, with edges encoding transitions among charts. GraphScape extends this work in multiple ways. First, the prior work considers only a subset of transition types and is conceptual in its treatment. We contribute an actionable model implemented using the Vega-Lite language, and demonstrate applications of its use. Second, we contribute a more sophisticated cost function that balances both local and global sequence features. We describe a method for deriving edge weights (to model the difficulty of interpreting a new chart given a previous chart) and a sequence weighting term that promotes subsequence consistency. Finally, we present the results of two experiments investigating how users author and rate visualization sequences.

4.3 Comparing Chart Transition Types

To formulate our model, we began by defining a taxonomy of specification edits that captures possible transitions among Vega-Lite unit visualizations. We performed triplet comparison judgments [81] to gauge relationships among edit operations. This process produced a set of inequalities among edit operations that induces a ranking of chart transitions in terms of perceived "cost" or complexity. We then use this ranking to formulate a linear program, whose solution provides numerical cost estimates for each operation. To test and refine our

ranking, we also conducted a formative experiment in which we gave students in a graduate visualization course a set of charts and asked them to sequence the charts in a manner that communicates the data most effectively.

Category	Edit Operations	Description	Example
Mark	$Mark_A$-$Mark_B$	Change the mark type from A to B, or vice versa.	Change Bar marks to Line marks.
Transform	*Scale*	Add, change or remove an axis scale transform.	Apply a log scale on the x-axis.
	Sort	Change the sort order of a visualized data field.	Sort 'Maker' on x-axis in descending order of 'Mean Price'.
	Bin	Discretize values of a visualized field into bins.	Bin 'Price' into 10-unit-wide bins.
	Aggregate	Aggregate values according to an aggregation function such as sum, mean or median.	Aggregate 'Price' to mean.
	Modify Filter, *Add / Remove Filter*	Filter data records according to a filter predicate function.	Filter 'Price' values $\leq$ 100.0. Keep 'Maker' values equal to 'Company A' or 'Company B'.
Encoding	*Transpose*	Swaps the (*x, y*) or (*row, column*) channels.	Transpose the x- and y-axes.
	Move	Move a field on one channel to another channel.	Move 'Price' from x-axis to y-axis.
	Add / Remove	Add or remove a field from an encoding channel.	Add 'Price' to x-axis. Remove 'Price' from x-axis.
	Modify	Replace a field in a channel with another field.	Replace 'Price' with 'Maker' on the x-axis.

Table 4.1: Edit operations for transitions among visualization specifications. The table is sorted in ascending order of estimated pairwise cost.

4.3.1 Step 1: Identifying Edit Operations

Informed by prior studies of visualization transitions [30, 33], we constructed a taxonomy of possible edits to Vega-Lite [62] unit specifications. We identified atomic editing operations that, when combined, can transform any Vega-Lite unit visualization into another. Examples include changing the mark type (e.g., from bar to line marks), applying a log transform to an axis, deriving aggregate statistics, and assigning data fields to visual encoding channels such as position, size, shape and color. These edit operations form a directed graph in which visualization specifications are the nodes and the edit operations are edges defining potential

transitions among charts. We further group these transitions into mutually exclusive categories of *mark type*, *data transformation*, and *visual encoding* transitions. Edit operations and categories are listed in Table 4.1.

Encoding operations such as *Add Field*, *Move Field*, *etc.* permit a large number of combinatorial possibilities relative to the available encoding channels and data fields. A field may be added or removed from the x channel, the *color* channel, and so on. *Filter* transformations also take additional parameters, in the form of filter predicates. Later in this chapter we discuss how our cost model accommodates filter parameters.

4.3.2 Step 2: Ranking Edit Operations

Given a set of edit operations, we next sought to rank these operations in terms of approximate interpretation difficulty (in other words, how hard it is to interpret a target visualization given a source visualization). Our goal was not to derive precise estimates of user behavior, such as response time or cognitive load measures, but rather find a suitable ordering of edit operations to support ranking of visualization sequences.

Due to the large combinatorial space, manually ranking all pairs of edit operations is infeasible (even ignoring issues such as the choice of data set or visual context). To prune the space, we leverage assumptions about transition categories: we assume that mark type transitions are less costly than data transformation transitions, which in turn are less costly than visual encoding transitions. Our rationale stems from the semantics of each transition type: changing the mark type holds all data constant, data transformations retain the same backing data fields but can change the level of summarization or the set of data points visualized, and visual encoding transitions fundamentally change what data fields are being shown. Each involves more substantial modifications to what the visualization conveys, presumably entailing more interpretative effort.

We then investigated how to rank order edit operations within each transition category. Following the perceptual kernels method of Demiralp et al. [81], we formulated a set of *triplet comparisons*: given a source visualization and two target visualizations, which target

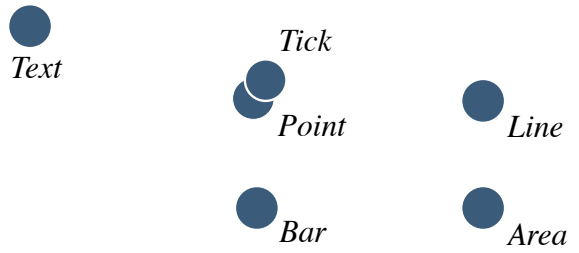


Figure 4.2: 2D embedding of mark type comparisons. The distance between mark types indicates the estimated transition cost between them.

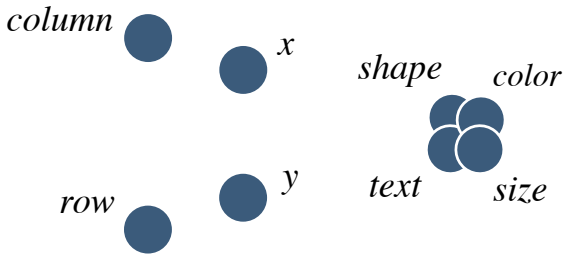


Figure 4.3: 2D embedding of encoding channel comparisons. Distance indicates the estimated cost of *Move Field* operations between channels.

visualization is easier to interpret in the context of the source? For example, given a bar chart, which is easier to understand: an area chart of the same data or a line chart? We conducted self-experiments to collect triplet judgments involving edit operations within each transition category. We then used Generalized Non-Metric Multidimensional Scaling (GNMDS) [1] to produce metric-space embeddings.

Mark Types. Figure 4.2 shows an embedding of mark type comparisons. The embedding exhibits symmetries for discrete (*point*, *tick*, *bar*) vs. continuous (*line*, *area*) marks as well as filled shapes (*bar*, *area*) vs. point embeddings (*point*, *tick*, *line*). *Text* marks are distinctly placed, with *point* and *tick* marks as nearest neighbors.

Data Transformations. We constructed a similar embedding for data transformations, finding that axis *scaling* (e.g., log scale) led to the smallest distances, followed by *sorting*. These transforms distort or rearrange a chart without changing the data. Meanwhile, *bin* and *aggregation* transformations lie along a 1D spectrum in the embedding, ranging from raw data, to binned data (e.g., histograms), to point aggregates such as means and medians. We judged *filter* operations to be the most costly, as they modify which data is included in a chart.

Visual Encodings. Figure 4.3 shows an embedding of encoding channels for *Move Field*

operations, which move a data field from one encoding channel to another. Spatial channels (x, y position; subdivision by *row* or *column*) exhibit expected symmetries. The *color*, *shape*, *size*, and *text* channels form a cluster exhibiting smaller distances among each other. For all other encoding operations (*Add Field*, *etc.*) the underlying inequalities are $x, y > row, column > color > size > shape > text$. This ranking assumes that changes to spatial encodings require more interpretation effort, or, conversely, that it is easier to interpret a sequence of charts with a stable set of axes. The *row* and *column* channels are considered cheaper than x and y to favor consistent sub-plot structure in the face of changing trellis subdivisions, rather than vice-versa. Our model does not permit assignment to *row* or *column* channels if the corresponding x or y channel is unassigned. This scheme ensures that simple plots take precedence over trellis plots and does not incur any significant expressivity limits, as assigning to *row* or *column* with no corresponding x or y assignment is nearly equivalent to assigning a discrete field to x or y directly.

Comparing encoding operations, our triplet judgments produce the ranking *Transpose* > *Move* > *Add*, *Remove* > *Replace*. *Transpose* was deemed least costly, as it swaps spatial positions only. *Move* similarly preserves the set of data fields being visualized, though can involve changes of encoding channel. *Add* and *Remove* are treated as equally costly. Finally, *Replace* operations both add and remove a data field in one operation, and are thus more complex than a single add or remove. When comparing two operations of the same type (e.g., two adds), we apply the encoding channel rankings described above.

4.3.3 Step 3: Deriving Edit Operation Costs

Our triplet judgments and resulting embeddings provide reasonable initial models of difference within transition categories, but forming a general cost model presents multiple challenges. First, how should the results for different categories be combined? How should distances among mark types be scaled relative to distances among data transformations? Second, metric embeddings do not preserve additive distances. Within a 2D embedding the distance of applying $Edit_A$ followed by $Edit_B$ is measured as the (L2) Euclidean distance

in the embedding space, not the (L1) sum of distances. An embedding-based cost model might thus "cut corners" in undesirable ways. Third, by construction metric embeddings generate *symmetric* distances. However, we would like a model that is capable of expressing *asymmetric* transition costs, as it may be the case that $T(a, b) \neq T(b, a)$.

These issues led us to consider alternative analyses of the triplet judgments. Each judgment expresses an inequality among two edit operations (e.g., *Sort* < *Add Filter*). We can encode these inequalities in a single linear programming [18] problem. The solution to this linear program is a set of cost estimates for *all* edit operations. This solution also preserves additive (L1) distances and permits asymmetric costs.

Recall that linear programming solves for a vector x given problems of the form: minimize $f'\,x$, subject to $Ax \leq b$ and $x \geq 0$, where f is a given vector of coefficients, and matrix A and vector b encode linear inequalities.

Consider an example with two edit operations *add* and *rem*. Assume all edits have positive, non-zero cost and $add < rem$. We can encode this as the linear program $-add \leq -1;\ -rem \leq -1;\ add - rem \leq -1$. Or, in matrix form:

$$A = \begin{bmatrix} -1 & 0 \\ 0 & -1 \\ 1 & -1 \end{bmatrix} \quad b = \begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix} \quad f = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

The -1 terms in the b vector enforce a minimum cost of 1, while f specifies that the solution should minimize the sum of all costs relative to the inequality constraints. The solution for x consists of the cost estimates for *add* and *rem*.

We use this encoding approach to learn costs for GraphScape, using the edit operations and inequalities described above. We augment these judgments with additional inequalities that encode our category rankings (*mark type* < *data transformation* < *encoding operations*). We require that the sum of all costs in one category (e.g., *mark type*) be less than the cost of any single operation in a more costly category (e.g., *data transformation*). Our complete set of inequalities and code for generating linear programs (solvable via MATLAB's *linprog* function) are included as supplemental material.

4.4 Empirical Study of Edit Operation Rankings

Our initial modeling efforts produced a set inequalities among edit operations, gathered via self-experimentation. To further test and refine our model, we conducted a controlled experiment in which 51 students (19 female, 31 male, 1 declined to state) in a graduate visualization course manually sequenced sets of charts. We asked subjects to order the charts in a manner that most clearly and effectively communicated the data. Subjects were compensated with extra course credit.

Each subject sequenced a total of seven chart sets: four included a *fixed* initial chart to force subjects to decide between two difficult options; the other three permitted arbitrary placement of all charts. Each set was constructed to test specific modeling questions, as described below. All chart sets are included in supplemental material. To assess statistical significance, we used chi-squared tests of categorized counts of subject responses. Where subject responses involved only two possible choices, we also ran binomial tests; these provide identical significance results and are not reported here. Results of a post-hoc power analysis that confirms sufficient power for detecting differences between frequencies of sequence types are included in supplemental material.

1. Add vs. Remove Fields. This set contains three charts of college admissions data: the first (fixed) chart shows total numbers or accepts and rejects by gender, the others (1) subdivide the plot into columns by department and (2) remove gender as a color-encoded field. There are two possible sequences: a sequence that adds one field and then removes two fields, or a sequence that removes one field and then adds two. Is the cost of adding a field different than the cost of removing a field? Subject preferences are split between multiple removes (28/51) and multiple adds (23/51), $\chi^2(1) = 0.4902$, $p = 0.484$. As a result, our model treats add and remove field operations within an encoding channel as having equivalent cost.

2. Mark Type vs. Scale Transform. This set contains three stock charts: a first (fixed) line chart, an otherwise identical scatter plot (with mark type *point*), and a log-scaled

line chart. Here, we sought to compare mark type changes and data transformations: one possible sequence involves two mark type changes and one log scale transform, while the other instead involves two scale changes and one mark type change. Here, subjects roundly preferred two mark type changes (44/51) over two scale changes (7/51), $\chi^2(1) = 26.84$, $p < 0.001$. This result provides corroborating evidence for ranking mark type changes as less costly than data transformations.

3. Filter vs. Transpose. This set contains three charts with data about automobiles. The first (fixed) scatter plot depicts acceleration vs. displacement for a set of European, Japanese and American cars. The other charts (1) transpose the first plot and (2) filter the first plot to show only European cars. Here our goal was to compare the perceived cost of filtering a view (a data transformation) to transposing a view (a minimally disruptive encoding change). Subjects preferred the sequence with two filter modifications (37/51) over the sequence with two transpositions (14/51), $\chi^2(1) = 10.37$, $p = 0.001$. This result supports our decision to consider data transformation transitions less costly than encoding transitions.

4. Add vs. Remove Filter. This set contains three charts showing unemployment data across industries. The first (fixed) scatter plot shows unemployment data for three industries over the period 2005–2010. The other charts (1) filter to show manufacturing only or (2) expand the time period to 2000–2010. This example tests whether users exhibit preferences regarding adding vs. removing filters. We found no significant preference across the two possible sequences (28/51 vs. 23/51), $\chi^2(1) = 0.4902$, $p = 0.484$. As a result, our model does not include asymmetric costs for adding or removing filter terms.

5. Roll-Up vs. Drill-Down. This set contains four charts showing IMDB ratings for movies from the years 1940–2010. One chart plots all films, while others show average rating by year or by decade. A final chart additionally filters to show average ratings by year within the "Horror" genre. We designed this trial to assess subject preferences for specific-to-general (e.g., "roll-up") or general-to-specific (e.g., "drill-down") transitions. We found a preference for starting from the entire data and introducing increasing levels of summarization (26/51), as opposed to drilling-down (11/51) or alternating between levels of abstraction (14/51),

$\chi^2(2) = 7.412$, $p = 0.025$. In the case of alternating levels, we saw examples where subjects start by showing the raw data, then jump to the highest level of summarization and drill-down. The results exhibit a fair degree of individual variation, and different presentations might benefit from different narrative strategies. Still, the results support a default strategy of starting from instance-level data and building up to aggregate displays.

6. Edit Minimization. This set consists of four charts showing US population data by gender and age in both 1860 and 2000. The set includes a stacked bar chart and three trellis plots; two charts have the y-axis for age increasing from top-to-bottom and two have it increasing from bottom-to-top. Our goal here was to test if users would prefer sequences that minimize the number of edits across each frame. One possible sequence changes only one aspect at a time, others require multiple changes (combining a visual encoding, axis direction, and/or filter changes). A strict analysis finds a marginally significant preference for minimizing edits (32/51), $\chi^2(1) = 3.314$, $p = 0.069$. If the analysis is relaxed to ignore changes in axis direction, the result strengthens (44/51), $\chi^2(1) = 26.84$, $p < 0.001$. Our interpretation is that subjects prefer to reduce the number of edits at each step, but are less concerned with a change in axis direction. This result supports our strategy of limiting pairwise transition costs. Subject responses additionally show a strong preference for chronologically ordered charts (42/51), $\chi^2(1) = 21.35$, $p < 0.001$.

7. Subsequence Parallelism. This set contains six scatter plots of horsepower vs. weight for a data set of cars. The plots alternate between showing data for all cars in the database, only European cars, and only American cars. Three plots show data from 1976, while three show data from 1980. This trial was intended to test whether subjects strictly minimize pairwise transition costs or favor consistent parallel subsequences. For example, subjects might consistenly order charts filtered by region within each year (parallelism) or could use alternate orderings that minimize the total number of edits (see Figure 4.4 for an example). We observe a clear preference for parallel structure as opposed to aggressively minimizing pairwise transition costs (36/51), $\chi^2(1) = 8.647$, $p = 0.003$. In agreement with prior work [33], these results suggest that using consistent transitions within groups of re-

lated views (parallel structure) may be preferred to organize a sequence, involving analysis of sequence structure beyond pairwise relations alone.

The experimental results align with our modeling choices or, in cases where no significant differences are found, suggest that our ranking resides well within inter-subject variation. In addition, task 7 reveals a critical requirement for sequence cost models: preserving parallel structure may be preferable to minimizing the edit distance at each step of a sequence. Naïve minimization of pairwise costs may thus overfit. In the next section, we present our full sequence model, which includes both pairwise transition costs and a global sequence weighting term that rewards consistency among subsequences.

Of course, our study has limitations. We did not exhaustively test all edit combinations. Instead, we created visualization stimuli to represent a set of specific combinations where we suspected systematic preferences might exist, but where we were uncertain about the ranking.

4.5 The GraphScape Model

GraphScape consists of a *directed graph* that models the visualization design space in terms of editing operations between chart specifications, and a *sequence cost function* that assigns a numerical score to an ordered set of chart specifications.

4.5.1 Directed Graph of Visualization Specifications

We formally define a $GraphScape = (V, E, D)$ as a directed graph of visualizations, where nodes $v \in V$ are Vega-Lite unit chart specifications [62] and edges $e \in E$ are edit operations that turn one specification into another (illustrated in Figure 4.1). The possible edit operation types are listed in Table 4.1.

A GraphScape is defined relative to a relational data table D. For any node v, incident edges are determined by the data set and the encoding channels used in v. For example, only data fields $f \in D$ can be added to a visualization, $Remove\,Field_{f,x}$ can not be performed if the x encoding channel is empty, and $Add\,Field_{f,y}$ can not be performed if a field is already

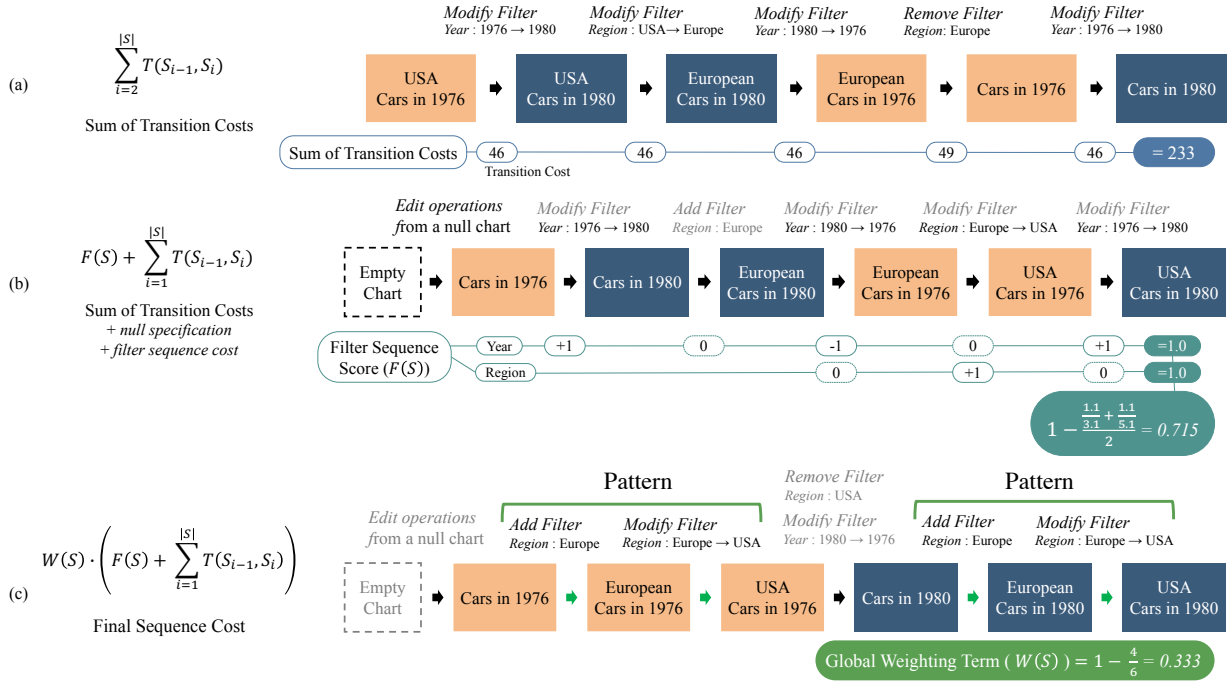


Figure 4.4: GraphScape sequence cost calculation. (a) Lowest-cost sequence when minimizing the sum of transition costs only. (b) Lowest-cost sequence after adding an initial *null specification* and filter score. (c) Lowest-cost sequence with a global weighting term to reward consistent subsequences.

present on the y channel. Given all possible combinations of data fields, edit operations, and encoding channels, each node v may have many incident edges. As a result, we typically do not materialize a full GraphScape model, but rather dynamically enumerate valid edges when traversing the model.

4.5.2 Transition Costs

Each GraphScape edge includes an edit cost $w(e)$, as determined by our linear program. We simply lookup an edge's edit operation type and return the corresponding cost estimate. More generally, we define the transition cost $T(u, v)$ between visualizations u and v as the

sum of edge weights along the shortest (lowest weight) path between them:

$$T(u,v) = \Sigma_{e \in ShortestPath(u,v,w)}\ w(e)$$

We alter this formula slightly in the case of encoding operations involving the *count* aggregation function, which is extremely common as part of summary plots (e.g., for categories and histograms). We discount the cost by ignoring the contribution of co-occurring *Bin* or *Aggregate* edits. For example, $Add\,Field_{count,x}$ incurs only the cost of an *Add Field* operation, ignoring the *Aggregate* data transformation cost.

4.5.3 Filter Sequence Costs

Filter operations are expressed as a conjunction of predicates. The supported predicates are *equality*, *range*, and *set inclusion* tests. For example, the filter $range(Price, [100, 200])$ includes a data point if the 'Price' field value lies in the range 100–200. As they are data-dependent, predicate terms are *not* accounted for by our linear program, and sequences with different filter operations may incur identical transition costs.

To account for different orders of filter operations, our cost model analyzes filter modification sequences. We focus on *equality* predicates, which are common during visual analysis (e.g., to view data for individual years or category values). Whereas trellis plots (*row* or *column* channels) subdivide data over space, sequential filtered views subdivide data over time. We leave *range* and *set inclusion* predicates for future work.

We compare data values within *equality* predicates and reward sequences with values in sorted order. We favor ascending over descending order, which promotes alphabetic (for categorical data) and chronological order (for dates). We first identify all recurring filtered fields within an input sequence S, and extract a set V consisting of per-field sequences of predicate values v_i. Our filter cost term F is given by:

$$F(S) = 1 - \frac{1}{|V|} \sum_{v \in V} \frac{|\sum_{i=2}^{|v|} d(v_{i-1}, v_i) + 0.1|}{|v| - 1 + 0.1}$$

$$d(v_a, v_b) = \frac{v_a - v_b}{|v_a - v_b|} \text{ if } v_a \neq v_b,\ 0 \text{ otherwise}$$

For each filtered field we score ascending and descending value changes as +1 and -1, respectively. We compute the absolute value of the sum of scores for each field, normalized by the number of filter changes. We include an additional 0.1 to bias the score in favor of ascending order. A sorted ascending order will receive the highest possible score, and a sorted descending order will receive a slightly lower score. Orders with both ascending and descending changes will cancel each other out, leading to low scores. We then take the average of all filter scores, and subtract it from one to convert the score to a cost. The result provides a fractional offset for the total sequence transition cost. This calculation is illustrated in Figure 4.4.

4.5.4 Global Weighting: Rewarding Consistent Subsequences

Both prior work [33] and our formative experiment find that people prefer chart sequences that are grouped and sorted in a consistent order, even if this does not perfectly minimize the edit distance. Accordingly, our cost model includes a global weighting term W to reward sequences that order charts into consistent, parallel subsequences.

We define a pattern P as a repeated, non-overlapping subsequence of identical transitions. The global weighting term W is then determined by the fraction of the sequence S covered by the pattern P that induces maximal coverage over S:

$$W(S) = 1 - \max_P \frac{count(P, S) \cdot |P|}{|S|}$$

For example, in Figure 4.4(c), P contains two transitions ($|P| = 2$) that add and then modify a filter. This pattern occurs twice ($count(P, S) = 2$) within a sequence of length $|S| = 6$.[1] The global weighting term is thus $W(S) = 1 - 2 \cdot 2/6 = 1/3$.

[1]The normalizing term $|S|$ (as opposed to $|S| - 1$) accounts for the initial *null specification* transition described in the next sub-section.

The $W(S)$ term reduces the final sequence cost by a multiplicative factor that reflects the number of consistent subsequence transitions. If no repeating pattern P exists, we set $W = 1$ and the sequence cost is unchanged.

4.5.5 The GraphScape Sequence Cost Function

Bringing each component together, the GraphScape cost function for a visualization sequence S is:

$$Cost(S) = W(S) \cdot \left(F(S) + \sum_{i=1}^{|S|} T(S_{i-1}, S_i) \right)$$

To account for the cost of interpreting an initial chart within a sequence, we treat each input sequence S as having an initial entry S_0 consisting of a *null specification* in which no encoding fields are specified. Adding this entry ensures that transition costs for building up the initial chart are included in the cost calculation. This step also biases the cost function in favor of specific-to-general transitions that start with unaggregated data and build up to summary visualizations, a preference observed in our formative study.

4.6 Evaluation: Sequence Ratings

We conducted an experiment to measure how the GraphScape cost model aligns with user preferences for chart sequences. In each trial, subjects viewed five different sequences of six charts and rated how well each sequence presented the data in a clear and logical manner using a 5-point Likert scale. Each set included the top-scoring sequence according to GraphScape, along with others chosen to cover a set of ordering strategies. We hypothesized that, by balancing both local transition costs and subsequence consistency, GraphScape's sequence rankings would strongly correlate with subjects' ratings.

4.6.1 Experimental Design

Participants. We recruited 55 participants (29 female, 26 male) on Amazon's Mechanical Turk, each located in the U.S. and with a HIT approval rate $\geq 95\%$. Each received \$6.50 USD

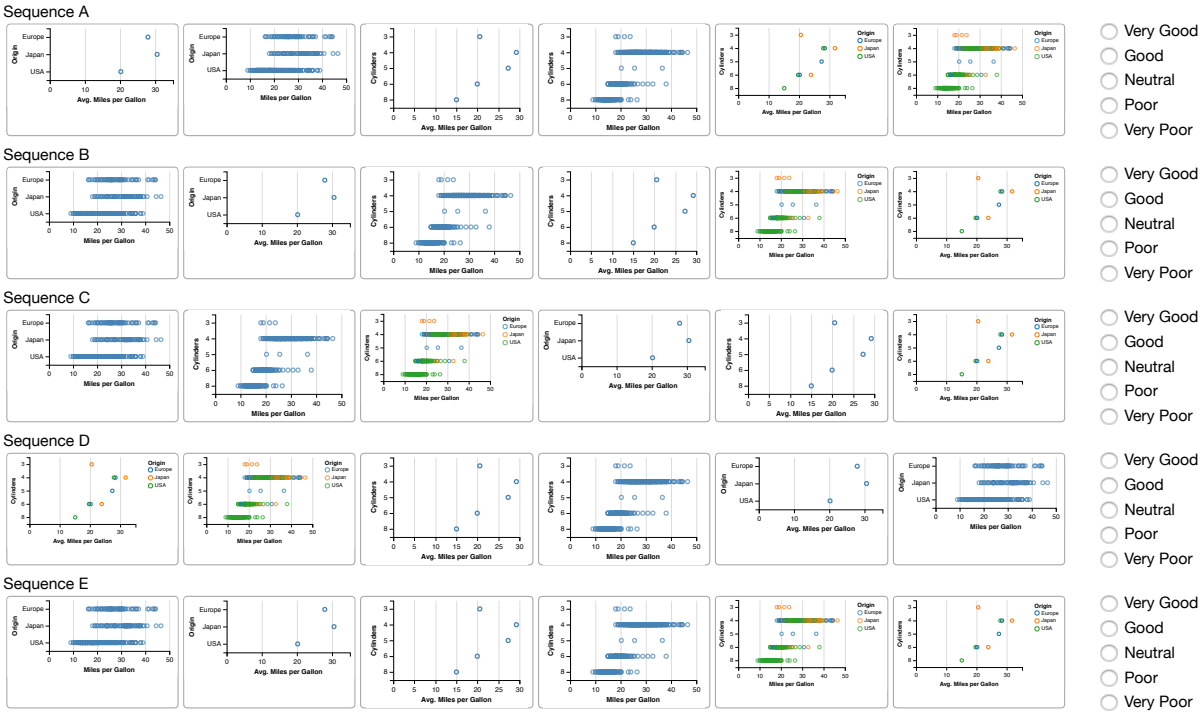


Figure 4.5: Interface for the sequence rating experiment. Subjects rated how well each sequence presents the data in a clear and logical manner. Sequences were randomly assigned to table rows. Here, sequence B receives the highest rankings from both subjects and the GraphScape cost model.

in compensation. Among the participants, 6 (11%) reported having a graduate or professional degree, 28 (51%) a bachelors or associate degree, 15 (27%) some college coursework, and 6 (11%) a high school diploma. No subjects reported vision impairments. Participants took an average of 41 minutes (s.d. 19 minutes) to complete the study.

Stimuli. Subjects performed 6 trials. In each trial, subjects were shown a set of 6 charts and 5 possible sequences of those charts. We designed the sets to cover the transition styles proposed by Hullman et al. [33] and observed in our formative experiment. These styles include measure walks (viewing different quantitative measures against a fixed set of categories), dimension walks (viewing one or more fixed measures against a rotating set of

categorical fields), filter walk (changes of filter values), consistent subsequence orders (parallelism), chronological ordering, and coverage of mark, transform, and encoding operations. To ensure a viable range of feasible sequences, we applied more than one style in each set. For example, the sequences in Figure 4.5 cover encoding and transform transitions, and include parallelism.

In each set, one (and only one) sequence was the highest ranked sequence by GraphScape. Another sequence was generated by random permutation. The rest were manually designed to provide meaningful, non-random sequences distinct from the others. These hand-crafted stimuli included sequences respecting parallel structure and ordered filter terms.

Procedure. For each set, participants first viewed all 6 charts and answered three multiple choice questions. We used these questions to screen responses where participants did not adequately read the charts. We then prompted participants to think about how they would sequence the charts to best communicate the data. Once ready, subjects clicked a confirmation button and were presented with the 5 stimuli sequences, randomly placed in 5 rows (Figure 4.5). Upon mouse hover, the interface showed magnified views of a chart and its immediate sequence neighbors. We asked subjects to rate each sequence according to how well it presents the data in a clear and logical manner. Subjects responded using a 5-point Likert scale ranging from "Very Poor" to "Very Good". After a trial, subjects were asked to describe what features motivated their ratings. Finally, subjects completed a short demographic survey.

4.6.2 Results

We analyzed a total of 1,535 ratings, omitting 115 responses where participants incorrectly answered one or more screening questions. Figure 4.6 shows 95% confidence intervals of mean ratings for each sequence. To compare sequences across all task sets, we first analyzed the data using a linear mixed-effects model. We then analyzed task-level responses using non-parametric tests. Finally, we analyzed the rank correlation among subject ratings and variants of the GraphScape cost model. All results — including experimental data, analy-

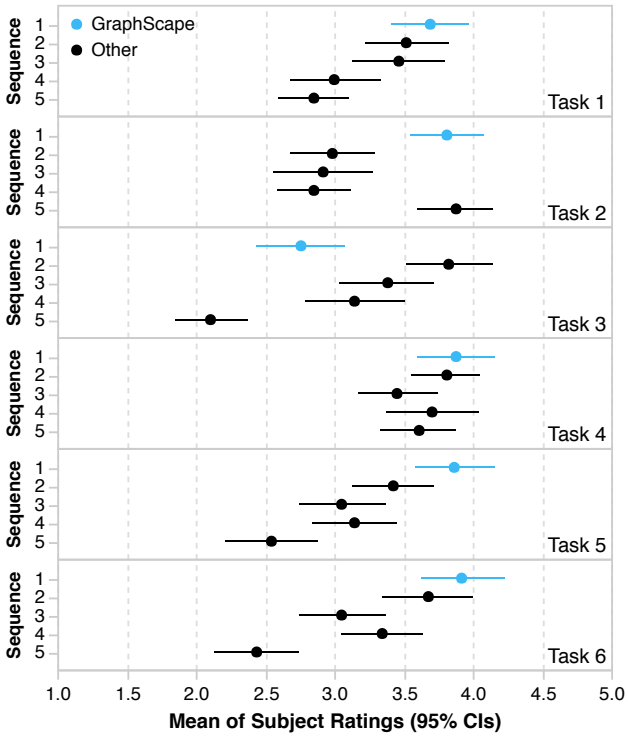


Figure 4.6: Mean ratings and 95% confidence intervals per sequence, computed via bootstrap. Blue indicates the top GraphScape sequence.

sis scripts, and post-hoc power analysis used to confirm sufficient power — are included as supplemental material.

GraphScape recommendations receive higher ratings. We first fit a linear mixed-effects model to assess if GraphScape sequences led to higher ratings across tasks. The model terms were a *graphscape* factor indicating if a sequence was the highest ranked by GraphScape within its set, and the integer *position* of the row containing the sequence in the UI to control for presentation order. We included random effects for both *subject* and *task*, using a maximal random effects structure [6] with random intercepts, slopes for *position*, and covariances between the two. The *graphscape* coefficient was 0.46 (95% CI: [0.32, 0.61]), indicating a half-point boost for GraphScape's top-rated sequences. This result is statistically

significant according to a likelihood ratio test ($\chi^2(1) = 39.87$, $p < 0.001$). The *position* term did not exhibit a significant effect on subject ratings ($\chi^2(1) = 0.8801$, $p = 0.348$).

For all but one task, the top GraphScape sequence is among the highest rated. To analyze data at the per-task level, we performed non-parametric Friedman tests for each set. All tests were significant except for set 4. For all other sets we then performed post-hoc pairwise comparisons of sequence ratings using paired Wilcoxon rank-sum tests, with Holm correction to account for multiple comparisons. GraphScape sequences received the highest average rating for sets 1, 4, 5 & 6, and the second highest for set 2. Both statistical tests and 95% confidence intervals (Figure 4.6) indicate statistical "ties" among the top-scoring sequences, for which no significant differences were found. For all tasks except set 3, GraphScape lies in the top-scoring group.

In task 3, GraphScape's top choice (sequence 1) received neutral ratings. It fared significantly worse than sequences 2 & 3 ($p < 0.01$), but better than sequence 5 ($p < 0.01$). To understand why GraphScape did not perform as well, we turned to subjects' rationales. Set 3 consists of charts about automobiles, including the year of manufacture and the region of origin. Most subjects (37/55) referenced chronological ordering as a primary motivation. The top-ranked sequences primarily organize the charts by year, then by category. GraphScape instead prioritizes filtering by origin and then secondarily filtering by year. Each provides a reasonable order, depending on whether one wishes to compare regions within years, or compare years within regions. We also note that among the charts in set 3, sequences 2 and 3 were the next highest rated by GraphScape, with cost estimates only slightly higher than sequence 1. An interface showing multiple GraphScape recommendations would thus list each of these options.

GraphScape and subject rankings correlate strongly. The analyses above compare only the sequence with the lowest GraphScape cost to all others. To compare subject's overall rankings to GraphScape rankings, we computed Spearman rank correlation coefficients among subject rankings, GraphScape rankings, and GraphScape rankings calculated using only transition costs or only global sequence weighting. We observe that GraphScape

and user rankings have a strong, significant correlation with each other ($\rho = 0.8$, $p < 0.001$). A cost model using only the global sequence weighting term results in a weaker correlation with user rankings ($\rho = 0.6$, $p < 0.001$). Using only local transition costs leads to weak, insignificant correlations with user rankings ($\rho = 0.35$, $n.s.$). In addition, the cost models using only transition cost and only global weighting are largely uncorrelated ($\rho = -0.10$, $n.s.$), indicating that each makes an independent contribution to the full GraphScape cost model. These results validate GraphScape's rankings and the importance of considering both local transition costs and subsequence consistency.

Overall, the GraphScape cost model largely matched subject preferences. GraphScape recommendations led to higher ratings on average and were among the highest-ranked sequences for all but one task set. We observed a discrepancy in terms of chronological order, where subjects' top-ranked sequence scores highly — but not highest — according to GraphScape.

4.7 GraphScape Applications

The GraphScape model is both a generative tool (e.g., given one or more starting points, one can traverse the graph to enumerate related charts) and an evaluative tool (e.g., to measure costs among multiple charts). To illustrate the utility of GraphScape, we present a trio of applications that *recommend visualization sequences*, *elaborate transition paths* between visualizations (e.g., for designing animated transitions), and *recommend design alternatives* given an initial design.

4.7.1 Sequence Recommendation

A motivating application of GraphScape is to automatically sequence a set of charts. Sequencing is valuable for improving interpretability in narrative visualization (e.g., when a user creates a presentation of charts bookmarked during exploratory analysis) and visualization recommendation (e.g., given a set of recommended charts, how might they best be ordered?). The GraphScape cost model provides an objective function for determining opti-

mized sequences. In the case of a small input set, we can simply enumerate all possibilities and score them. To efficiently recommend longer sequences, the GraphScape cost function can be used within an optimization method. For basic narrative use cases, including measure walks and dimension walks [33], we envision automated ordering integrated with manual review and fine-tuning by end users. Tools could present multiple high-scoring sequences (or even apply different GraphScape variants based on alternative rankings or weighting terms) to suggest additional options.

4.7.2 Path Elaboration

Another application is elaborating transition paths (lowest-cost edit sequences) between two visualizations. Path finding can be performed via traversal of GraphScape's directed graph model. As there may be multiple graph paths representing different orders of edits, GraphScape transition costs can be used to determine a shortest path using standard techniques such as Dijkstra's algorithm. One use case for path elaboration is the automatic design of animated transitions. A shortest-path traversal can be used to enumerate "key frames" for staged animations, where each frame is an intermediate chart along the shortest path. That said, prior work [30] suggests that aggressive staging is not always ideal. One area for future work is to investigate how GraphScape cost estimates might aid identification of path "cut points" at which to break up an animation into stages to mitigate the increasing interpretation costs of long, complex transitions.

4.7.3 Design Alternatives

Given an initial design, GraphScape can be used to search for alternative designs that meet some desired criteria. One compelling example is creating scalable visualizations. A basic scatter plot is effective with a limited number of records, but suffers from overplotting as data volume increases. We may consider a chart *scalable* if the number of rendered marks is below some threshold (say 5,000), and use this criteria to search for scalable visualizations that preserve design features of a (non-scalable) input chart. Figure 4.7 illustrates this

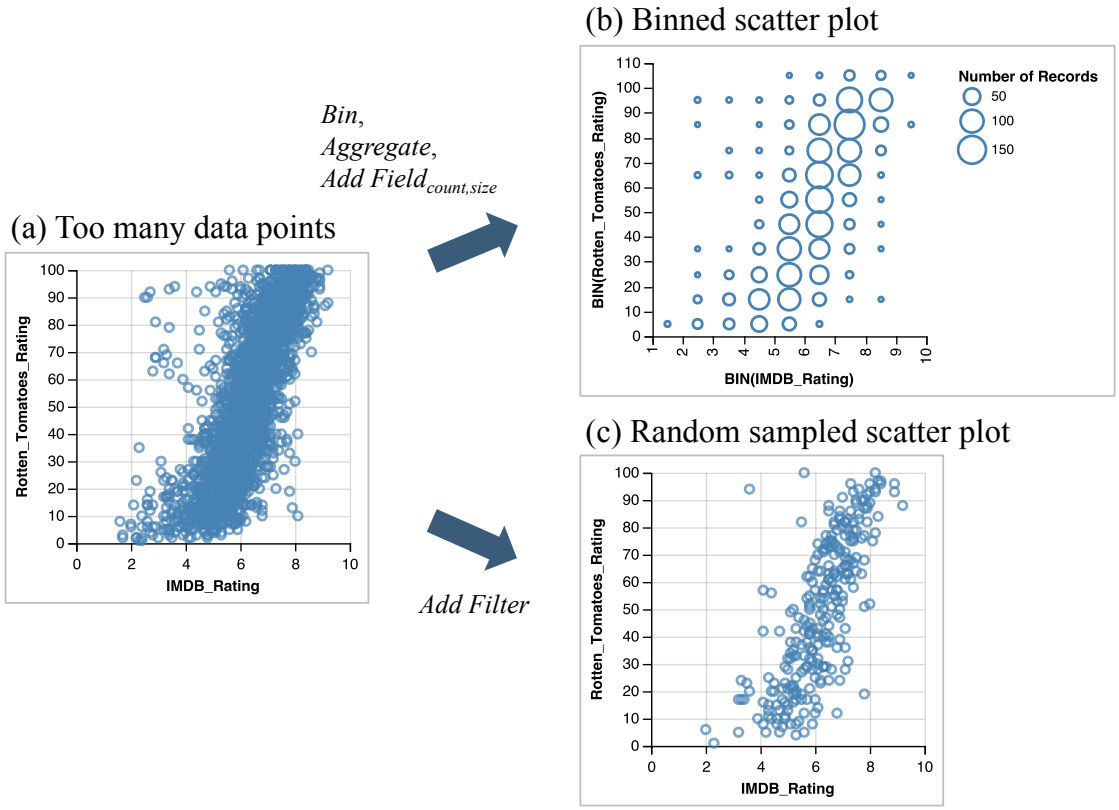


Figure 4.7: Using GraphScape to search for scalable design alternatives. Given (a) an initial chart with overplotting, we can recommend (b) a binned 2D histogram and (c) a view filtered to a random sample.

approach: given a basic scatter plot specification as input, the GraphScape model can be used to find scalable specifications ranked by edit distance from the input. Here, we perform a constrained graph traversal that preserves the field assignments to the x and y channels. The search results include addition of a filter transform to visualize a sub-sample and use of binned aggregation (adding two bin transforms and assigning *count* to the size channel) to form a 2D histogram. Here, scalability constraints and edit distance together determine the results. Future work could directly incorporate additional ranking terms (e.g., perceptual effectiveness scores or statistical measures of the underlying data).

4.8 Conclusion & Future Work

We presented GraphScape, a directed graph model of the visualization design space and a corresponding cost function for ranking visualization sequences. We contribute both a transition cost model determined by a ranking of edit operations, and additional cost terms that leverage sequence analysis to promote consistent subsequences and sorted filter values. We also presented the results of two controlled experiments that inform and validate our model, and demonstrated applications of GraphScape that enable new features for visualization tools. Our GraphScape implementation, including cost model generation code and an automatic sequencing tool, are available as open source software at `https://github.com/uwdata/graphscape`.

The version of GraphScape presented here can support higher-level reasoning about visualization designs. Still, a great deal of future work remains. First, our modeling assumptions and cost function terms should be further studied and refined. GraphScape covers a reasonable portion of the design space spanned by Vega-Lite, but could be extended to more nuanced specifications. For example, GraphScape's cost model does not differentiate between log and square root axis scale transformations, and does not yet handle filter predicates beyond basic equality comparisons. Further study might also yield improvements to our cost function, for example by learning optimized weights for each cost function component, trained on a corpus of user sequencing decisions.

In this work we adopted the simple and convenient model of flat, linear sequences. While we include terms to promote consistent subsequences, future research might directly model more complex organizations. For example, one might formulate cost models for tree or graph structures capable of representing non-linear narratives [65]. In addition, the current work focuses its analysis at the level of visualization specifications, but does not include analysis of the data itself. Should some otherwise-identical transitions (e.g., adding field A vs. field B) receive different cost estimates based on statistical relationships among the data fields?

Our experimental studies focused on sequence authoring tasks performed by knowledge-

able visualization students and sequence rating tasks performed by a more general audience recruited on Mechanical Turk. Future experiments might include task-based performance measures. For example, how do different sequence choices affect subsequent decision making or information recall? While more complicated to design, such studies could provide additional design guidance, and may identify cases where subject performance and preferences do not align. In addition, the "correct" sequence for a given situation may vary with context, such as the data set, intended audience, and presenter goals. Continued research is needed to cultivate a more nuanced understanding.

Chapter 5

GEMINI: A GRAMMAR AND RECOMMENDER SYSTEM FOR ANIMATED TRANSITIONS IN STATISTICAL GRAPHICS

One challenge to automatically generate animated transitions between statistical graphics is creating a representation that machines can read and write. This chapter introduces groundwork to formally represent animated transitions between statistical graphics and a basic animation recommendation model for a given transition. This work was done with a collaborator, Jeffrey Heer, and published at IEEE VIS 2020.

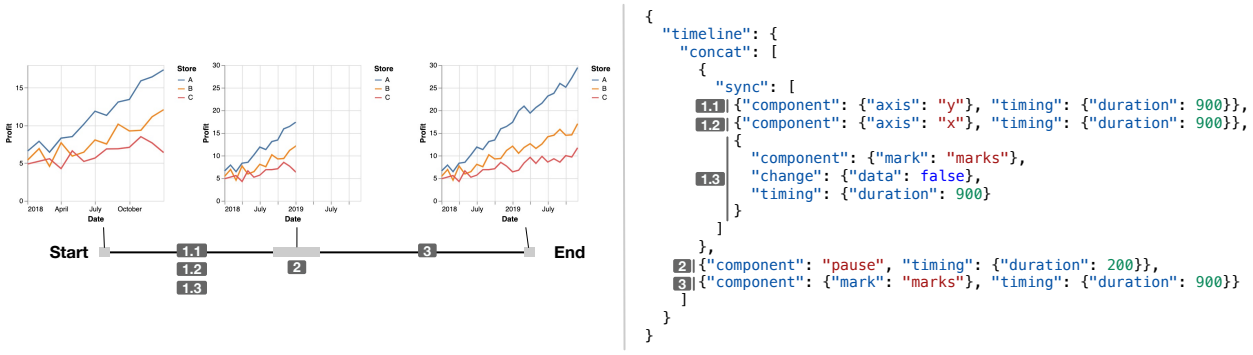


Figure 5.1: Left: An animated transition zooms a line chart to a larger time window. The timeline indicates the animation sequence; the enlarged gray interval indicates a pause. Right: The Gemini specification for the transition, which changes the scales of the axes and lines (1), pauses (2), and then extends the lines (3). The numbered items represent *steps*, the basic units of the Gemini grammar.

5.1 Introduction

When exploring data or communicating results, people often transition between related statistical graphics. To facilitate understanding of what has changed across a transition, visualization researchers have developed and studied animation techniques. Prior studies have examined the effectiveness of animation for conveying transitions [8, 28, 34, 38, 49, 57] and proposed guidelines and strategies for animation design, including the use of techniques such as staging and staggering [16, 30, 39, 55, 70].

Despite this guidance, creating effective animations remains challenging, as current tools either cannot express more nuanced designs [3, 58] or do so only with significant effort. A designer may need to select elements and properties to animate, specify transition parameters, and coordinate the relative timing of separate stages. Using D3 [11], for example, the implementation of animated transitions often requires manual orchestration of animation stages using a transition abstraction that intertwines visual encoding and animation specifications, impeding rapid design exploration and reuse.

To reduce this hurdle we contribute Gemini, a declarative grammar and recommender system for animated transitions between two single-view statistical graphics. In Gemini, animated transitions are formally represented by *transition steps* in terms of high-level visual components (marks, axes, legends) and *composition rules* synchronizing and concatenating steps into staged animations (Figure 5.1). This formal representation allows software to reason systematically about animated transitions. By taking advantage of this formalism, Gemini's recommender system produces candidate animated transitions between given start and end visualization states expressed in the Vega visualization grammar [63]. The recommendations can facilitate the design process by serving as starting points so that users need not manually create animations from scratch.

We begin by articulating our design goals for the Gemini grammar. We target a balance between expressiveness and ease-of-use by reviewing existing animation tools and alternate approaches. We also observe how people describe animated transitions in a preliminary

study to gauge a proper level of abstraction. We then introduce the primitives of Gemini along with our backing design rationale.

We go on to describe the workflow of the Gemini recommender system: change detection, enumeration, and ranking. We introduce a heuristic cost function that ranks enumerated candidate designs based on their complexity. To assess and refine the cost function, we conduct a user study and tune the cost function parameters to user preferences by promoting single-stage designs and demoting multi-stage ones.

We verify the utility of Gemini by replicating animated transitions created by designers using D3. We observe that among 11 user-crafted animated transitions, 9 can be expressed exactly in the Gemini grammar. In addition, 5/9 replicable designs can be achieved by changing only timing parameters of a top-3 Gemini suggestion. A total of 8 can be achieved by editing a top suggestion. We also find that 7/11 designs exhibit mistakes that all Gemini-produced designs avoid. These findings show Gemini's potential to suggest useful starting points for user-desired animations. We conclude by discussing ways to improve Gemini's user interface, achieve more nuanced suggestions, enhance expressiveness, and overcome implementation challenges in our current proof-of-concept system.

5.2 Related Work

Animated transitions are used to convey state changes and engage viewers. We focus on animated transitions between statistical graphics, with the goal of accurately conveying changes, directing attention, and helping viewers stay oriented.

5.2.1 Event Structure In Perceptual Psychology

Zacks & Tversky review how people conceptualize events in perception [79]. They maintain that events are structured in two hierarchical ways: *partonomy* (one event can be a part of another bigger event) and *taxonomy* (one event is an instance of one category). For example, in event partonomies, two event segments of "changing the color of visual marks" and "filtering out the marks" are part of "the change of the marks." In event taxonomies,

the former is a kind of "encoding change" and the latter a kind of "data transformation." On top of these hierarchical structures, people show better perception when communicating at *a basic level* of the event segments. To align with this psychological framing, we determined the basic level of events in Gemini ("steps") by conducting preliminary interviews. In addition, Gemini arranges these events in a hierarchical structure (timeline or block) to support event segmentation and programmatic enumeration.

5.3 Preliminary Interviews on Animation Design

To observe how people segment animated transitions and to gauge an appropriate level of abstraction for Gemini, we conducted an informal study of how people describe animated transition designs. We recruited five people (3 female, 2 male) near our university campus who had at least 2+ years of data visualization experience. We gave each participant start and end states of an animated transition and asked them to draft an animation design, identifying specific graphic elements, changes to those elements, and timing information. After receiving their first draft, we asked participants to describe possible alternatives. Finally, we showed them implemented animations for the stimuli and asked them to characterize those. All sessions were conducted in person.

To cover multiple transition types and techniques (*e.g.,* staging and staggering), we chose three stimuli from interactive articles and a video (available in supplemental material). These examples cover 5/7 of Heer and Robertson's transition types [30] by omitting *Ordering* and *View Transformation* due to the limited number of recruited subjects. We analyzed participants' transcripts for words indicating timing constraints and graphic components, then derived the following three insights about appropriate abstraction levels:

I1. Participants referred to groups of marks by their shape (*e.g.,* lines, texts, points), role (*e.g., "uncertainty band"*), and/or backing data (*"NY points"*). This observation implies that the grammar should be able to select elements by their geometry, roles in the visualization context, or data properties.

I2. Participants described changes to guides (axes or legends) both in general terms (*"expand the y-axis"*) and by referring to specific sub-elements (*"render the x-axis title"*).

I3. Participants included different staging and staggering elements. Staged animations were described using constraints: synchronizing (*"at the same time"*) and concatenating (*"then"*, *"after"*).

5.4 The Gemini Grammar: Motivation and Design

Gemini is a declarative grammar for specifying animated transitions between two (*start* and *end*) single view visualizations, defined using the Vega grammar [63]. By "single view," we refer to charts with at most one x-axis and one y-axis. The Gemini compiler processes a specified transition design (Figure 5.1) and the provided start and end states to produce a playable animation plan. When designing Gemini, we considered three ways to specify animations: (1) extend a single visualization specification, (2) define transformations that map a start state to an end state, or (3) specify transitions relative to explicit start and end states. We discuss each in turn.

Animation specifications can *extend* existing visualization specifications, as in gganimate [58]. As Figure 5.2 shows, authors can append animation directives to existing ggplot2 code to specify the transition in Figure 5.1. In this approach, the animation design context closely matches the visualization context and is similar to the animation model of CSS Transitions for Web design [20]. However, it limits expressible transitions to variations of a single visualization specification, typically data changes under static visual encodings. Since we aim for a broader spectrum of transitions, we do not use this approach.

An alternative is to specify *transformations* that turn a starting visualization state into an end state. Rather than define explicit start and end states separately, *the end state is implicitly defined* by the transformations applied. For instance, in order to zoom, D3 [11] users can: create new x and y scales; select the existing axes, grids, and mark elements; and assign the new scales to change the elements to final states (Figure 5.3). This approach

```
p <- ggplot(data, aes(date_num, profit, group = store, colour=store)) +
  geom_line()

# Animation-related Lines
anim <- p +
  transition_filter(
    transition_length = 1,
    filter_length = 1,
    state == 0,
    state == 0 || state == 1
  ) +
  view_step(pause_length = 1, step_length = 1, nsteps = 2, include = FALSE)
animate(anim, duration=2)
```

Figure 5.2: gganimate code to produce Figure 5.1. Given a ggplot2 chart (`p`), gganimate's `transition_filter` and `view_step` are appended to animate the lines between two states and a scale change, respectively.

requires the definition of all specific manipulations necessary to produce both the animation and end state. It thereby interleaves animation and visualization design: one must redefine an independent target state, in terms of desired transitions.

A third approach is to provide explicit start and end states and then describe desired *transitions* between them. This strategy separates a design into three specifications: two for visualization states and one for the transition. As a result, the transition specification can be more succinct, as it needs only to orchestrate the changes to get to the end state *without implicitly specifying the end state*, as in D3's transformation approach. However, to tailor animations there must be a way to refer to chart elements across the start and end states, and so a compiler must identify corresponding elements.

Gemini follows this third approach: we aim to support an authoring scenario in which designers (or systems) have explicit start and end visualization states (or "keyframes"). This approach allows designers to focus on these states before introducing animation concerns, and accords with existing tools. For example, visual analysis tools (*e.g.,* Tableau, Voyager [76]) already define visualization states using separate, declarative specifications and so are more amenable to this approach. This approach can support a keyframe-authoring paradigm for

```
x.domain(d3.extent(data, d => parseTime(d.date)));
y.domain([0, d3.max(data, d => d.profit)]).nice();

const t = d3.transition().duration(900);
const xAxisChange = d3.select(".xAxis")
  .transition(t)
  .call(d3.axisBottom(x))
  .end();
const xAxisGridChange = d3.select(".grid.x")
  .transition(t)
  .call(d3.axisBottom(x).tickSize(-height).tickFormat(""))
  .end();
const yAxisChange = d3.select(".yAxis")
  .transition(t)
  .call(d3.axisLeft(y).ticks(5))
  .end();
const yAxisGridChange = d3.select(".grid.y")
  .transition(t)
  .call(d3.axisLeft(y).ticks(5).tickSize(-width).tickFormat(""))
  .end();
const lineScaleChange = svg.selectAll(".line")
  .transition(t)
  .attr("d", d => valueline(d.values));

await Promise.all([xAxisChange, xAxisGridChange, yAxisChange,
yAxisGridChange, lineScaleChange]);

const newGrouped = d3.nest().key(d => d.store).entries(data);
const t2 = d3.transition().duration(900).delay(200);
svg.selectAll(".line")
  .data(newGrouped)
  .transition(t2)
  .attr("d", d => valueline(d.values));
```

Figure 5.3: D3 implementation of Figure 5.1. D3 requires transformations that map the starting visualization state to the end state.

animating data graphics, which designers reportedly prefer [67].

We now describe the Gemini grammar in detail. A Gemini specification uses two primary abstractions: the *step*, a basic unit of animation, and the *timeline*, which composes units.

5.4.1 Step: Unit Transitions

A *step* is a unit transition that interpolates changes to a selected graphic component according to specified timing parameters. The numbered code blocks in Figure 5.1 are examples. Using steps, a complex transition can be split into digestible pieces. Formally, a step is a four-tuple:

$$step \coloneqq (component, change, timing, enumerator)$$

Step Component

The step *component* indicates the group of visual elements to be changed. There are *mark*, *axis*, or *legend* components, which can be selected by name. This high-level of abstraction partially satisfies observations from our study, where users select groups of guide elements and marks with the same geometry (**I1**). Gemini can also control more nuanced components, for example by staggering marks by data fields, or referencing sub-elements of guide elements (*e.g.,* ticks, labels, title). In addition, a *view* component refers to the overall frame (*e.g.,* chart sizing), and a *pause* (see 2 in Figure 5.1) induces delays.

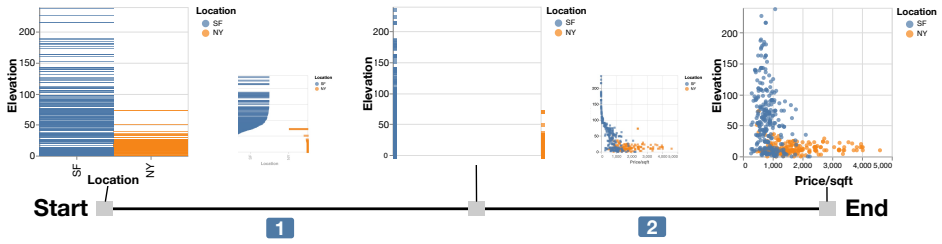
Step Change

An animated *change* to a component is defined as a five-tuple:

$$change \coloneqq (data, encode, scale, signal, mark\text{-}type)$$

The *data* entry indicates changes to the data backing a mark component, including inserts, removes, updates, and aggregations of the data. For example, 3 in Figure 5.1 performs a data change: it inserts new data from the end state, extending the domain of the line marks. Data changes are applicable only to mark components.

In Vega, datasets are defined by data sources and transformations. Assuming that the start and end states use the same data sources, Gemini identifies data changes between the two states by comparing their data transformations. If the dataset is aggregated (or




```
{
  "timeline": {
    "concat": [
      {
        "sync": [
          {
            "component": {"mark": "marks"},
            "change": {
              "encode": {
                "update": {
                  "xc": {"signal": "datum.in_sf ? 0 : width"},
                  "width": {"value": 5},
1.1               "height": {"value": 5}
                }
              },
              "marktype": false
            },
            "timing": {"duration": {"ratio": 0.5}, "staggering": "byElev"}
          },
1.2       {"component": {"axis": "x"}, "timing": {"duration": {"ratio": 0.5}}}
        ]
      },
      {
        "sync": [
          {
2.1         "component": {"mark": "marks"},
            "timing": {"duration": {"ratio": 0.5}, "staggering": "byElev"}
          },
2.2       {"component": {"axis": "x2"}, "timing": {"duration": {"ratio": 0.5}}}
        ]
      }
    ]
  },
  "staggerings": [
    {"name": "byElev", "by": "elevation", "order": "ascending", "overlap": 0.99}
  ],
  "totalDuration": 2000
}
```

Figure 5.4: A Gemini specification example for the first part of the animated transition in R2D3 Part 1 [78]. The dot plots (`"elevation"` vs `"in_sf"` (='Location')) are transformed to a scatter plot (`"elevation"` vs `"price_per_sqft"`). **1.1** specifies the encoding of the point marks to describe the intermediate state, and the mark steps (**1.1** and **2.1**) are staggered to move the points with lower `"elevation"` first.

disaggregated), Gemini binds the aggregated values to the raw data so that the raw data can exit (or enter) while being interpolated to (or from) the aggregated values. If no aggregation occurs, Gemini directly joins the start and end datasets. If the datasets are grouped by the same data fields, Gemini uses the grouping fields as join keys. Otherwise, it takes user-provided fields or data list indices as defaults. Akin to D3's data join [11], Gemini internally classifies the joined data into an enter set (newly introduced data), update set (persistent data), or exit set (stale data). Joins for these sets can be deferred in order to separate those changes into different animation stages.

The *encode* entry controls changes to a component's visual encodings. By default, Gemini interpolates between start and end visualizations. For entering and exiting data, the default transition is to fade marks in or out. Encoding changes can be separated into different steps (e.g., change x encoding at the first step and y encoding at the next step). Further, Gemini lets users directly specify temporary encodings to define intermediate states. For example, 1.1 in Figure 5.4 shrinks marks toward the sides before settling them into their final positions. For guide components, encode changes can separately target sub-elements to enable fine-grained control (**I2**). Particularly for the enter set, initial encodings can be set to match the given start state encoding.

The *scale* entry defines changes to the scales applied to the component. A subset of scales can change between the start and end visualizations. For guide components (*e.g.,* axes and legends), scale changes can produce new data for sub-elements, such as axis ticks/gridlines/labels, and legend symbols/labels. Especially for axis components, new data replace old data without joining (i.e., the old ones fade out, and the new ones fade in) when the dimension of the scale's domain changes (e.g., price(\$) to square feet(ft^2)) By default, Gemini automatically detects the change by comparing backing data fields of the corresponding scales in the visualizations specs. A change of dimension (or lack thereof) can also be explicitly indicated by users.

The *signal* entry refers to changes in Vega signals (dynamic variables) and can be controlled in the same way as *scales*. The *mark-type* entry signifies changes to the geometry of

the marks.

By default, Gemini assumes that all changes should be applied concurrently, resulting in a direct interpolation. More elaborate animations can be achieved by specifying more nuanced stages. For example, the 1.3 block in Figure 5.1 suppresses the data change, deferring that change until after the chart scales up (3).

Step Timing

Every step requires *timing* to schedule changes in a component. This element consists of four properties:

$$timing \coloneqq (duration, delay, ease, staggering)$$

The *duration* entry specifies the length of the step, either as an absolute value in milliseconds or as a fraction of the total duration. The *delay* entry is specified similarly and imposes a delay before enacting the change. The *duration* should be provided. If either *duration* or *delay* if expressed as a fraction, the total duration should be specified at the root level. The *ease* property selects a temporal distortion between the progress of the change and the elapsed time [22]. Matching D3 [11], the default *ease* value is cubic slow-in slow-out pacing.

Staggering is applied to step timing by referring to a named staggering specification defined at the root level (1.1, 2.1 in Figure 5.4). We place staggering definitions at the root level so they can be shared by multiple components. Gemini extends Chevalier et al.'s [16] definition of staggering to include five properties:

$$staggering \coloneqq (data\ field, order, overlap, ease, staggering)$$

The *data field* and *order* entries determine grouping and staggering order per element. The *overlap* parameter controls the temporal overlap between the consecutive elements using the ratio:

$$overlap = \frac{end(elem_i) - start(elem_{i+1})}{end(elem_i) - start(elem_i)}$$

where *start* and *end* are the element's start and end times. The ratio indicates how much an element's change overlaps with its preceding element. The overlap should be less than

or equal to 1.0. Figure 5.5 shows how overlap changes are staggered. The *ease* parameter determines how the duration is distributed to each element. As shown in the top-right of Figure 5.5, fast-out easing can be used to assign a smaller amount of time to later elements, emphasizing changes for the first few items and then overlapping subsequent movement. Gemini also supports nested *staggering* for elements in a subgroup (Figure 5.5, the bottom-right).

Step Enumerator

$$enumerator := (filter, step\text{-}size | [value_1, ...])$$

An *enumerator* defines a series of data changes (*e.g.,* values for specific years) so that it can express consecutive data changes (*e.g.,* showing one year at a time). When an enumerator is added to a step, Gemini produces a corresponding Vega filter transform and calculates data sets by replacing the filter expression's right-hand side value with enumerated values. These filtering values are derived from the start and end states with a given *step-size*, or provided as an array of values. The calculated data sets are consecutively joined, and these iterations are distributed within a single step (left of Figure 5.6). This approach is Vega-specific, as Vega filter transforms are used to obtain data set selections at a specific moment. Figure 5.7 shows an example that uses an enumerator (1 and 2) to depict trajectories of three points over time.

5.4.2 Timeline: Orchestrating Steps

Animations can be staged by composing steps into a *timeline*. A timeline entry (or *block*) consists of either a step or one of the composition operations *sync* and *concat*. The sync operation synchronizes an array of blocks at the start or the end based on the *at* property (default is start). The concat operation plays an array of blocks in a sequence. For example, the timeline of Figure 5.1 executes 1.1, 1.2, 1.3 at the same time, but it runs 2 and

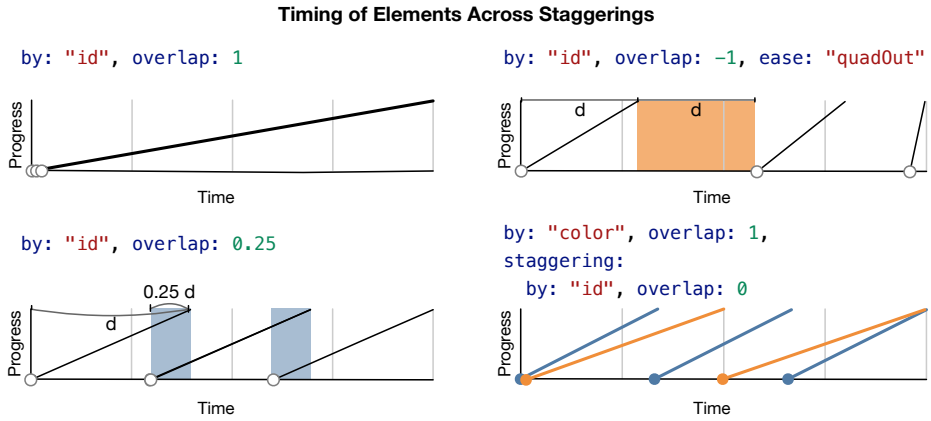


Figure 5.5: Timing of elements in staggered animations. Points represent individual elements, and lines indicate progress over time. The `overlap` parameter controls the intervals between consecutive elements, while `ease` distributes the duration of the step across the elements.

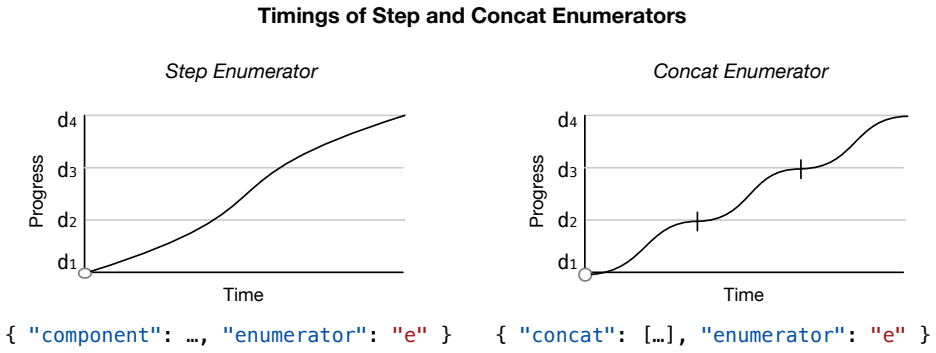


Figure 5.6: Example timing of an enumerator applied to a step (left) or a concat block (right). The enumerator consecutively joins data sets ($d_1 \rightarrow d_2 ...$). Step enumerators iterate the changes within a single step's timing. Concat enumerators iterate changes as multiple steps.

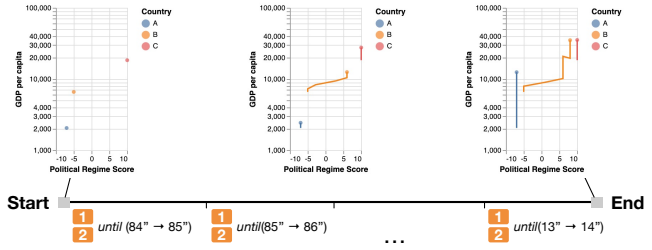
3 step-by-step. Formal representations for block and composition operations are:

$$block \coloneqq step \,|\, sync \,|\, concat$$

$$sync \coloneqq ([block_1, ...],\ at)$$

$$concat \coloneqq ([block_1, ...],\ enumerator,\ autoScaleOrder)$$

These operations align with our study observations that interviewees tended to describe animation stages using temporal constraints (**I3**).




```
{
  "timeline": {
    "sync": [
      {
        "component": {"mark": "circle"},
        "change": {"data": ["Country"]},
        "enumerator": "until”,
        "timing": {"duration": {"ratio": 1}, "staggering": "stg"}
      },
      {
        "component": {"mark": "line"},
        "enumerator": "until",
        "timing": {"duration": {"ratio": 1}, "staggering": "stg"}
      }
    ]
  },
  "totalDuration": 6000,
  "enumerators": [
    {"name": "until", "filter": "untilYear", "stepSize": 1}],
  "staggerings": [
    {"name": "stg", "overlap": 0.8, "by": "Country", "order": "ascending"}
  ]
}
```

Figure 5.7: Gemini specification for moving points along temporal trajectories (*c.f.*, [25]). The same enumerator is applied to iterate 1 and 2. These steps are staggered so that each point starts and ends individually.


```
{
  "timeline": {
    "concat": [
      {
        "component": {"mark": "marks"},
        "change": {"scale": false},
        "timing": {"duration": 1000}
      },
      {
        "sync": [
          {"component": {"axis": "y"}, "timing": {"duration": 1000}},
          {
            "component": {"mark": "marks"},
            "change": {"data": false},
            "timing": {"duration": 1000}
          }
        ]
      }
    ],
    "autoScaleOrder": ["marks"]
  }
}
```

Figure 5.8: Gemini specification for updating the values of bars. The concat block with *autoScaleOrder* automatically swaps the order of its two stages, ({Md} and {As, Ms}), to prevent overflow on the vertical scale.

In addition to explicit composition, Gemini provides timeline flexibility via *enumerators* and *autoScaleOrder*. Enumerators for a concat block have the same syntax as step enumerators, although they iterate block-by-block with a divided duration equal to the original duration. Figure 5.6 illustrates the difference between these enumeration styles.

The *autoScaleOrder* parameter ensures that the named components' data does not exceed its scale domains by sorting the children blocks of the concat. Figure 5.8 (A) and (B) show an example that updates quantitative values of a bar chart (Md), adjusting the scale (As, Ms) according to the new values. If the values decrease ((A)), the animation updates the

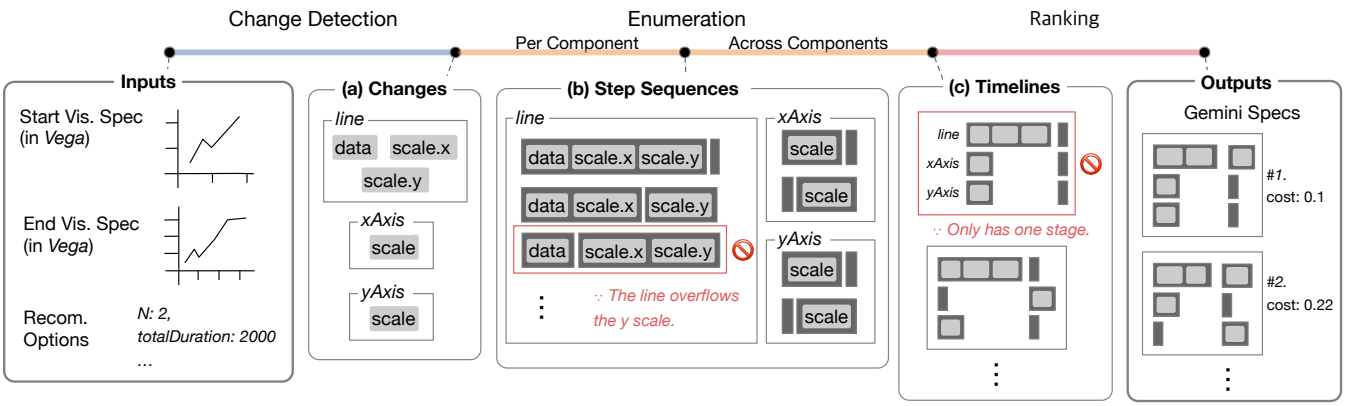


Figure 5.9: The Gemini recommendation workflow. The system takes as input the start and end visualization states of a transition along with user-provided options, such as the number of animation stages and whether axes maintain their domain dimensions. The system processes the inputs in three steps: (a) it *detects* changes in each component, (b) it *enumerates* candidate timelines by combining detected changes while pruning according to expressiveness constraints, and (c) it evaluates and *ranks* the timelines using its heuristic cost function.

values before changing the scale (Md → As, Ms); otherwise, the old data may overflow the new, smaller scale. If the values increase (B), the scale changes before the data updates (As, Ms → Md); otherwise, the new data can overflow the old scale. The autoScaleOrder property lets a single Gemini spec handle these data-dependent cases.

Users can specify *autoScaleOrder* using the name of a mark component. The Gemini compiler then generates all permutations of concat children blocks that use the specified marks (*e.g.*, Md and the sync of As, Ms), and picks one without scale overflow. If the compiler cannot find one, it returns a warning message and uses the original order.

5.5 *Recommending Animated Transitions*

Leveraging its grammar, Gemini can recommend animation designs by systematically enumerating and ranking transitions. This process can be parameterized using design op-

tions, such as the number of transition stages (N) and the total duration. Gemini generates suggestions in three steps (Figure 5.9): change detection, enumeration, and ranking. Our current implementation supports transitions between Vega specifications compiled from single-view Vega-Lite charts. The live version of this implementation is available at `https://uwdata.github.io/gemini-editor/`.

5.5.1 Change Detection

Gemini's recommendation system first analyzes differences between input visualizations. It pairs the marks, scales, axes, and legends by their names. Scale domain values, such as numeric ranges and sets of nominal values, are compared to determine whether changes occurred. For changes that Gemini finds hard to detect, users can provide direct input; for example, if the join keys of the data changes or if the domain dimension of a scale changes.

Detected changes are grouped per component, as in Figure 5.9 (a). Each change is one of the change types supported by the Gemini grammar (data, scale, encode, mark-type, signal) and contains meta information, such as initial and final states of the corresponding component, and whether the component expands or shrinks the width and height of the chart. Gemini separates the encode changes of mark components into high-level channels (x, y, color, shape, size, opacity, text) so that generated animations can stage these encode changes separately.

5.5.2 Timeline Enumeration

Gemini enumerates candidate timelines by combining detected changes. The enumeration is processed in two steps. First, for each component, Gemini enumerates sequences consisting of N sets of changes, where N is the target number of animation stages. Gemini prunes sequences inducing illegal intermediate encodings or data overflow, as shown in Table 5.1. For example, consider a transition that expands a line chart so that data and scales of the marks (lines) change as shown in Figure 5.9 (or Figure 5.1). The mark component (*line*) has three changes, `scale.x`, `scale.y`, and `data`. These three changes can be combined into a

Table 5.1: Constraints for pruning enumerated step sequences. Sequences violating any of the constraints are excluded.

Constraint	**Description**
Unavailable Scale	Any scale that is not available, e.g., changing a point mark's encoding to have color using a color scale but without the scale change introducing the color scale.
Unavailable Data Field	Any data field that is not available, e.g., changing data to aggregate without changing the encoding so the prior encoding refers to an old data field not in the newly aggregated data.
Unavailable Encoding	Any visual attribute that is not supported by its mark-type, e.g., changing the mark-type from point to rectangle without changing the encoding, such that the rectangle mark does not have encodings for width or height.
Overflow	Data overflows any domain of the scales used in the encoding, e.g., changing a bar chart's data to include 3 more categories without changing the corresponding categorical scale so newly introduced categories cannot be represented.

two-stage animation in 8 ($= 2^3$) ways. Among the possible sequences, Gemini filters out the three that cause overflow ([{`data`}, {`scale.x`, `scale.y`}], [{`data`, `scale.x`}, {`scale.y`}], [{`data`, `scale.y`}, {`scale.x`}]) by expanding the data before expanding both scales. The sets in each resulting sequence correspond to steps in the Gemini grammar.

Gemini, in turn, iteratively picks one of the resulting sequences per component and combines selections, while excluding combinations having an empty stage. For instance, if Gemini enumerates two components (A, B) involving one change for 2-stage designs ($change(A) =$ {`a`}, $change(B) =$ {`b`}, $N = 2$), it first enumerates step sequences ($seq(A) =$ { [{`a`}, $\varnothing$], [$\varnothing$, {`a`}] }, $seq(B) =$ { [{`b`}, $\varnothing$], [$\varnothing$, {`b`}] }). Then Gemini enumerates complete sequences ([{`a`}, {`b`}], [{`b`}, {`a`}]) while excluding those with an empty stage ([{`a,b`}, $\varnothing$], [$\varnothing$, {`a,b`}]). Gemini then reviews the combined sequences and inserts view changes that expand and shrink the width or height of the chart. An expanding view step is inserted in the first stage that expands the chart, and a shrinking view step is inserted on the last stage that shrinks the chart. Gemini evenly distributes the user-specified total duration across the stages and leaves further customization to users.

Table 5.2: Bundling effect conditions (B). The first three conditions maintain valid data graphics during the transition [30]; the second one applies the Gestalt common fate effect. The last two conditions encourage bundling of spatial or non-spatial changes.

Condition	**Effect**
Apply scale changes with different domain dimensions while not changing the corresponding encodings, e.g., new "temperature" scale applied on the old encoding to "precipitation" data field.	Penalty
Change the x(/y) scale of the mark and x(/y)-axis together.	Discount
Change the non-spatial scale (color, size, shape, opacity) of the mark and the corresponding legend together.	Discount
Change x and y scales of marks together but do not change their domain dimensions.	Discount
Change non-spatial scales of marks together.	Discount

5.5.3 Transition Ranking

Gemini evaluates all enumerated timelines using a scoring function that attempts to quantify the complexity of a Gemini animation specification. We define "complexity" as a proxy measure of how much the interpretation cost exceeds an assumed capacity: "cost" models the effort viewers must expend to identify animation changes, while "capacity" refers to how much cost viewers can tolerate. Gemini uses a heuristic function to rank enumerated timelines in ascending order of complexity (lower is better). We make the following assumptions:

1. *Cost and capacity*: Each change type has a positive cost, and people have a positive capacity per stage.

2. *Capacity as a function of duration*: Capacity monotonically increases with the duration of a stage.

3. *Bundling effects*: Some animated changes may become easier or harder to perceive if they are synchronized.

The first assumption aligns with prior work that finds that too many changes at once are hard to follow [30, 80]. We model capacity as a monotonically increasing function of

animation duration by hypothesizing that people can identify more changes as they can track slower objects with more time to process them. For example of an animation for updating data points, a 2-second-long animation might be easier to be tracked than a half-second-long animation since the former moves 4 tiems slower than the later. Finally, we assume bundling effects, which penalize or discount the cost of specific bundles of synchronized changes. For example, if the scale of the y-axis and the mark components y-scale change together, viewers may perceive this as one bundled change ("vertical change") by Gestalt common fate. This approach may be more effective than presenting them separately without synchronization. Specifically, the complexity function is:

$$Complexity = \sum_{s \in Stages} max(0, W(s) - C(duration(s)) + B(s))$$

$$\text{where } W(s) = \sum_{x \in change(s)} w(x)$$

W, C, and B represent the total weighted step cost, capacity, and bundling effect of a stage, respectively. We assign step costs W to follow the edit operation costs of GraphScape [41]. For example, $w(\textit{mark-type}) < w(\textit{data})$ means that edit operations in the Mark category are cheaper than Add/Remove/Modify Filter and Aggregate operations in the Transform category. We determined the bundling effects in Table 5.2 in reference to existing design principles [30]. The precise values of the costs and the bundling effects are available as supplemental material. Lastly, we model capacity as a sigmoid function whose output converges to a ceiling as its input increases. We initially fit the parameter of the sigmoid to reflect the scale of the costs and our empirical observations: $C(t) = \frac{0.8}{1+exp(-(t-800)/300)} + 0.2$, where t is in ms. These parameters can be adjusted to align with other design guidelines or preferences.

Formative Crowdsourced Study

To test and refine our recommendation logic, we conducted an online experiment. We recruited 53 people (26 females, 26 males, 1 declined to state) from Amazon Mechanical Turk and asked them to rank animated transitions recommended by Gemini in terms of how well

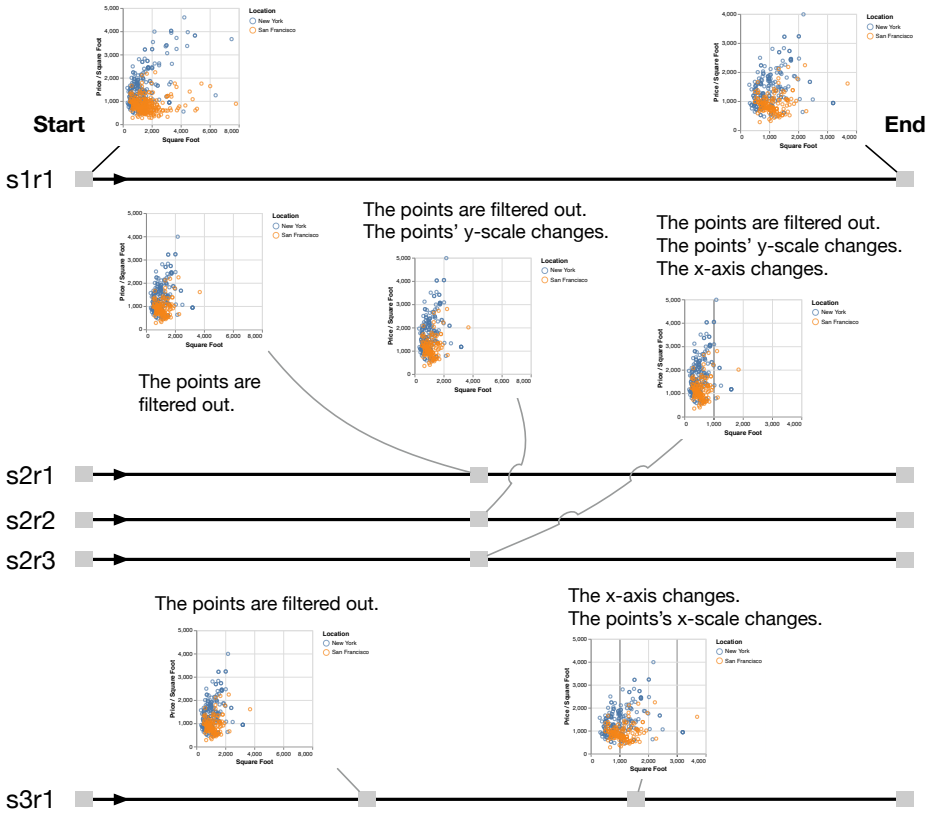


Figure 5.10: Gemini-generated animation designs for the Filtering Points stimulus of the formative study. The scatter plots represent house price data in two locations, and the transition filters out houses with 3+ beds. The other stimuli are available in the supplemental material.

they represent the transition in a clear and logical way. Then, we compare the collected users' rank to Gemini's cost function.

We gave 5 Gemini-recommended animation designs to each subject for each of 4 transitions covering 6 of the 7 transition categories of Heer & Robertson's transition taxonomy [30]. Of the 7, *View Transformation*, which moves camera positions, is omitted since it is unusual in statistical graphics, and scale changes can depict similar effects (e.g., zooming and panning). We picked the following 5 designs per transition: the single-stage design (s1r1), the best 2-stage design (s2r1), the median-ranked 2-stage design (s2r2), the worst-ranked 2-stage design (s2r3), and the best 3-stage design (s3r1). All animations were 2 seconds long, and

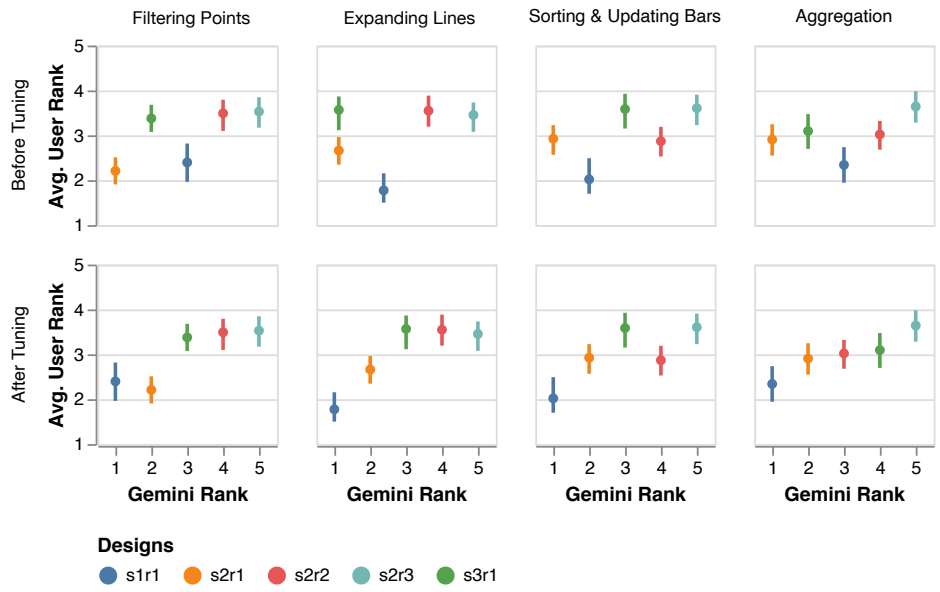


Figure 5.11: The Gemini rank against the average ranks by the subjects across the stimuli. The line shows the bootstrapped 95% confidence interval of the ranks. We tuned Gemini costs to better align with the observed rankings by promoting the single-stage design (s1r1) and demoting the 3-stage design (s3r1). We note that s2r1 and s3r1 for Expanding Lines were tied for the top Gemini rank prior to tuning.

this duration was evenly distributed to the stages. As an example, one stimulus is shown in Figure 5.10. The participants ranked the 5 designs of each stimulus at a time. The order of the stimuli and the order of the designs within a stimulus were randomized. Participants were required to play each animation at least twice before ranking them. We also asked subjects to describe what features of the best and worst designs informed their ranking. Participants were compensated $2.50 USD.

Our analysis excludes responses of participants who submitted irrelevant rationale text (e.g., "Pixar is best known for CGI-animated..."). To determine significant differences among the designs in each stimulus, we use the Friedman rank-sum test with post hoc pairwise comparisons. We report a user-preference A over B as $A > B$ when the pair has a significant difference ($p < 0.05$).

The charts on the top row of Figure 5.11 show the experimental Gemini-evaluated ranks against the average of the subjects' ranks per stimulus. Overall, Gemini underestimates user preferences for single-stage transitions (s1r1 > *), while over-estimating preferences for 3-stage transitions (s3r1 < s1r1, s2r1). Participants most favored s1r1 designs since they are smooth and slow. Some preferred s3r1 designs the least because they show too many animation effects.

The first stimulus (Filtering Points) filters out the data points and adjusts the x and y scales to the remaining data, exemplifying *Filter & Substrate Transformation* transition types [30]. Similarly, the second stimulus (Expanding Lines) expands lines along the time axis while expanding the domain of the two scales (*Timestamp & Substrate Transformation*). In both stimuli, the animations staging the horizontal and vertical scale changes were less preferred than the ones synchronizing them in one stage (s1r1, s2r1 > s2r2, s2r3, s3r1, except s2r1 > s2r3 in Expanding Lines). This observation aligns with Gemini bundling effects that discount the cost when the x and y scales change together (the fourth item in Table 5.2).

The third stimulus (Sorting & Updating Bars) sorts vertical bars and updates their values by substituting the data field (*Order & Schema Change*). Only the s1r1 animation is preferred over other animations (s1r1 > *). Among the other less-favored four, s3r1 and s2r3 are the least favored (though not significantly so) because they sort the bars and x-axis labels separately. This preference corresponds with a Gemini bundling effect that discounts the cost when the axis and mark change the same scale (the second item in Table 5.2).

The last stimulus (Aggregation) aggregates the averages of the data points per group and transforms them into bars (*Visualization Change*). For this transition, participant preferences diverged without significant differences except for s1r1 > s2r3. Notably for s2r3, subjects reported that the rising bars without the corresponding y-axis change at the last stage seemed manipulated.

Based on the surveyed ranking, we manually adjusted the parameters of the capacity function by decreasing the intercept, increasing the ceiling value, and shifting along the x-axis: $C(t) = \frac{1.4}{1+exp(-(t-1200)/300)}$. By doing so, the curve starts from the lower value and rises

up to the higher ceiling value. It increases the capacity of the stages with $t > \sim 1,373$ while reducing it for the other shorter stages. As a result, Gemini recommendations promote the single-stage animation design (2000ms per stage) and demote the 3-stage designs (667ms per stage). The tuned ranks more strongly correlate (though not perfectly) with the observed user ranks, as shown in the bottom row of Figure 5.11. Further systematic and data-driven tuning is a promising future work area relative to our manual adjustment.

5.6 Evaluation: Replicating User-created D3 Animated Transitions Using Gemini

We conducted a user study in which we collected manually authored animated transitions by designers using D3, and attempted to replicate them using Gemini recommendations. We chose D3 as it is a popular and expressive tool for creating animated visualizations.

Although evaluation studies typically compare user experiences with a proposed tool and other baseline tools, we instead take an experimenter-replication approach for multiple reasons: 1) the baseline tool (D3) has unequal amount of resources (e.g., API documentation, tutorial, examples) for users, and 2) our primary purpose at this juncture is to assess whether Gemini can express and recommend what users want to design, rather than whether the Gemini grammar improves the user experience. Accordingly, we do not make any direct claims regarding user experience. We can, however, verify expressiveness by assessing whether the replications express authors' original designs, and the recommendation quality is implied by the amount of required edits to the recommendations to replicate participants' designs.

5.6.1 Study Design

We recruited 8 (4 female, 4 male) D3 users with data visualization design experience. Seven participants were graduate students studying Human-Computer Interaction (P8) or data visualization. One (P3) worked at an IT company in a data visualization-related role. Participants used D3 for 2+ years ($\mu = 4.5$), though they reported that they do not currently use

D3 daily. P3 participated remotely, while the others spent at least 45 minutes in the presence of the first author. After the session, subjects were allowed to complete the task remotely. All subjects waived compensation for the study except for P8, whom we compensated with a $15 USD gift card.

We gave each participant D3 code for the start and end visualizations for one of the four transitions used in our formative (tuning) study. We asked them to design and implement 1+ animation of the given transitions in D3. They used their own laptops and were allowed to access any resources for the task (e.g., Internet search). After finishing the task, the subjects sent the first author their code with text reports about their prior D3 experience and completion time for each animation.

After collecting all submissions, we noted suspicious aspects that may not have been what the participants intended. These parts were identified by 1) differences between the final states of the subjects' animations and the given end visualization, and 2) static changes without any animation effect. We reached out to each participant to confirm if the suspicious parts were indeed mistakes. After confirmation, we replicated the submissions without mistakes by editing one of the top 3 Gemini recommendations. Finally, we confirmed with participants that our replications matched their original intent.

5.6.2 Results

Table 5.3 summarizes the results. Participants crafted a total of 11 animations for the given transitions. The self-reported average completion time for their first animation design was 77 minutes ($t \in [40, 150]$). The manually authored D3 animations, the top 3 Gemini recommendations, and the Gemini replications derived from the recommendations are available in the supplemental material and `https://uwdata.github.io/gemini-d3-study/`. Based on the results, we now assess the expressiveness of the Gemini grammar and the utility of Gemini suggestions.

Stimulus	Subject-Design	Completion Time (min)	Identified Mistakes	Replicable	Required Edits (except for duration & delay changes)
Filtering Points	P1-1	60	The marks instantly disappear.	Yes	*None.*
	P2-1	40	The symbols of the legend moves.	Yes	Add staggering to the first mark step.
Expanding Lines	P3-1	60	The lines shrink with some delay (200ms) comparing to the axes.	Yes	*None.*
	P4-1	45	None.	Yes	*None.*
	P4-2	+15 from P4-1	None.	Yes	Swap x- and y-axis steps, edit the first mark step to synchronize x-axis and data updates.
	P4-3	+30 from P4-2	None.	No	Not applicable.
Sorting & Updating Bars	P5-1	150	None.	Partially	Insert a stage to shrink the bars and the y-axis before updating the data.
	P6-1	130	Axis changes instantly at the end.	Yes	Add staggering to mark and x-axis.
Aggregating	P7-1	55	The y-axis title does not change.	Yes	Insert a 0 ms stage introducing the bars at aggregated value positions. Change the first mark step to keep prior x-scale.
	P8-1	79	The y axis does not change, the y scale of the marks does not change, and the width of bars get halved.	Yes	Significant edits are required, including appending extra marks to the start and end visualization specs.
	P8-2	+22 from P8-1		Yes	

Table 5.3: Replicating user-created D3 animations in Gemini. Subjects spent 40–150 minutes to animate the given transitions in D3. Of 11 animations, 3 exactly match the top-3 Gemini recommendations, and 5 can be derived from the recommendations by adjusting timing parameters only.

Expressiveness

Using Gemini, we successfully replicated 9 of 11 (82%) participant designs, as corroborated by the participants. Regarding the two replication failures, P5-1 was partially created, but P4-3 was not. P5-1 vertically squeezes and stretches the bars and the y-axis toward the

bottom line by SVG scale transformation while fading in and out. Since the Gemini grammar cannot apply this geometric scaling, we achieved the vertical scaling by changing the position encoding of all components except for the y-axis title, which just faded out and in without moving. P5 confirmed that our replication was similar to the original intent except for the title. P4-3 meticulously stages the horizontal and vertical expansions of each component so that the y-scale expands when the expanding lines reach the ceiling. This staged animation requires low-level individual timing of x- and y-scales of the line, which is not supported by the Gemini grammar due to its high-level abstraction.

Gemini Recommendations as Starting Points

We found that most authored animations can be derived from Gemini recommendations without significant modifications. Among the 10 expressible user-created animations, 5 can be replicated from one of the top 3 Gemini recommendations by adjusting timing only, such as the duration and delay of steps (P1-1, P3-1, P4-1) or staggering (P2-1, P6-1). Moreover, P4-2, P5-1, and P7-1 can be achieved by editing the recommendations by modifying the existing steps with minor edits (P4-2) or inserting a new stage with one or two steps to elaborate the recommended flow (P5-1, P7-1). Interestingly, P4-2 matches a design rejected by Gemini's recommender due to overflow. In other words, Gemini could recommend this design if we relaxed our expressiveness constraints (Table 5.1). On the other hand, P8-1 and P8-2 require significant effort to express in Gemini, requiring edits to the given visualization specifications to append extra marks and data transformations (P8-1, P8-2) and a parallel staging structure to overlap the animations of different components (P8-2).

User Performance in Crafting Animation with D3

Collecting the user-crafted animations and self-reports reveals the effort participants spent on their designs. First, all participants report spending 40+ minutes ($\mu = 77$) to arrive at an initial design, which includes the time for running and understanding the given D3 code, ideation, and implementation. Moreover, 7 of 11 designs exhibited mistakes. The mistakes

in P1-1, P2-1, P3-1, and P7-1 were made because the authors overlooked differences, and the ones in P6-1, P8-1, and P8-2 were due to time constraints (i.e., the authors did not want to spend additional time.). All mistakes are avoided by Gemini's recommendations, which are generated in under 1 minute on a laptop computer.

Implications

These findings imply that Gemini provides reasonable expressiveness and reliable recommendations with which designers can start authoring and exploring animation designs. For example, by employing Gemini, animated transition authoring tools could provide recommendations to facilitate the authoring process with initial designs.

5.7 Future Work and Conclusion

We presented Gemini, a declarative grammar and recommender system for authoring animated transitions between single-view statistical graphics. The high-level declarative format of the Gemini grammar provides a representation with which software can generate and explore animated transition designs. To evaluate Gemini, we replicated user-created animated transitions by modifying one of the top Gemini recommendations. Through these replications, we found that Gemini can provide reliable starting points for the authoring process. The Gemini compiler, recommender system, and examples are available as open-source software at `https://github.com/uwdata/gemini`.

There remain challenges for effectively using Gemini. Our evaluation results imply that Gemini suggestions can serve as reliable starting points for user-desired animation designs. However, an outstanding issue is the user interface for specifying visualization states and editing Gemini specifications. While existing APIs [42, 62] and graphical tools [61, 66, 76, 77] support authoring of visualization states, new interfaces are needed for animation specification. Future interfaces can lower the threshold for interacting with Gemini specifications (e.g., how graphical tools like Lyra [61] assist creating Vega visualizations) while enabling generation and review of recommendations.

In addition to user interface concerns, richer recommendations could facilitate design exploration. The current implementation of the recommendation system is a proof-of-concept that is limited to transitions between single view charts without layers. Additional types of transitions can be supported by increasing the expressiveness of the Gemini grammar. Recommendations could also be made finer-grained; for example, data changes can be divided into more specific units (e.g., aggregation, binning, filtering) by examining the data transformations of the start and end visualization states. Gemini could then apply more detailed guidelines for evaluation. Moreover, knowledge-based recommendation methods [42, 47] are more scalable in terms of handling evaluation factors (or changes) than our current heuristic methods. Open challenges include how to further formalize animation guidelines [30, 70] and our complexity measure within constraint programming systems.

Finally, to obtain more elaborate staged animations, it seems promising to recommend intermediate visualization states ("keyframes") and then cascade Gemini animations. For example, when points move from a scale domain of $[0, 1]$ to $[11, 12]$, it might be preferable to designate an intermediate state using a domain of $[0, 12]$ before moving the points, letting viewers see the 10-plus-length movements of the points. One way to recommend intermediate states is to leverage GraphScape [41] to explore and rank transition paths. After selecting a transition path with proper intermediate visualization states, Gemini could generate animated transitions for each hop (transition) on the path, then combine them into a final animation. For instance, the transition mentioned earlier ($S \rightarrow E$) can be divided into three Gemini animated transitions with a path including two intermediate states: ($S \rightarrow S_{[0,12]}$), ($S_{[0,12]} \rightarrow E_{[0,12]}$), ($E_{[0,12]} \rightarrow E$), where $S_{[0,12]}$ and $E_{[0,12]}$ are the start and end states on a $[0, 12]$ domain. With such extensions, Gemini could help facilitate a mixed-initiative keyframe authoring paradigm [67].

As noted, the expressiveness of Gemini is limited to transitions among single view charts. To express transitions for more varied charts (e.g., multi-view, pie, trail, and geographic charts), the grammar needs to be extended to support new components such as headers and mark groups, reference geographic projections, and support shape interpolation (e.g., pie

to bar). In addition, the Gemini grammar cannot assign separate timings across changes within a step (as we failed to replicate for P4-3). One way to support such nuanced timing is to allow the grammar to assign separate timings for low-level component properties (e.g., individual timings for each scale for a mark).

Regarding the Gemini implementation, remaining challenges involve the computational complexity stemming from data and step count sizes. The compiler and recommender system join data and enumeration sequences that combinatorially increase with the number of stages. For example, on the first author's personal laptop, compiling a Gemini spec for random movements of 5,000 points took 300-500 ms (versus 10-60 ms in D3), and the compiled animation played 300-600 ms (0-60 ms in D3) longer than the specified duration (5000 ms). In addition, the recommendation feature takes more than one minute to produce recommendations with $N = 4$ stages. Future compiler optimizations — such as indexing for data joins and dynamic programming to enumerate and evaluate the timelines — can improve the performance.

Chapter 6

GEMINI^2: GENERATING KEYFRAME-ORIENTED ANIMATED TRANSITIONS BETWEEN STATISTICAL GRAPHICS

In the previous chapter, I presented foundational work for recommending animated transitions for two given charts. Still, more is required to support authoring elaborate staged animations with complex changes. In this chapter, I present Gemini^2, which extends Gemini and GraphScape to express animated transitions via keyframes, and thereby provide an easier-to-use method and more nuanced recommendations. This work was done with a collaborator, Jeffrey Heer, and submitted to IEEE VIS 2021 as a short paper.

6.1 Introduction

Data analysis and communication often involve multiple statistical graphics and transitions among them [7]. To convey the changes that occur in transitions, visualization researchers study and employ animation techniques that assess the effectiveness of animated transitions [55, 70] and their various strategies, such as staging [30], staggering [16], time distortions [22], and bundling (or path optimizing) [23, 73]. Animation practitioners have created compelling media conveying data-driven insights, e.g., the famous Hans Rosling's presentation using Gapminder Trendalyzer [59], and an interactive article explaining machine learning algorithms [78].

However, authoring animated transitions between statistical graphics is not trivial; animation authors must implement low-level movements by writing code that considers many design variations, such as staging and timing. Recently, researchers facilitated this authoring experience using a non-programming interface [68] and domain-specific languages [27].

However, authors must still manually generate animation keyframes or specify changes by selecting low-level graphic components.

The previous chapter introduced a recommender system, Gemini [40], that recommends staged animations for a start and end chart. However, Gemini's expressiveness is limited since it can express only intermediate states as mixtures of the start and end chart. In addition, the recommended animation specs can be complex for users or systems to modify since they rely on visualization specs for intermediate states.

We introduce Gemini^2, a system for generating staged animated transitions between statistical graphics using keyframe sequences. Users first provide a chart array as a keyframe sequence. Then, Gemini^2 automatically generates animations connecting the consecutive keyframes. Users next create nuanced animations by adjusting the animation specifications that connect adjacent keyframe pairs. Gemini^2 thus supports a *keyframe-oriented animation authoring approach* [67], where users design animations by importing or demonstrating keyframes identified in visualization specifications and connecting them with animation specifications. We describe Gemini^2's enhanced expressiveness and ease-of-use vis-a-vis Gemini.

Further, we demonstrate Gemini^2's utility by presenting a usage scenario that shows how it helps users create elaborate staged animations using keyframes. Additionally, we evaluate keyframe recommendation through a crowd-sourced user study, where participants rank charts recommended by Gemini^2 and Gemini. We find that Gemini^{2}s recommendations are as compelling as Geminis, and that Gemini^2 can prioritize enumerated staged animations better than its predecessor. We conclude by discussing future directions, such as user interfaces, to improve the Gemini^2 animation authoring process.

6.2 Related Work

Animated transitions are commonly used to deliver changes between two states of data graphics. Visualization researchers have shown that animations facilitate a range of visual analysis tasks, from low-level ones (e.g., value comparisons, object tracking) to high-level ones, such as making decisions and inferring trends [8, 28, 34, 38, 49, 56, 57]. However, there

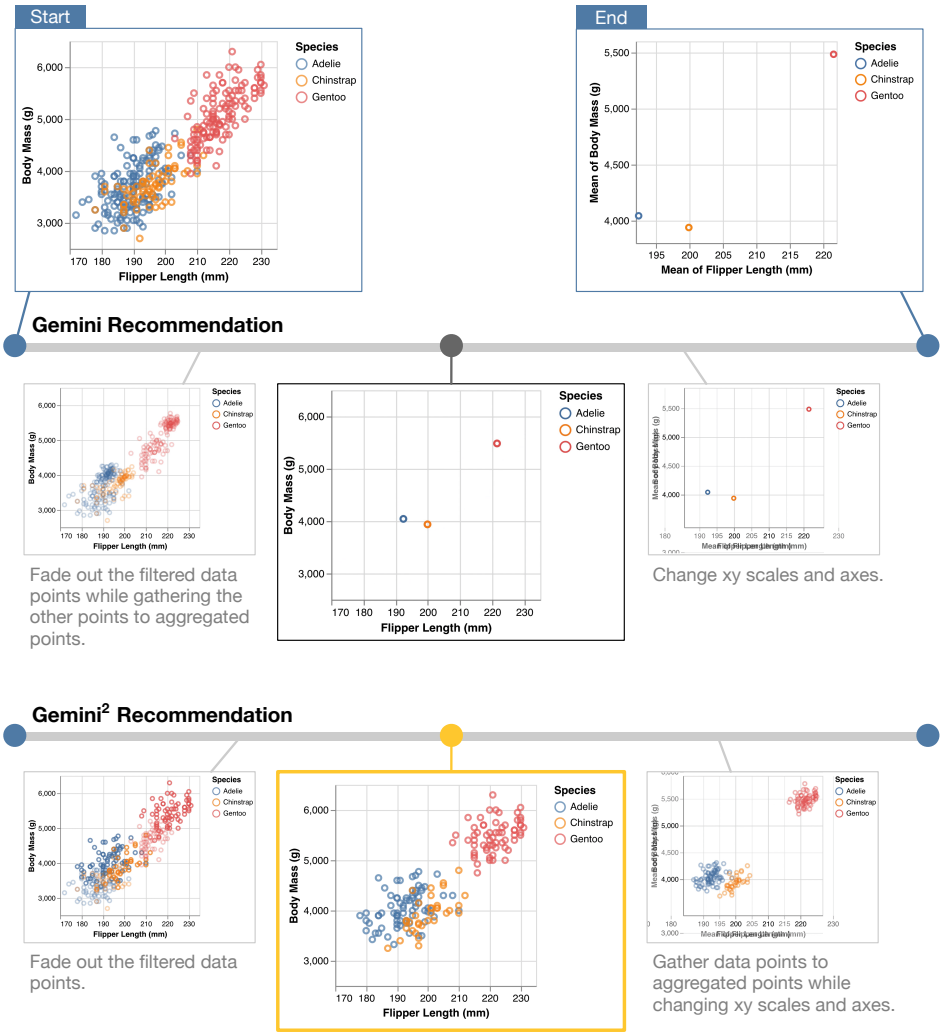


Figure 6.1: Staged animation timelines recommended by Gemini and Gemini^2 for a transition where data are filtered and aggregated. The gray lines represents timelines, and the circles indicate start, end, and intermediate states. Gemini2 generates a keyframe chart (yellow circle) to separate two data transformations, which Gemini cannot do.

has been skepticism about the effectiveness of animated transitions, as well. Tversky et al. [70] found that some studies showed bias toward animations, where animation conditions unfairly conveyed more information than the other conditions. Further, they suggested two high-level animation design principles, *Congruence*, "The structure and content of the external representation should correspond to the desired structure and content of the internal

representation.", and *Apprehension*, "the structure and content of the external representation should be readily and accurately perceived and comprehended". Robertson et al. [55] found that though animated time-series data in multiple small charts was less effective in analysis tasks, subjects preferred such data in a presentation context.

Prior research has proposed and studied various animation techniques to enhance communication capability, including timing variations [16, 22] and trajectory bundling [23, 73]. To convey complex changes, animations adopt staging techniques that divide the given changes into multiple sub-sequences and animate them consecutively. Staged animations outperformed unstaged ones at estimating and tracking value changes [30], conveying aggregate operations [39], and understanding online dynamic networks [19]. Gemini^2 aims to facilitate the authoring of staged animations by generating them automatically from given keyframes and recommending effective keyframes for given start and end visualization states.

Regarding paradigms of animating transitions between data visualizations, Thompson et al. [67] surveyed and characterized three authoring paradigms: keyframe animation (setting keyframes and tweening them consecutively), procedural animation (creating by behavioral parameters), and presets & template animation (applying pre-defined effects or templates). They found that most authors prefer keyframe animation, but animation authoring tools should support the other two paradigms depending on transition types and animation components. Gemini^2 facilitates the keyframe animation paradigm by letting users animate with customized keyframes and/or even by automatically generated keyframes for their transitions.

6.2.1 Comparison to Gemini

Gemini^2 enhances Gemini's expressiveness and ease-of-use. Gemini's critical limitation is that its grammar can specify intermediate states only as partially changed charts from the start to the end. In addition, the partial changes for intermediate states are specified in animation specifications. Gemini^2 extends Gemini to separate intermediate state representations into independent multiple visualization specifications, or keyframes. By doing so,

Gemini^2 increases expressiveness for staged animations and embodies the keyframe animation authoring paradigm.

Regarding recommendations, Gemini^2 can recommend staged animations by providing intermediate keyframes unlike Gemini, which recommends only animation specifications. To do that, we extend GraphScape [41] to itemize a given transition into semantic edit operations, recombine operations into multiple intermediate keyframes (charts), and prioritize each path (from the start to the end through intermediate charts) using heuristic rules. We enable these extensions by allowing GraphScape to synthesize a chart by applying edit operations. Gemini^2's keyframe recommendations can be more semantic level than Gemini's. For example, Gemini^2 can divide a transition with a filter and aggregate edit operations into two stages by separating them, but Gemini cannot since it treats such data transforms as a single data change of a mark component (Figure 6.1).

6.3 Gemini^2

Gemini^2 extends the Gemini grammar to support a keyframe-oriented animation authoring process. It represents an animation design as a sequence of N keyframes (Vega charts, $k_1, k_2, ..., k_N$) and $N-1$ Gemini specifications $(g_1, g_2, ..., g_{N-1})$ and is formally written as

$$Animation := \{(k_1, k_2, ..., k_N), (g_{1\to2}, g_{2\to3}, ..., g_{N\to N-1})\}$$

Keyframes are states that the animation passes through, and the Gemini specs are the specifications for each part of the animations connecting two adjacent keyframes. Therefore, users can express intermediate keyframes as arbitrary charts and connect them as an animation, which improves expressiveness relative to Gemini, as noted previously. Further, users can import multiple charts as keyframes and author stage-animated transitions.

Gemini^2 compiles its representation using the Gemini compiler; it creates $N-1$ animation objects for each consecutive chart pair (k_i, k_{i+1}) and corresponding Gemini spec $(g_{i\to i+1})$. The compiled $N-1$ animation objects are wrapped as a single object that plays the animation objects in a row.

6.3.1 Gemini² Recommendations

Gemini² provides two recommendation features by extending GraphScape and Gemini: (1) *keyframe recommendations* for a given transition (start and end Vega-Lite charts), and (2) *animation recommendations* for a given keyframe sequence. These two recommendation features let users automatically generate staged animations for a given transition or keyframe sequence.

Keyframe Recommendations

For given start and end Vega-Lite states, Gemini² can recommend intermediate keyframes to help authors explore staged animation designs. We implement these recommendations by leveraging GraphScape [41], a library for analyzing transitions between single-view Vega-Lite charts with Cartesian coordinates (e.g., bar, line, area, point charts). We extend GraphScape to enumerate chart sequences and rank them by how well they convey their changes in an identifiable way.

GraphScape recommends keyframe sequences in three steps, as shown in Figure 6.2. It first itemizes changes from the start to the end into edit operations, which are atomic units that can be combined to represent any transition. Then, GraphScape enumerates intermediate charts by recombining and applying the itemized edit operations. During enumeration, GraphScape modifies the intermediate charts' scale domains (i.e., ranges of data values for corresponding visual properties) as the union of the given two charts to prevent unnecessary scale changes. For example, imagine a transition filtering out some data and aggregating the remaining data into mean values. The start and end charts would have different scale domains ($domain_B \subset domain_A$). If an intermediate chart were applied only at the filter operation, the remaining data would have a different scale domain ($domain_C \notin \{domain_A, domain_B\}$), and the chart sequence from A to C to B would change scale twice. However, by modifying $domain_C$ to $domain_A$, the chart sequence would change scale only once.

After enumerating chart sequences, GraphScape prioritizes them by the order of re-

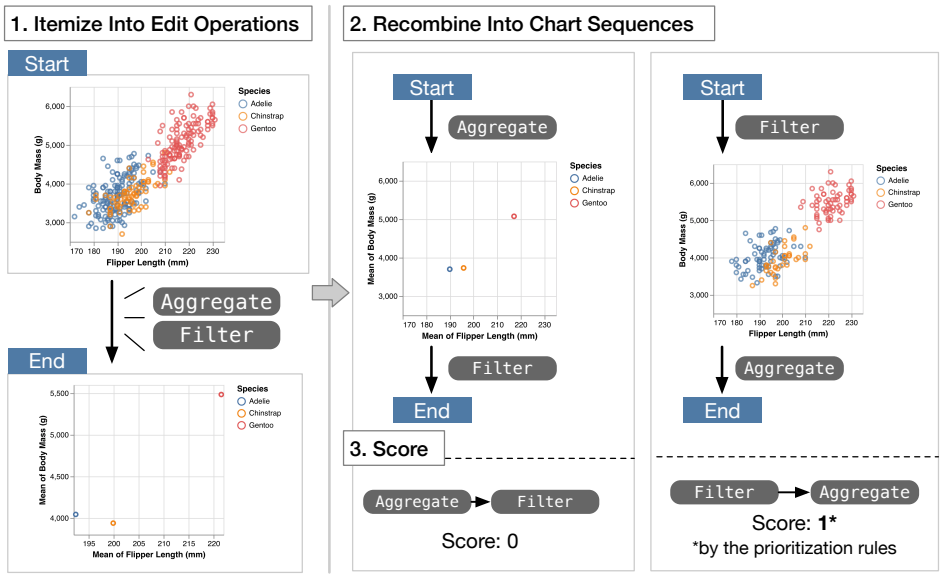


Figure 6.2: GraphScape's chart sequence recommendation process for a given transition. For given start and end charts, GraphScape enumerates chart sequences by itemizing their changes into edit operations and recombining the operations. Then, it scores the sequences to rank them according to predefined prioritizing rules (see Table 6.1).

combined edit operations. GraphScape applies pre-defined heuristic rules to evaluate the sequences. Each rule has (1) a *condition* determining if certain edit operations should follow other operations or be placed together, and (2) a *score* indicating if a sequence should be promoted or demoted if its operation order meets the condition. We define the rules in Table 6.1. Our goal is to make each edit operation identifiable by avoiding occlusions and implicit changes in aggregated data. We hand-tuned the rules' scores based on our experience and kept as future work further rigorous tuning (e.g., a data-driven adjustment done by conducting user studies). Also, we note that GraphScape's transition cost is not applied to this prioritization process, which also remains as future work. Finally, Gemini^2 imports GraphScape's chart sequence recommendations as keyframe recommendations for its staged animations.

Table 6.1: Pre-defined rules for prioritizing edit operation orders.

Condition & Description (Score)
Filter → Aggregate/Bin Data should be filtered before being aggregated or binned since aggregated/binned data marks make it difficult to show what points are being filtered out. Likewise, data should be dis-aggregated/un-binned before being filtered out. (Score: 1)
Aggregate → Bin Data should *not* be aggregated before being binned since data are aggregated by each bin. Likewise, data should *not* be un-binned before being dis-aggregated. (Score: -1)
Marktype → Aggregate Marktype should *not* be changed before data are aggregated since some marktypes cannot support raw (un-aggregated) data, such as `bar` and `line` in Vega-Lite. Likewise, data should *not* be dis-aggregated before changing marktypes. (Score: -1)
Add/Remove Encoding → Aggregate New encodings should be added before data are aggregated so that the aggregated distributions can be shown in the extended encoding. Likewise, data should be disaggregated before removing encodings. (Score: 1)
Modify Encoding with Scale When modifying an encoding channel, its corresponding scale changes should occur together so users can perceive these change as a single field change. For example, changing an existing variable A on the x-axis to B while modifying its scale to log scale should occur synchronously so users can see A → log(B) instead of A → B → log(B). (Score: 1)
Multiple Filters Multiple filter operations should *not* take place together since audiences cannot identify each filter operation. (Score: -1)
Bin and Aggregate Bin should be conducted with aggregate operations. Otherwise, the binned data points cause occlusions. (Score: 1)

Animation Recommendations with Keyframes

Gemini^2 can recommend staged animations for a given keyframe sequence $((k_i))$ and the number of stages (M). It is extension of Gemini's recommendation, which is a case when the given keyframe consists of start and end charts. The formal representation is:

$$Anim.Recom : \{(k_1, k_2, ..., k_N), M\} \rightarrow (H_1, H_2, ...)$$
$$\text{where } H_i = (g^i_{1\rightarrow 2}, g^i_{2\rightarrow 3}, ..., g^i_{N-1\rightarrow N}).$$

The recommendation process leverages Gemini's recommendation feature. Gemini^2 first gets staged animation designs for each consecutive chart pair through the Gemini recommendation

feature

$$G_{i \to i+1} = (g^1_{i \to i+1}, g^2_{i \to i+1}, ...).$$

In turn, it conducts a cross-join across each pair's recommendations. The cross-join enumerates Gemini-spec arrays (H_i), each of which consists of $N-1$ Gemini specs for every pair. Gemini^2 includes only those with a total number of Gemini spec stages equal to the given number of stages.

$$Candidates = \{H_i | \forall i, \sum_j stage(g^i_{j \to j+1}) = M \\ \& \; H_i \in G_{1 \to 2} \times G_{2 \to 3} \times ... \times G_{N-1 \to N+1}\}$$

The enumerated Gemini^2 animation specs are ranked by their total complexity scores using Gemini's complexity measure:

$$f_{\text{total complexity}}(H_i) = \sum_{j=1} f_{\text{complexity}}(g^i_{j \to j+1}).$$

Gemini's complexity measure ($f_{\text{complexity}}(g)$) is a proxy for human effort to follow the animated changes (see Chapter 5). Gemini recommends staged animation designs with lower complexity measures. Similarly, Gemini^2 uses the summation of the complexity as an evaluation metric. Doing so is still valid because (1) Gemini's complexity is designed to be summed across each stage, and (2) each consecutive chart pair forms a stage. As a result, Gemini^2 prioritizes the Gemini spec array with a low total complexity score.

6.3.2 Usage Scenario

We now demonstrate how Gemini^2 lets users create animations with keyframes through an example scenario. Here, a virtual author, K, creates Kim et al.'s two versions of staged animations, *staged elaborate* and *staged basic*, conveying a median aggregate operation for a 1D data distribution [39].

K starts to author *staged elaborate* by demonstrating its keyframe sequence in Vega-Lite charts (Figure 6.3, left). Once the keyframes are set up, K uses Gemini^2 to automatically

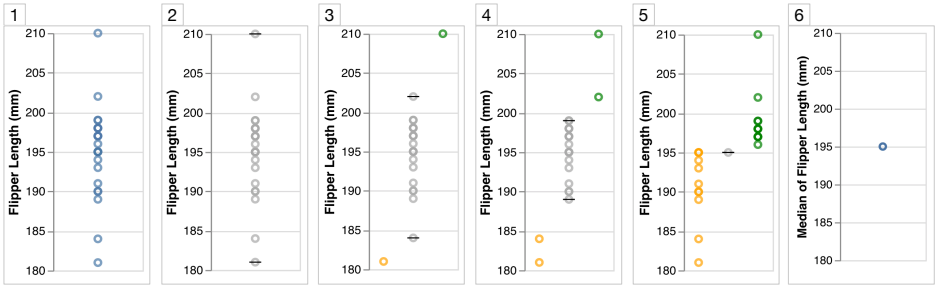


Figure 6.3: Example Gemini^2 keyframes for the staged elaborate animation for the median aggregate operation [39].

generate a basic staged animation through its recommendations for the keyframe sequence. Then, K finalizes by editing a small amount of Gemini^2-generated animation specifications to control the timing of each stage, including the pause and duration of each stage, and a staggering effect on 4→5. The result can be simplified to a *staged basic* version simply by removing keyframes 2, 3, and 4.

This authoring process is not available in Gemini since it cannot support defining intermediate states with extra mark layers that were not included in start and end charts.

6.4 User Study

We conducted a user study to assess whether Gemini^2 recommended more compelling staged animation designs than Gemini and whether its recommendation ranking was reliable.

We recruited 51 participants (36 males, 15 females) from Amazon Mechanical Turk. In the study, we provided 4 different types of transitions between statistical graphics. For each transition, subjects ranked 5 corresponding animation designs: two were Gemini's best recommendations for 2- and 3-stage animations (`g-2s`, `g-3s`), and two were Gemini^2's best recommendations for 2- and 3-stage animations (`g2-2s`, `g2-3s`) except for Stimulus 4. In Stimulus 4, Gemini^2's best is the same as the Gemini's best; therefore, we used 2- and 3-stage animations with lower keyframe recommendation scores to check if the score also aligned with subjects' preference. The fifth one was a not-staged animation (`no-stage`) directly interpolating all changes. We included the not-staged animation to determine if Gemini^2

can recommend compelling staged animations for complex transitions via comparisons.

We analyzed collected users' ranking within each stimulus instead of merging them since Gemini^2 gave them different transition types and different animations rankings. We investigated animation designs' significant rank differences using the Friedman rank-sum test and Wilcox pairwise comparisons with Bonferroni corrections. Figure 6.4 shows bootstrapped means and 95% confidence intervals of subjects' ranking across Gemini^2's recommendation ranking.

Stimulus 1: Encoding & Data Transform & Marktype. The first stimulus set concerns a transition that modifies a given chart's x-axis variable, aggregates its y-axis value, and changes marktype from point to bar. The main difference between Gemini's and Gemini^2's staged animation designs was handling the legend; Gemini^2 staged the legend changes according to the intermediate keyframe, but Gemini did not stage them. On average, subjects voted `g2-s3` as the best and `no-stage` as the worst. However, we could find no significant differences among the animations' rankings. Multiple subjects who ranked `no-stage` fifth reported that the animation "jumbled" the changes too quickly to understand how the data was transformed. This result implies that Gemini^2 can recommend staged animations for complex transitions as well as Gemini can.

Stimulus 2: Two Filters. In this set, animations showed two filtering operations on a COVID-19 daily positive case chart; the first increased the time range of the chart, and the second introduced data from the other regions ("Other States"). Gemini^2's staged recommendations separated the data transforms into two steps: extending existing data for the new time range and introducing the other regions' data on top of the existing data. But Gemini's staged animations did not separate these data changes. Regardless of these staging differences, subjects preferred `no-stage` but without a significant difference in the rankings. One possible interpretation here is that these two specific filtering operations should be conveyed together since they resemble each other in terms of showing a broader view.

Stimulus 3: Encoding & Data Transform. This set is about a transition that added a y-axis with a new categorical field and aggregated data into means by the categorical field.

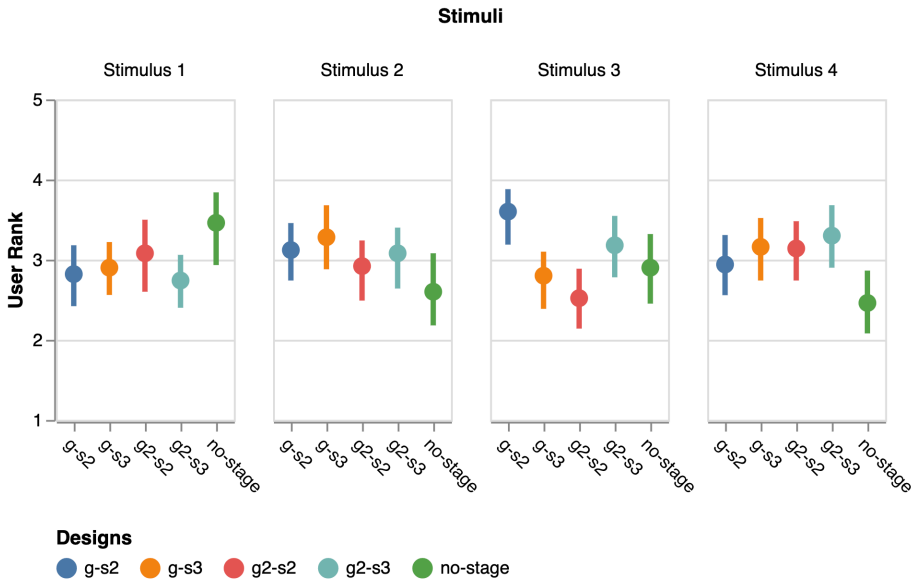


Figure 6.4: Bootstrapped means and 95% confidence intervals for each animation design per stimulus.

Gemini recommended `g-2s` and `g2-2s` as the two best 2-stage animation designs. However, Gemini^2 picked `g2-s2` as the sole best 2-stage animation since it prioritized keyframe sequences by applying an add encoding edit operation before an aggregate edit operation. Participants ranked `g-2s` and `g2-2s` as the worst and the best ($p < 0.05$), respectively. This result suggests that Gemini^2's keyframe sequence prioritization can distinguish staged animations that people may prefer.

Stimulus 4: Bin & Aggregate. The transition of this set changed a 2D scatter plot to a 2D histogram by binning and aggregating. As previously mentioned, `g-s2` and `g-s3` were the best staged animation designs in Gemini^2 as well as in Gemini. Gemini^2 demoted `g2-s2` and `g2-s3` since they separated the bin and aggregated edit operations. The observed subjects' ranking corroborates this prioritization on average, but we found no significant differences.

6.5 Conclusion and Future Work

In this work, we presented Gemini^2, a system for generating keyframe-oriented animated transitions between statistical graphics. We extended GraphScape and Gemini prior work to make Gemini^2 represent animations using keyframe sequences and to automate animations for a given transition or keyframe sequence. Gemini^2 is available as open-source software at `https://github.com/uwdata/gemini`.

Gemini^2 offers challenging and exciting future work opportunities. First, it is at present limited to represent animations among basic single-view Vega-Lite graphics supported by GraphScape. To support varied types of animated transitions, Gemini and GraphScape should be extended to handle more types of Vega/Vega-Lite charts. In terms of ease-of-use, one promising future direction is designing user interactions in animation authoring tools to exploit Gemini^2's recommendation features. Recently, Data Animator [68] introduced novel user interfaces to author keyframe animations between data graphics. The challenge is how to seamlessly blend Gemini^2's recommendation feature with such data graphic animation tools to help users explore alternative stagings while reducing their labor. Finally, it would be promising to deploy additional animation perception studies to improve Gemini^2's recommendation quality. Beyond observing users' preferences, measuring how well people understand or can identify changes through varied staged animation designs will help recommendation systems guide users toward increasingly effective designs.

Chapter 7

CONCLUSION & FUTURE WORK

As statistical graphics are often effective for depicting only a handful of data dimensions, people regularly use more than one graphic to analyze and communicate data. Therefore, transitions between those graphics are critical for conveying cohesive insights and build a holistic understanding across charts. Animation techniques have been employed to help people follow transitions, but they require significant effort to create (e.g., programming, design variations) and there is skepticism around their effectiveness.

This dissertation facilitates the authoring of transitions among statistical graphics by contributing (1) programmatic models and algorithms for automating animation design, and (2) design knowledge for animating transitions. The models and algorithms allow machines to formally represent and reason about the designs of transitions and animations. The design knowledge from empirical studies gives a better understanding of human perception of animated transitions. Finally, all contributed artifacts and empirical study data are open-sourced to encourage follow-up work.

As a result of these works, visualization designers can more easily generate animations for given transitions and explore various designs via recommendations. Moreover, my work can serve as a foundation for computational reasoning about animated transitions: machines can evaluate the effectiveness of an animated transition based on empirical evidence from human-subject studies and theories of event perception. This work seeks to expand the focus of automated design research from single individual data visualizations to their relationships and sequential presentation.

7.1 Future Work

Still, there are many challenging future research opportunities. First, there is a lack of understanding about how users can seamlessly interact with the Gemini specifications and recommendations. Evaluating the Gemini grammar as a domain-specific language will allow better formal representations that facilitate technical communication, including human-human and human-machine communications. On top of this underlying language design concern, better user interfaces for Gemini specifications and recommendations are needed. In the context of static data visualizations, researchers have studied graphical interfaces [43, 54, 61] and exploration tools with recommendation features [76, 77]. Corresponding research for the animation authoring process, such as adding Gemini recommendation features on Data Animator [68], will be the next step for facilitating animated transitions.

In terms of the expressiveness of the Gemini grammar, there is plenty of room for improvement. Gemini and GraphScape are limited to animated transitions between single-view statistical graphics, which have Cartesian coordinates with at most one x-axis and y-axis. More general graphics, such as geographic visualizations, multiple views, and pie charts, need the grammar to be extended. The Gemini grammar may need new types of components (e.g., maps, row/column headers, arcs) and shape interpolation functions among components (e.g., morphing bars to arcs). In addition, the current Gemini grammar does not support assignments of different timing parameters to different properties within the same component. More nuanced timings require the grammar to be able to select each of the sub-properties independently and assign specific timings.

Regarding recommendation, this work evaluates animation designs by using manually tuned rules that depend on heuristics and an empirical study. This hand-tuning approach is not robust since it requires administrators to update the rules whenever new corner cases are discovered or the grammar gets updated. Moreover, the tuning may get tricky as the rules become complicated. One way to respond to this issue is by employing a knowledge-based recommendation system [42, 47] that can easily integrate new design guidelines and automat-

ically optimize rules' weights. Two challenges here are (1) formally representing animation design guidelines within constraint programming and (2) collecting sufficient experimental data to learn appropriate constraint weights.

I evaluate animated transitions with high-level prompts: participants rate or rank how well transition stimuli unambiguously convey changes. However, a deeper understanding of how exactly animations and transitions help users relate charts demands finer grained evaluations with relevant low-level tasks. For example, measuring how quickly audiences can decode subsequent visual encodings after perceiving transitions or examining where they are looking during animations with eye-trackers may better reveal low-level mechanisms of transition perception.

BIBLIOGRAPHY

[1] Sameer Agarwal, Josh Wills, Lawrence Cayton, Gert R. G. Lanckriet, David J. Kriegman, and Serge J. Belongie. Generalized non-metric multidimensional scaling. *JMLR W&P (AISTATS 2007)*, 2:11–18, 2007.

[2] Danielle Albers Szafir Alper Sarikaya, Michael Gleicher. Design factors for summary visualization in visual analytics. *Computer Graphics Forum*, 37(3):145–156, 2018.

[3] Fereshteh Amini, Nathalie Henry Riche, Bongshin Lee, Andres Monroy-Hernandez, and Pourang Irani. Authoring data-driven videos with dataclips. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):501–510, Jan 2017.

[4] Anushka Anand and Justin Talbot. Automatic selection of partitioning variables for small multiple displays. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):669–677, 2016.

[5] Daniel Asimov. The grand tour: a tool for viewing multidimensional data. *SIAM journal on scientific and statistical computing*, 6(1):128–143, 1985.

[6] Dale J. Barr, Roger P. Levy, Christoph Scheepers, and Harry Tily. Random effects structure for confirmatory hypothesis testing: Keep it maximal. *Journal of Memory and Language*, 68(3):255–278, 2013.

[7] Leilani Battle and Jeffrey Heer. Characterizing exploratory visual analysis: A literature review and evaluation of analytic provenance in tableau. *Computer Graphics Forum*, 38(3):145–159, 2019.

[8] Benjamin B. Bederson and Angela Boltman. Does animation help users build mental

maps of spatial information? In *Proceedings 1999 IEEE Symposium on Information Visualization (InfoVis'99)*, pages 28–35, Oct 1999.

[9] Jacques Bertin. *Sémiologie Graphique*. Gauthier-Villars, 1967.

[10] Michael Bostock and Jeffrey Heer. Protovis: A graphical toolkit for visualization. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1121–1128, Nov 2009.

[11] Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. D3 data-driven documents. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2301–2309, Dec 2011.

[12] Steven P. Callahan, Juliana Freire, Emanuele Santos, Carlos Eduardo Scheidegger, Cláudio T. Silva, and Huy T. Vo. Managing the evolution of dataflows with vistrails. In *Proc. 22nd International Conference on Data Engineering Workshops (ICDEW'06)*. IEEE, 2006.

[13] Stuart K. Card, Peter Pirolli, Mija M. Van Der Wege, Julie Bauer Morrison, Robert W. Reeder, Pamela K. Schraedley, and Jenea Boshart. Information scent as a driver of web behavior graphs: Results of a protocol analysis method for web usability. In *Proc. ACM Human Factors in Computing Systems (CHI)*, pages 498–505. ACM, 2001.

[14] Richard Catrambone and A. Fleming Seay. Using animation to help students learn computer algorithms. *Human Factors*, 44(3):495–511, 2002. PMID: 12502166.

[15] Bay-Wei Chang and David Ungar. Animation: From cartoons to the user interface. In *Proceedings of the 6th Annual ACM Symposium on User Interface Software and Technology*, UIST '93, page 4555, New York, NY, USA, 1993. Association for Computing Machinery.

[16] Fanny Chevalier, Pierre Dragicevic, and Steven Franconeri. The not-so-staggering effect of staggered animated transitions on visual tracking. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2241–2250, Dec 2014.

[17] William S. Cleveland and Robert McGill. Graphical perception: Theory, experimentation, and application to the development of graphical methods. *Journal of the American Statistical Association*, 79:531–554, 1984.

[18] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. MIT Press, 3rd edition, 2009.

[19] Tarik Crnovrsanin, Shilpika, Senthil Chandrasegaran, and Kwan-Liu Ma. Staged animation strategies for online dynamic networks. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):539–549, 2021.

[20] CSS Transitions. W3c working draft, 11 October 2018.

[21] Kunin Daniel, Guo Jingru, Dae Devlin Tyler, and Xiang Daniel. Seeing theory. [Online; accessed 26-March-2018].

[22] Pierre Dragicevic, Anastasia Bezerianos, Waqas Javed, Niklas Elmqvist, and Jean-Daniel Fekete. Temporal distortion for animated transitions. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, pages 2009–2018, New York, NY, USA, 2011. ACM.

[23] Fan Du, Nan Cao, Jian Zhao, and Yu-Ru Lin. Trajectory bundling for animated transitions. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI '15, pages 289–298, New York, NY, USA, 2015. ACM.

[24] Niklas Elmqvist, Pierre Dragicevic, and Jean-Daniel Fekete. Rolling the dice: Multidimensional visual exploration using scatterplot matrix navigation. *IEEE transactions on Visualization and Computer Graphics*, 14(6):1539–1148, 2008.

[25] Max Fisher and Amanda Taub. Is there something wrong with democracy? Video, January 2018. Retrieved April 06, 2020 from https://youtu.be/qdK5uK7B0Lo?t=70.

[26] Thomas Frank and Ollie Johnston. Disney animation–the illusion of life. *Abbeville Pub*, 1981.

[27] Tong Ge, Yue Zhao, Bongshin Lee, Donghao Ren, Baoquan Chen, and Yunhai Wang. Canis: A High-Level Language for Data-Driven Chart Animations. *Computer Graphics Forum*, 2020.

[28] Cleotilde Gonzalez. Does animation in user interfaces improve decision making? In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '96, pages 27–34, New York, NY, USA, 1996. ACM.

[29] Jeffrey Heer, Jock D. Mackinlay, Chris Stolte, and Maneesh Agrawala. Graphical histories for visualization: Supporting analysis, communication, and evaluation. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1189–1196, 2008.

[30] Jeffrey Heer and George Robertson. Animated transitions in statistical data graphics. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1240–1247, November 2007.

[31] Jeffrey Heer, Fernanda B. Viégas, and Martin Wattenberg. Voyagers and voyeurs: Supporting asynchronous collaborative information visualization. In *Proc. ACM Human Factors in Computing Systems (CHI)*, pages 1029–1038. ACM, 2007.

[32] Jessica Hullman, Eytan Adar, and Priti Shah. Benefitting infovis with visual difficulties. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2213–2222, Dec 2011.

[33] Jessica Hullman, Steven M. Drucker, Nathalie Henry Riche, Bongshin Lee, Danyel Fisher, and Eytan Adar. A deeper understanding of sequence in narrative visualization. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2406–2415, Dec 2013.

[34] Jessica Hullman, Paul Resnick, and Eytan Adar. Hypothetical outcome plots outperform error bars and violin plots for inferences about reliability of variable ordering. *PLOS ONE*, 10(11):1–25, 11 2015.

[35] Christopher D. Hundhausen, Sarah A. Douglas, and John T. Stasko. A meta-study of algorithm visualization effectiveness. *Journal of Visual Languages & Computing*, 13(3):259–290, 2002.

[36] Samuel Huron, Romain Vuillemot, and Jean-Daniel Fekete. Visual sedimentation. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2446–2455, Dec 2013.

[37] T. J. Jankun-Kelly, Kwan-Liu Ma, and Michael Gertz. A model and framework for visualization exploration. *IEEE Transactions on Visualization and Computer Graphics*, 13(2):357–369, 2007.

[38] Alex Kale, Francis Nguyen, Matthew Kay, and Jessica Hullman. Hypothetical outcome plots help untrained observers judge trends in ambiguous data. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):892–902, Jan 2019.

[39] Younghoon Kim, Michael Correll, and Jeffrey Heer. Designing animated transitions to convey aggregate operations. *Computer Graphics Forum*, 38(3):541–551, 2019.

[40] Younghoon Kim and Jeffrey Heer. Gemini: A grammar and recommender system for animated transitions in statistical graphics. *IEEE Transactions on Visualization and Computer Graphics (Submitted)*, 2020.

[41] Younghoon Kim, Kanit Wongsuphasawat, Jessica Hullman, and Jeffrey Heer. Graphscape: A model for automated reasoning about visualization similarity and sequencing. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, CHI '17, pages 2628–2638, New York, NY, USA, 2017. ACM.

[42] Halden Lin, Dominik Moritz, and Jeffrey Heer. Dziban: Balancing agency & automation in visualization design via anchored recommendations. In *Proceedings of the 2020 CHI*

Conference on Human Factors in Computing Systems, CHI 20, page 112, New York, NY, USA, 2020. Association for Computing Machinery.

[43] Zhicheng Liu, John Thompson, Alan Wilson, Mira Dontcheva, James Delorey, Sam Grigg, Bernard Kerr, and John Stasko. Data illustrator: Augmenting vector design tools with lazy data binding for expressive visualization authoring. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, CHI 18, page 113, New York, NY, USA, 2018. Association for Computing Machinery.

[44] Kwan-Liu Ma. Image graphs – a novel approach to visual data exploration. In *Proc. IEEE Visualization*, pages 81–88. IEEE, 1999.

[45] Jock Mackinlay. Automating the design of graphical presentations of relational information. *ACM Transaction on Graphics (TOG)*, 5(2):110141, April 1986.

[46] Jock D. Mackinlay, Pat Hanrahan, and Chris Stolte. Show me: Automatic presentation for visual analysis. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1137–1144, Nov 2007.

[47] Dominik Moritz, Chenglong Wang, Greg L. Nelson, Halden Lin, Adam M. Smith, Bill Howe, and Jeffrey Heer. Formalizing visualization design knowledge as constraints: Actionable and extensible models in draco. *IEEE Transactions on Visualization and Computer Graphics*, 25:438–448, 2018.

[48] Allen Newell. *Human Problem Solving*. Prentice-Hall, 1972.

[49] Brian Ondov, Nicole Jardine, Niklas Elmqvist, and Steven Franconeri. Face to face: Evaluating visual comparison. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):861–871, Jan 2019.

[50] Stephen E Palmer. *Vision Science: Photons to Phenomenology*. MIT press, 1999.

[51] Deokgun Park, Steven M. Drucker, Roland Fernandez, and Niklas Elmqvist. Atom: A grammar for unit visualizations. *IEEE Transactions on Visualization and Computer Graphics*, PP(99):1–1, 2017.

[52] Nancy Pennington and Reid Hastie. Explaining the evidence: Tests of the story model for juror decision making. *Journal of personality and social psychology*, 62(2):189, 1992.

[53] Quantile. [Online; accessed 12-December-2018].

[54] D. Ren, B. Lee, and M. Brehmer. Charticulator: Interactive construction of bespoke chart layouts. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):789–799, 2019.

[55] George Robertson, Roland Fernandez, Danyel Fisher, Bongshin Lee, and John Stasko. Effectiveness of animation in trend visualization. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1325–1332, November 2008.

[56] George G. Robertson, Kim Cameron, Mary Czerwinski, and Daniel C. Robbins. Animated visualization of multiple intersecting hierarchies. *Information Visualization*, 1:50–65, 2002.

[57] George G. Robertson, Jock D. Mackinlay, and Stuart K. Card. Cone trees: Animated 3d visualizations of hierarchical information. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '91, pages 189–194, New York, NY, USA, 1991. ACM.

[58] David Robinson and Thomas Lin Pedersen. gganiamte, 2018. [Online; accessed 15-September-2019].

[59] Hans Rosling. The best stats youv́e ever seen.

[60] Hans Rosling, Rönnlund Anna Rosling, and Ola Rosling. Gapminder trendalyzer, 2003.

[61] Arvind Satyanarayan and Jeffrey Heer. Lyra: An interactive visualization design environment. *Computer Graphics Forum*, 33(3):351–360, 2014.

[62] Arvind Satyanarayan, Dominik Moritz, Kanit Wongsuphasawat, and Jeffrey Heer. Vega-lite: A grammar of interactive graphics. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):341–350, Jan 2017.

[63] Arvind Satyanarayan, Ryan Russell, Jane Hoffswell, and Jeffrey Heer. Reactive vega: A streaming dataflow architecture for declarative interactive visualization. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):659–668, Jan 2016.

[64] Carlos Eduardo Scheidegger, Huy T. Vo, David Koop, Juliana Freire, and Cláudio T. Silva. Querying and creating visualizations by analogy. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1560–1567, 2007.

[65] Edward Segel and Jeffrey Heer. Narrative visualization: Telling stories with data. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1139–1148, Nov 2010.

[66] Chris Stolte, Diane Tang, , and Pat Hanrahan. Polaris: a system for query, analysis, and visualization of multidimensional relational databases. *IEEE Transactions on Visualization and Computer Graphics*, 8(1):52–65, Jan 2002.

[67] J. Thompson, Z. Liu, W. Li, and J. Stasko. Understanding the design space and authoring paradigms for animated data graphics. *Computer Graphics Forum*, 39(3):207–218, 2020.

[68] John Thompson, Zhicheng Liu, and John Stasko. Data animator: Authoring expressive animated data graphics. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, CHI '17, pages 2628–2638, New York, NY, USA, 2017. ACM.

[69] Perry W Thorndyke. Cognitive structures in comprehension and memory of narrative discourse. *Cognitive psychology*, 9(1):77–110, 1977.

[70] Barbara Tversky, Julie Bauer Morrison, and Mireille Bétrancourt. Animation: can it facilitate? *International Journal of Human-Computer Studies*, 57(4):247–262, 2002.

[71] Manasi Vartak, Sajjadur Rahman, Samuel Madden, Aditya G. Parameswaran, and Neoklis Polyzotis. Seedb: Efficient data-driven visualization recommendations to support visual analytics. *Proceedings of the VLDB Endowment*, 8(13):2182–2193, 2015.

[72] W3C. W3c recommendation, 21 February 2013.

[73] Yong Wang, Daniel Archambault, Carlos E. Scheidegger, and Huamin Qu. A vector field design approach to animated transitions. *IEEE Transactions on Visualization and Computer Graphics*, 24(9):2487–2500, September 2018.

[74] Hadley Wickham. A layered grammar of graphics. *Journal of Computational and Graphical Statistics*, 19(1):3–28, 2010.

[75] Leland Wilkinson. *The Grammar of Graphics*. Springer, 1999.

[76] Kanit Wongsuphasawat, Dominik Moritz, Anushka Anand, Jock Mackinlay, Bill Howe, and Jeffrey Heer. Voyager: Exploratory analysis via faceted browsing of visualization recommendations. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):649–658, Jan 2016.

[77] Kanit Wongsuphasawat, Zening Qu, Dominik Moritz, Riley Chang, Felix Ouk, Anushka Anand, Jock Mackinlay, Bill Howe, and Jeffrey Heer. Voyager 2: Augmenting visual analysis with partial view specifications. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, CHI '17, pages 2648–2659, New York, NY, USA, 2017. ACM.

[78] Stephanie Yee and Tony Chu. A visual introduction to machine learning, 2015. [Online; accessed 06-April-2020].

[79] Jeffrey M Zacks and Barbara Tversky. Event structure in perception and conception. *Psychological bulletin*, 127(1):3, 2001.

[80] Douglas E. Zongker and David H. Salesin. On creating animated presentations. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA 03, page 298308, Goslar, DEU, 2003. Eurographics Association.

[81] agatay Demiralp, Michael S. Bernstein, and Jeffrey Heer. Learning perceptual kernels for visualization design. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):1933–1942, 2014.