GraphScape: A Model for Automated Reasoning about Visualization Similarity and Sequencing

Younghoon Kim, Kanit Wongsuphasawat, Jessica Hullman, Jeffrey Heer
University of Washington
Seattle, WA, USA
{yhkim01, kanitw}@cs.washington.edu, {jhullman, jheer}@uw.edu

ABSTRACT

We present GraphScape, a directed graph model of the visualization design space that supports automated reasoning about visualization similarity and sequencing. Graph nodes represent grammar-based chart specifications and edges represent edits that transform one chart to another. We weight edges with an estimated cost of the difficulty of interpreting a target visualization given a source visualization. We contribute (1) a method for deriving transition costs via a partial ordering of edit operations and the solution of a resulting linear program, and (2) a global weighting term that rewards consistency across transition subsequences. In a controlled experiment, subjects rated visualization sequences covering a taxonomy of common transition types. In all but one case, GraphScape's highest-ranked suggestion aligns with subjects' top-rated sequences. Finally, we demonstrate applications of GraphScape to automatically sequence visualization presentations, elaborate transition paths between visualizations, and recommend design alternatives (e.g., to improve scalability while minimizing design changes).

ACM Classification Keywords

H.5.2 Information Interfaces and Presentation: UI

Author Keywords

visualization; sequence; transition; model; automated design

INTRODUCTION

Data visualizations such as statistical charts are often viewed in sequence. Users of visualization tools may transition among plots during exploratory analysis [8, 9], view sets of charts suggested by a recommender [20, 23], or author a presentation [11, 18] 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 [14, 15, 23], less attention has been paid to modeling visualization *sequences*. In the area of narrative visualization [18], Hullman et al. [11] 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.

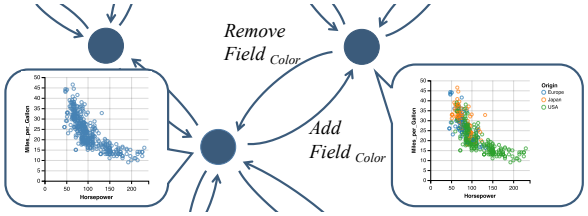

Figure 1. A GraphScape model. Nodes represent Vega-Lite chart specifications, edges represent edit modifications between charts. Edge weights encode estimated transition costs between charts.

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 [16], 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.

CHI 2017, May 6-11, 2017, Denver, CO, USA.

ACM ISBN 978-1-4503-4655-9/17/05 ...$15.00.
http://dx.doi.org/10.1145/3025453.3025866

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.

RELATED WORK

GraphScape extends prior work on modeling visualization state spaces and grammar-based visualization specification.

Modeling Visualization State Spaces

Multiple projects have proposed models of visualization state spaces, often in order to analyze user exploration. The P-Set [12] and Image Graph [13] 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 [10] 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 [4, 17] analyzes visualization workflows in order to track provenance and automatically suggest operations based on prior activity. Tableau graphical histories [8] 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 [9] 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 [11] 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.

Visualization Recommender Systems

Visualization recommender systems also model a space of visualizations and use an objective function to determine which charts to show. The Voyager [23], SeeDB [20] and automatic variable partitioning [2] projects all enumerate a space of possible plots and rank them according to statistical properties or perceptual effectiveness measures [5, 14]. GraphScape similarly provides methods for enumeration and ranking of visualizations. However, while recommendation systems focus on suggesting individual charts, GraphScape focuses on the relationships between charts, for example to sequence charts or elaborate paths between them. GraphScape is thus complementary to existing recommender systems: after a recommender suggests *which* charts to show, GraphScape can *order* the presentation of those charts to facilitate reading.

Grammar-Based Visualization Specification

GraphScape also extends work on grammar-based visualization specification. Visualization grammars such as Wilkinson's Grammar of Graphics [22], the Stanford Polaris [19] system (now commericialized as Tableau), and Wickham's ggplot2 [21] can concisely express a range of customized visualizations and provide a formalism for reasoning about the visualization design space. However, most treatments of visualization grammars focus on their use for manual chart creation. GraphScape provides a means for reasoning about the design space spanned by these grammars by modeling the relationships between individual charts in terms of the specification edits needed to transform one chart into another.

We implemented GraphScape in the context of the Vega-Lite [16] specification language. First introduced to power the Voyager [23] system, Vega-Lite is a high-level visualization grammar, inspired by Tableau's VizQL, for creating a range of statistical graphics within a Cartesian coordinate system. Vega-Lite specifications include *data* definitions (e.g., to load data from a URL) and a *mark type* that determines the form of geometric mark used (e.g., bars, lines, areas, plotting symbols). *Visual encoding* directives map data fields (and associated data types, such as *quantitative* or *nominal*) to visual *channels* such as position (x, y), *color*, *shape* and *size*. Encoding channels for *row* and *column* enable the creation of trellis plots, subdividing a chart into small multiples. In addition, data fields can be subject to a number of *data transformations*, including log scaling, sorting, filtering, binning, and aggregation (e.g., sum, mean, median, count, *etc.*). A more complete description of the Vega-Lite language is available elsewhere [16].

In this paper we focus on the chart specifications described above. Inclusion of more advanced Vega-Lite features, such as interactive selections and chart composition operators (e.g., layering, concatenation) is left as future work. That said, the GraphScape model could be applied to order individual plots within a composite display such as an information dashboard.

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 [7] 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.

Step 1: Identifying Edit Operations

Informed by prior studies of visualization transitions [9, 11], we constructed a taxonomy of possible edits to Vega-Lite [16] 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 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 the paper we discuss how our cost model accommodates filter parameters.

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.

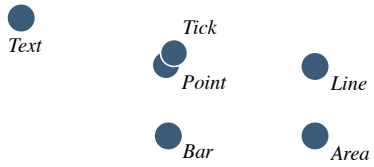

Figure 2. 2D embedding of mark type comparisons. The distance between mark types indicates the estimated transition cost between them.

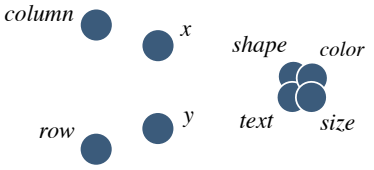

Figure 3. 2D embedding of encoding channel comparisons. Distance indicates the estimated cost of *Move Field* operations between channels.

We then investigated how to rank order edit operations within each transition category. Following the perceptual kernels method of Demiralp et al. [7], we formulated a set of *triplet comparisons*: given a source visualization and two target visualizations, which target 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 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 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

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 ≤ 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 1. Edit operations for transitions among visualization specifications. The table is sorted in ascending order of estimated pairwise cost.

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.

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 [6] 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.

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

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 [11], these results suggest that using consistent transitions within groups of related 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.

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.

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 [16] and edges $e \in E$ are edit operations that turn one specification into another (illustrated in Figure 1). The possible edit operation types are listed in Table 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, $RemoveField_{f,x}$ can not be performed if the x encoding channel is empty, and $AddField_{f,y}$ can not be performed if a field is already 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.

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, $AddField_{count,x}$ incurs only the cost of an *AddField* operation, ignoring the *Aggregate* data transformation cost.

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:

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

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

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.

Global Weighting: Rewarding Consistent Subsequences

Both prior work [11] 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

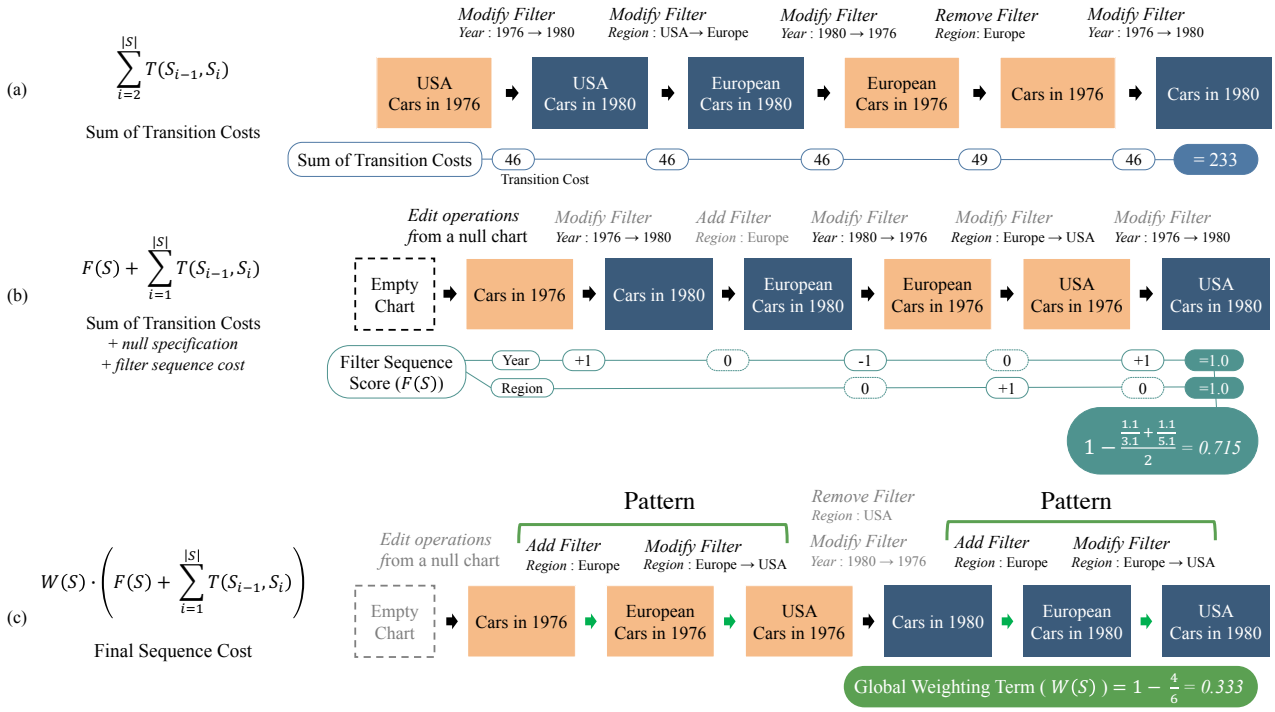

Figure 4. GraphScape sequence cost calculation. (a) Lowest-cost sequence when minimizing the sum of transition costs only. (b) Lowest-cost sequence after adding an initial *null specification* and filter score. (c) Lowest-cost sequence with a global weighting term to reward consistent subsequences.

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(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 $W(S) = 1 - 2 \cdot 2/6 = 1/3$.

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.

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.

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

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.

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 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. [11] and observed in our formative experiment. These styles include measure walks (viewing different quantitative measures against a fixed set

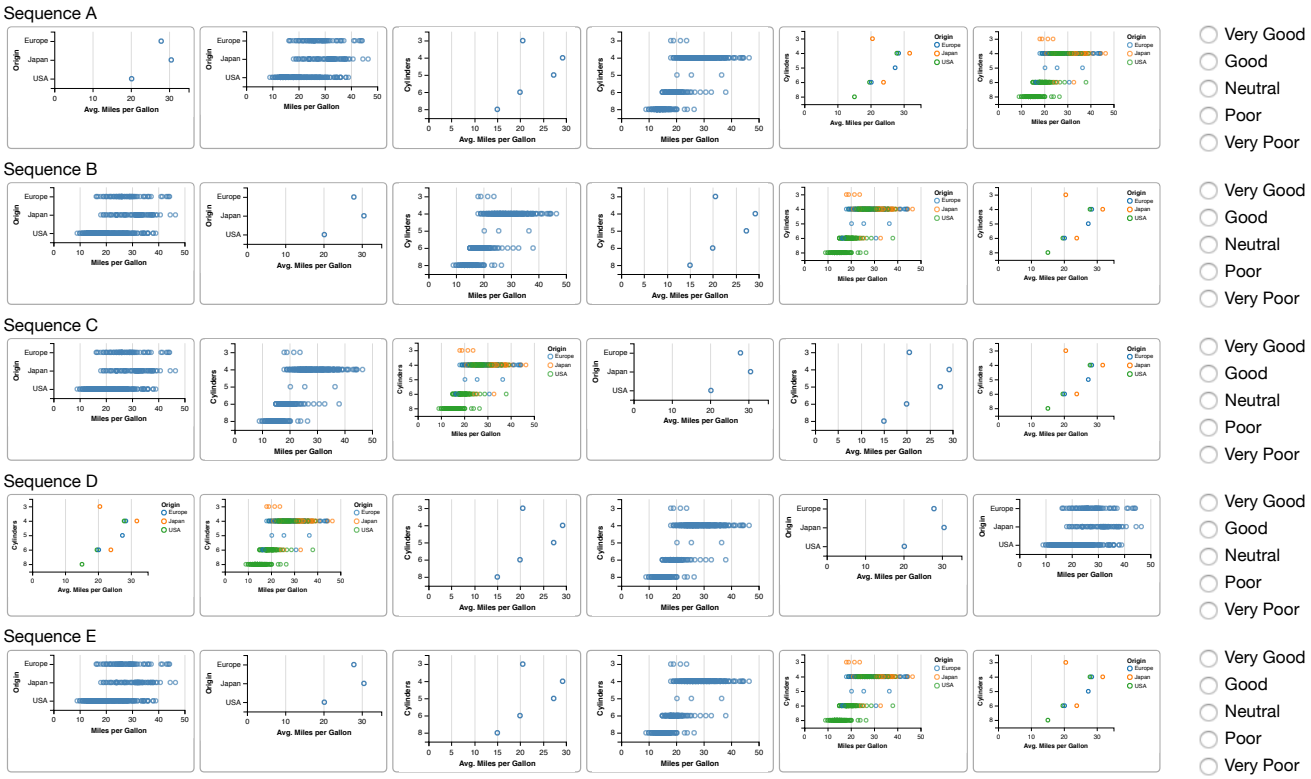

Figure 5. Interface for the sequence rating experiment. Subjects rated how well each sequence presents the data in a clear and logical manner. Sequences were randomly assigned to table rows. Here, sequence B receives the highest rankings from both subjects and the GraphScape cost model.

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 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 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.

Results

We analyzed a total of 1,535 ratings, omitting 115 responses where participants incorrectly answered one or more screening questions. Figure 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, analysis 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 [3] 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 ($\chi^2(1) = 39.87$,

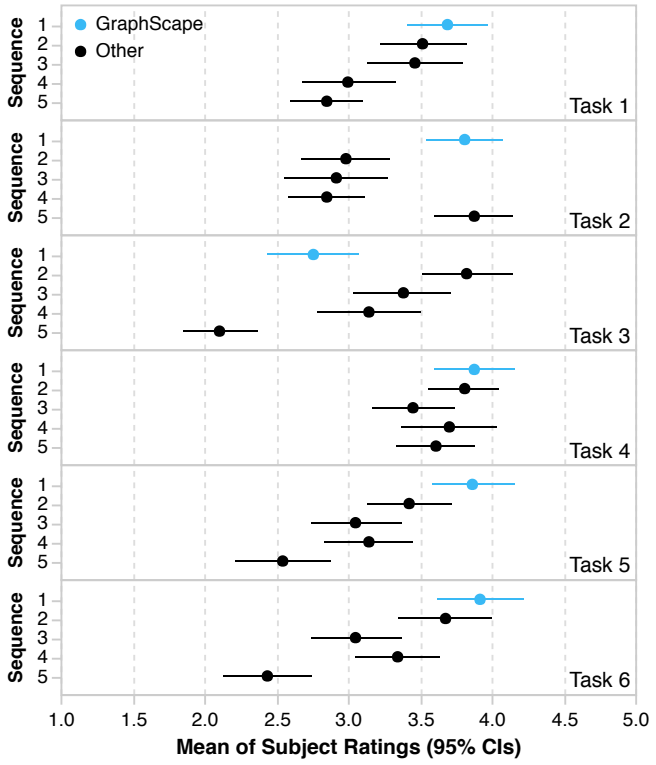

Figure 6. Mean ratings and 95% confidence intervals per sequence, computed via bootstrap. Blue indicates the top GraphScape sequence.

$p < 0.001$). 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 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.

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.

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 optimized 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 [11], 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.

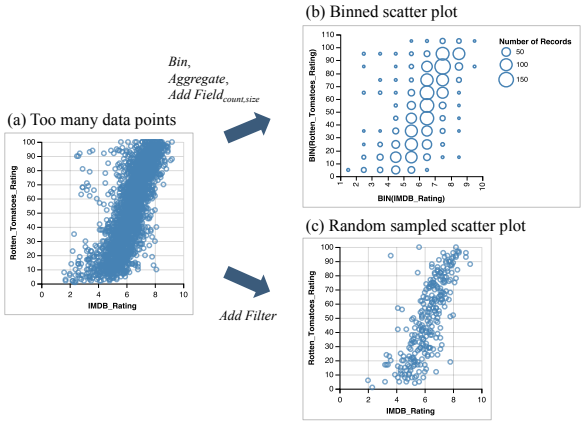

Figure 7. Using GraphScape to search for scalable design alternatives. Given (a) an initial chart with overplotting, we can recommend (b) a binned 2D histogram and (c) a view filtered to a random sample.

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 [9] 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.

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 7 illustrates this 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).

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 [18]. 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 knowledgeable 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.

ACKNOWLEDGMENTS

This work was supported by the Paul G. Allen Family Foundation, a Moore Foundation Data-Driven Discovery Investigator Award, and DARPA XDATA.

REFERENCES

1. S. Agarwal, J. Wills, L. Cayton, G. Lickriet, D. Kriegman, and S. Belongie. 2007. Generalized Non-metric Multidimensional Scaling. *JMLR W&P (AISTATS 2007)* 2 (2007), 11–18.
2. A. Anand and J. Talbot. 2016. Automatic Selection of Partitioning Variables for Small Multiple Displays. *IEEE Transactions on Visualization and Computer Graphics* 22, 1 (2016), 669–677.
3. D. J. Barr, R. Levy, C. Scheepers, and H. J. Tily. 2013. Random Effects Structure for Confirmatory Hypothesis Testing: Keep it Maximal. *Journal of Memory and Language* 68, 3 (2013), 255–278.
4. S. P. Callahan, J. Freire, E. Santos, C. E. Scheidegger, C. T. Silva, and H. T. Vo. 2006. Managing the Evolution of Dataflows with VisTrails. In *Proc. 22nd International Conference on Data Engineering Workshops (ICDEW'06)*. IEEE.
5. W. S. Cleveland and R. McGill. 1984. Graphical Perception: Theory, Experimentation, and Application to the Development of Graphical Methods. *J. Amer. Statist. Assoc.* 79 (1984), 531–554.
6. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. 2009. *Introduction to Algorithms* (3rd ed.). MIT Press.
7. Ç. Demiralp, M. S. Bernstein, and J. Heer. 2014. Learning Perceptual Kernels for Visualization Design. *IEEE Transactions on Visualization and Computer Graphics* 20, 12 (2014), 1933–1942. `http://idl.cs.washington.edu/papers/perceptual-kernels`
8. J. Heer, J. Mackinlay, C. Stolte, and M. Agrawala. 2008. Graphical Histories for Visualization: Supporting Analysis, Communication, and Evaluation. *IEEE Transactions on Visualization and Computer Graphics* 14, 6 (2008), 1189–1196. `http://idl.cs.washington.edu/papers/graphical-histories`
9. J. Heer and G. G. Robertson. 2007. Animated Transitions in Statistical Data Graphics. *IEEE Transactions on Visualization and Computer Graphics* 13, 6 (2007), 1240–1247. `http://idl.cs.washington.edu/papers/animated-transitions`
10. J. Heer, F. B. Viégas, and M. Wattenberg. 2007. Voyagers and Voyeurs: Supporting Asynchronous Collaborative Information Visualization. In *Proc. ACM Human Factors in Computing Systems (CHI)*. ACM, 1029–1038. `http://idl.cs.washington.edu/papers/senseus/`
11. J. Hullman, S. Drucker, N. H. Riche, B. Lee, D. Fisher, and E. Adar. 2013. A Deeper Understanding of Sequence in Narrative Visualization. *IEEE Transactions on Visualization and Computer Graphics* 19, 12 (2013), 2406–2415.
12. T.J. Jankun-Kelly, K.-L. Ma, and M. Gertz. 2007. A Model and Framework for Visualization Exploration. *IEEE Transactions on Visualization and Computer Graphics* 13, 2 (2007), 357–369.
13. K.-L. Ma. 1999. Image Graphs – A Novel Approach to Visual Data Exploration. In *Proc. IEEE Visualization*. IEEE, 81–88.
14. J. Mackinlay. 1986. Automating the Design of Graphical Presentations of Relational Information. *ACM Transactions on Graphics* 5, 2 (1986), 110–141.
15. J. Mackinlay, P. Hanrahan, and C. Stolte. 2007. ShowMe: Automatic Presentation for Visual Analysis. *IEEE Transactions on Visualization and Computer Graphics* 13, 6 (2007), 1137–1144.
16. A. Satyanarayan, D. Moritz, K. Wongsuphasawat, and J. Heer. 2017. Vega-Lite: A Grammar of Interactive Graphics. *To appear in IEEE Transactions on Visualization and Computer Graphics* (2017). `http://idl.cs.washington.edu/papers/vega-lite`
17. C. E. Scheidegger, H. T. Vo, D. Koop, J. Freire, and C. T. Silva. 2007. Querying and Creating Visualizations by Analogy. *IEEE Transactions on Visualization and Computer Graphics* 13, 6 (2007), 1560–1567.
18. E. Segel and J. Heer. 2010. Narrative Visualization: Telling Stories with Data. *IEEE Transactions on Visualization and Computer Graphics* 16, 6 (2010), 1139–1148. `http://idl.cs.washington.edu/papers/narrative`
19. C. Stolte, D. Tang, and P. Hanrahan. 2002. Polaris: A System for Query, Analysis, and Visualization of Multidimensional Relational Databases. *IEEE Transactions on Visualization and Computer Graphics* 8, 1 (2002), 52–65.
20. M. Vartak, S. Rahman, S. Madden, A. Parameswaran, and N. Polyzotis. 2015. SeeDB: Efficient Data-Driven Visualization Recommendations to Support Visual Analytics. *Proceedings of the VLDB Endowment* 8, 13 (2015), 2182–2193.
21. H. Wickham. 2009. *ggplot2: Elegant Graphics for Data Analysis*. Springer.
22. L. Wilkinson. 1999. *The Grammar of Graphics*. Springer.
23. K. Wongsuphasawat, D. Moritz, A. Anand, J. Mackinlay, B. Howe, and J. Heer. 2016. Voyager: Exploratory Analysis via Faceted Browsing of Visualization Recommendations. *IEEE Transactions on Visualization and Computer Graphics* 22, 1 (2016), 649–658.