A Probabilistic Grammar of Graphics

Xiaoying Pu
University of Michigan
Ann Arbor, MI, US
xpu@umich.edu

Matthew Kay
University of Michigan
Ann Arbor, MI, US
mjskay@umich.edu

ABSTRACT

Visualizations depicting probabilities and uncertainty are used everywhere from medical risk communication to machine learning, yet these probabilistic visualizations are difficult to specify, prone to error, and their designs are cumbersome to explore. We propose a Probabilistic Grammar of Graphics (PGoG), an extension to Wilkinson's original framework. Inspired by the success of probabilistic programming languages, PGoG makes probability expressions, such as $P(A|B)$, a first-class citizen in the language. PGoG abstractions also reflect the distinction between probability and frequency framing, a concept from the uncertainty communication literature. It is expressive, encompassing product plots, density plots, icon arrays, and dotplots, among other visualizations. Its coherent syntax ensures correctness (that the proportions of visual elements and their spatial placement reflect the underlying probability distribution) and reduces edit distance between probabilistic visualization specifications, potentially supporting more design exploration. We provide a proof-of-concept implementation of PGoG in R.

Author Keywords

Grammar of Graphics; Uncertainty visualization

CCS Concepts

•**Human-centered computing** → **Visualization theory, concepts and paradigms;** Visualization systems and tools;

INTRODUCTION

Creating effective visualizations of probability distributions is a critical task: there is a growing consensus that we need to show uncertainty in scientific and everyday data, often expressed as probabilities [59, 29, 48]. Visualizations can help communicate uncertainty in ways that draw users' attention to it and help them understand it. For example, during the 2016 US presidential election, the *New York Times* created a forecast "needle" visualization (Figure 1.5), using animated jitter to encode the uncertainty in a real-time prediction for the electoral college margin [51], an example of a hypothetical outcome plot [25]. Other examples of principled uncertainty visualizations have been found to help people comprehend data and make better decisions in domains such as medical risk communication [22], hurricane forecasting [34], and transit prediction [28, 13]. All of these examples belong a broader class of visualizations we call *probabilistic visualizations*: visualizations of (possibly conditional) probability distributions.

Effective probabilistic visualizations can be hard to specify: the *New York Times* election needle is a bespoke JavaScript application, the kind of visualization only a small number of premier data journalism outlets are producing. The implementation of the needle involves the careful handling of probability distributions, sampling, and animation, and these aspects are intertwined in the implementation. Existing declarative formalisms for specifying visualizations, such as the *Grammar of Graphics* [35, 58] do not fully incorporate notions of probability distributions and conditional probabilities, forcing the user to carefully handle probability distributions to ensure the correctness of the output. As a result, visualizations for probabilities are currently difficult to create, their design space costly to explore, and their specification process error-prone.

To address these shortcomings, we propose a Probabilistic Grammar of Graphics (PGoG), a high-level grammar for specifying probabilistic visualizations. Extending the original *Grammar of Graphics* [58], PGoG makes probability distributions first-class citizens in its specifications and defines other grammar components around them. PGoG improves the specification of probabilistic visualizations in two ways: 1) by reinforcing parsing rules on probability data and aesthetics mappings, PGoG guarantees that the visualization reflects the probability distribution the user intends to communicate; 2) by covering a wide range of common statistical graphics with a consistent set of language elements (Figure 3), PGoG aims to enable rapid prototyping and reasoning with minimal edit distances among designs.

The design of PGoG is motivated by calls for "systematic ways of displaying uncertainty" [56] and "a closer integration of visualization and statistical algorithms" [24]. We developed and evaluated PGoG based on principles from cognitive ergonomics [6] — the usability of notational systems. To that end, we follow a similar approach to probabilistic programming, a paradigm where syntax is close to statistical notation while shielding users away from implementation details. Given the success of probabilistic programming for helping users with diverse backgrounds (from biologists to social scientists) specify complex statistical models, we expect to empower similar users to specify probabilistic visualizations.

CHI '20, April 25–30, 2020, Honolulu, HI, USA.

ACM ISBN 978-1-4503-6708-0/20/04 ...$15.00.
http://dx.doi.org/10.1145/3313831.3376466

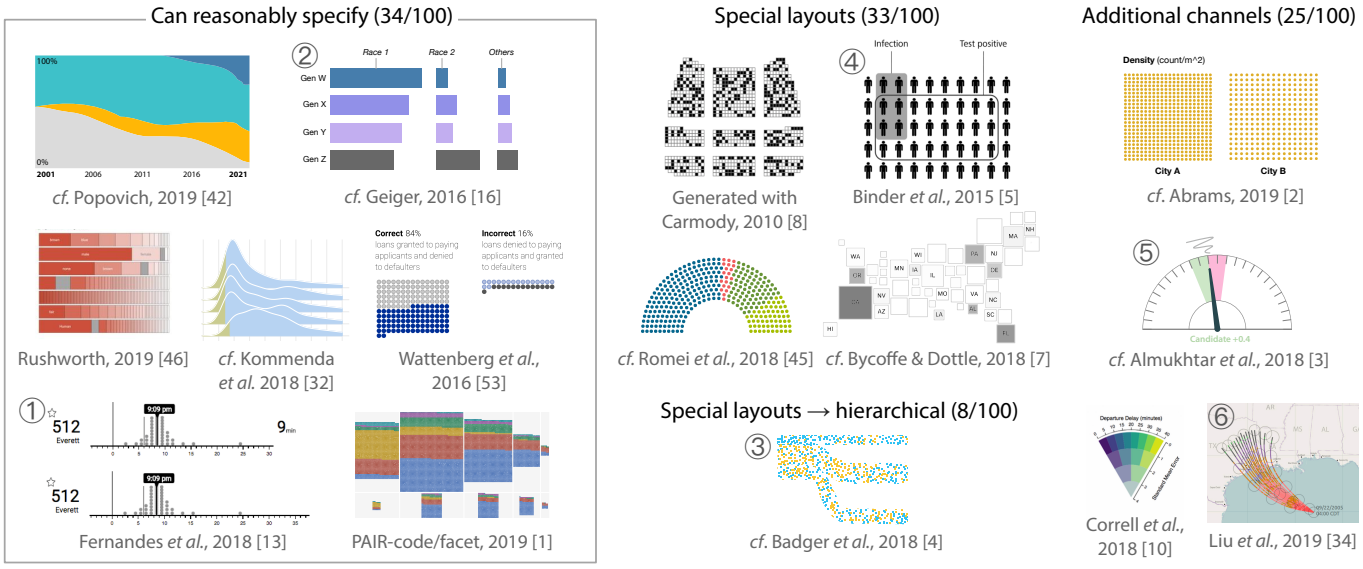


Figure 1. A sample of our probabilistic visualization collection. On the left: a subset of visualizations PGoG *can reasonably specify*. On the right: categories of visualizations that informed the grammar design but are not yet fully reproduced by PGoG; see the *Expressiveness* section of the evaluation for more details and a discussion of how PGoG could be extended to support them. (1) quantile dotplot for uncertain bus arrival times [28]; (2) barchart for America's shifting demographics [16]; (3) icon array that flows, showing social mobility [4]; (4) icon array for medical risk communication [5]; (5) NYT election needle [3]; (6) ensemble of hurricane path predictions [34]. The full collection (N = 100) is in the supplemental materials.

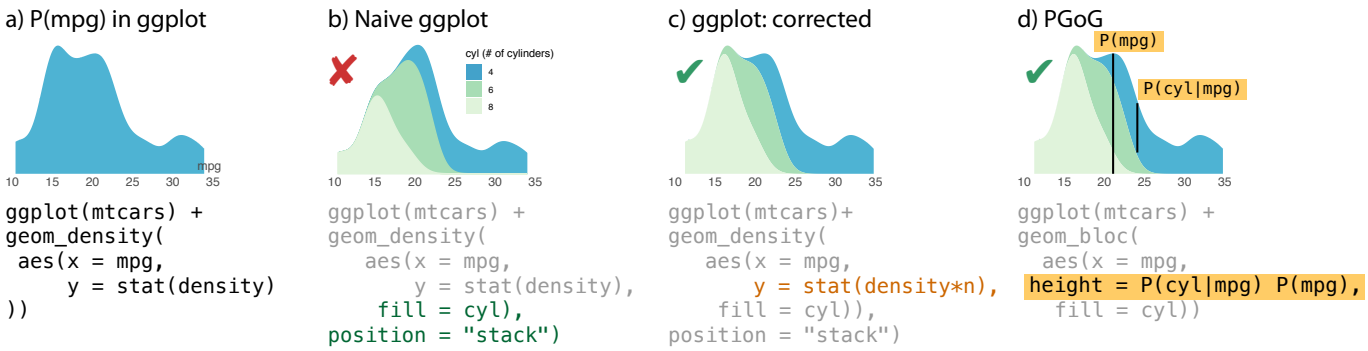


Figure 2. A motivating example for the Probabilistic Grammar of Graphics. a) For reference, a density plot for variable `mpg` alone: $P(mpg)$. b) In base `ggplot2`: naively introducing the variable `cyl` creates partition of the density plot disproportional to the true `cyl` counts. c) In base `ggplot2`: normalizing the colored regions by hacking internal variables (`density * n`) creates a correct stacked density plot. d) PGoG generates the correct density plot using syntax closer to users' statistical language, in terms of probability expressions.

We provide a proof-of-concept implementation of PGoG in the programming language R, based on the visualization library `ggplot2`. Since PGoG is an abstract formalism, we believe that it can be integrated with other declarative visualization frameworks, such as Vega-Lite [47]. PGoG can also serve as a theoretical framework for future work to formalize other visualizations of *uncertainty* and to ground uncertainty visualization research questions.

A MOTIVATING EXAMPLE: SPECIFYING DENSITY PLOTS

Figure 2 shows how current specifications can visualize probability distributions in an incorrect or convoluted manner. The specification code is in `ggplot2`, a popular visualization package in R [21] and the dataset is the Motor Trend Cars Road Tests data in R [43]. At first sight, Figure 2.b looks like a depiction of the distribution of car mileages ($P(mpg)$), proportionally colored by cylinder count `cyl`[1]. Accordingly, the `ggplot2` code contains `fill = cyl`, the common way of expressing "break down by this variable using fill color". Little does the inexperienced user know that Figure 2.b is erroneous, because the system does not understand the laws of probability. Instead of showing the proportion of 8-cylinder cars within the dataset as one would assume, the portion of 8-cylinder cars (in light green) will always be 1/3 no matter the data. We show how `ggplot2` commits this error in Figure 4. Essentially, the system pieces together three separate densities $P(mpg|cyl)$ without normalizing by `cyl` group counts $P(cyl)$, resulting in

[1] *cyl* = number of cylinders in a car engine. *mpg* = miles per gallon, a measure of mileage. *am* = automatic or manual transmission.

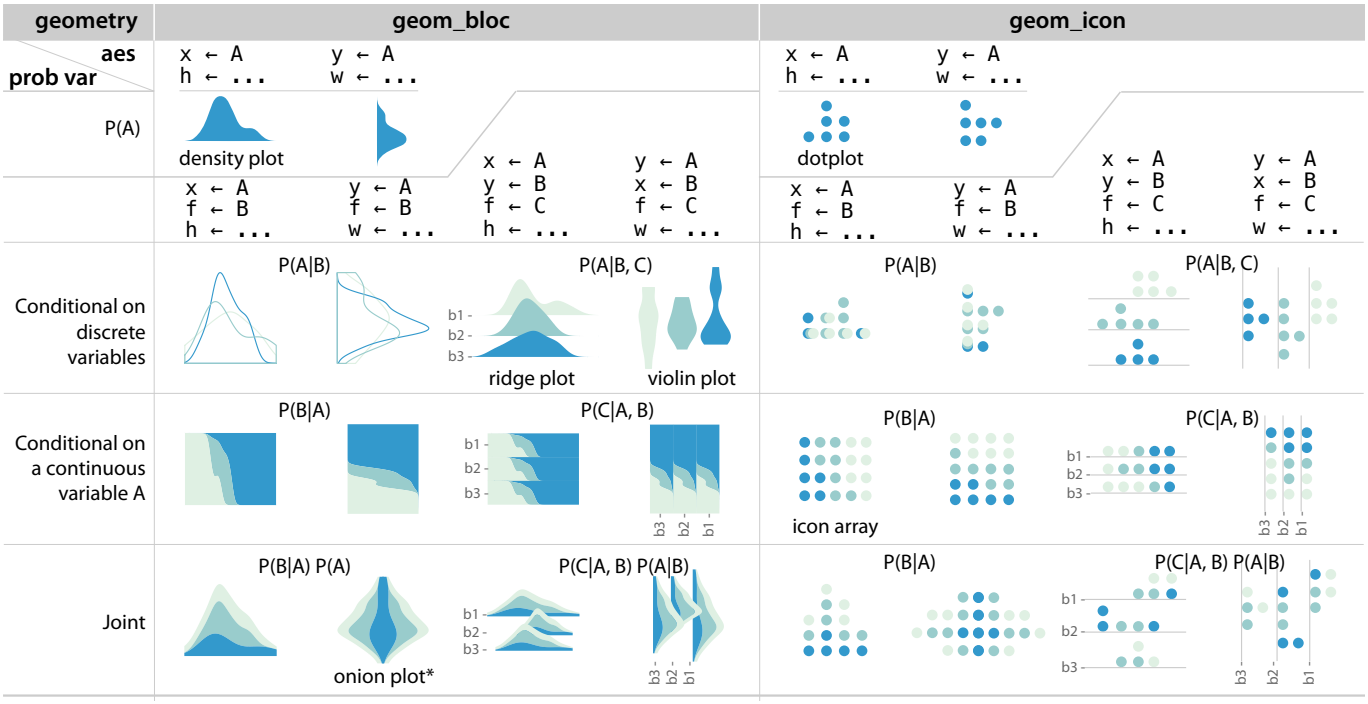


Figure 3. A subset of visualizations described by the PGoG grammar. For `geom_bloc`, we assume that variable A is continuous, and B, C are discrete. (If A is also discrete, `geom_bloc` will produce area plots [56].) `h <- ...` is `height <- [the probabilistic variable in the first column]`. `f` = `fill` aesthetic.

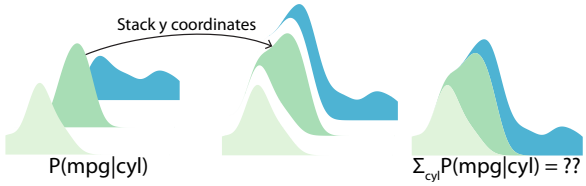


Figure 4. How `ggplot2` constructs Figure 2.b. Left: the system first computes three density estimates, each of unit area, that represent $P(mpg|cyl)$ (`cyl` = 4, 6 and 8). Right: `ggplot2` stacks the densities naively (`position = stack`), creating an incorrect figure: the viewer might interpret there to be roughly 1/3 8-cylinder cars, while in the data, this proportion is 43%. In terms of probability notations, stacking creates a part-whole relationship and thus implies summation, but $\sum_{cyl} P(mpg|cyl)$ does not simplify to $P(mpg)$.

incorrect proportions of `cyl` when the user views the visualization as a whole. In this example, the system's approach to visualizing a bivariate distribution departs from what the user might expect, or what is correct statistically.

For the purposes of this paper, **a probabilistic visualization is *correct* if the proportions of visual elements (such as counts or areas) and their spatial placement reflect the underlying probability distribution**, including any conditional probabilities or part-to-whole relationships. Thus, Figure 2.b is incorrect.

Even when the existing languages produce correct probabilistic visualizations, the necessary specifications can be convoluted. In Figure 2.c and 2.d, we compare how `ggplot2` and our specification, PGoG, describe the same distribution $P(mpg, cyl)$. In Figure 2.c, the normalization term, $P(mpg)$, is implied with the line `stat(density*n)`, which is a hack to access the internal cylinder count n. As Figure 2.c further shows, specifying color (`fill`) is an indirect way of creating a conditional density $P(cyl|mpg)$. This `ggplot2` example illustrates how complex it is to specify even simple probability distributions in existing languages. In comparison, PGoG understands how to map probability expressions directly to visual elements. This allows users who understand probability distributions but are unfamiliar with implementation details to specify precisely the probabilities they wish to depict, analogous to how probabilistic programming lanugages allow users to directly specify statistical models.

RELATED WORK

Probabilistic Programming

The *probabilistic* aspect of PGoG is partly inspired by probabilistic programming languages. These languages use explicit notations to represent probability distributions, with the capacity to condition on (observed) values of variables [19]. For instance, in the Stan language, an independent Bernoulli model can be specified using syntax very close to statistical notations (Figure 1 in Carpenter *et al.* [9]):

```
model {
  theta ~ beta(1, 1);      //prior
  y ~ bernoulli(theta);    //likelihood
}
```

Stan, and probabilistic programming languages in general, hide implementation details of samplers (including scalability improvements), enabling not only statisticians but also a broader user group to specify custom models [18, 49, 41, 9].

As evidence for the value of probabilistic programming for Bayesian inference alone, JAGS [41] and Stan [9] are widely cited (3000+ and 1600+, respectively), with applications in a range of disciplines. PGoG similarly defines visualizations with probability notations (hence the "probabilistic" in name), hiding the complexity of lower-level specification and opportunities for errors. PGoG thus enables users with knowledge of probability distribution notation to more easily create probabilistic visualizations.

Probabilistic visualizations

Probabilistic visualizations have various designs and are used to communicate uncertainty information in many domains, see Figure 1. The probabilities they express can be a probability mass function (pmf) for discrete variables or a probability density function (pdf) for continuous variables. PGoG covers a subset of probabilistic visualizations.

Probability & frequency formats in uncertainty visualization

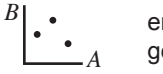
We use a distinction in uncertainty communication [17] — probability *v.s.* frequency formats — to ground the types of probabilistic visualizations covered by PGoG. A *probability format* is a continuous representation, represented verbally with percentages ($x\%$) or visually with *area plots* [56], where the area of a visual element is proportional to its probability. Examples of area plots include variants of pie charts, bar charts, and density plots. A *frequency format* is a discrete representation, represented verbally with fractions or ratios (x-in-100) or visually with *icon*-based (or *unit* [39]) visualizations, which include variants of icon arrays and dotplots.

A designer may want to choose between probability and frequency formats. Frequency format visualizations are often found to help comprehension and decision-making; such is the case with quantile dotplots in a public transit setting in Figure 1.1 [28, 13], and icon arrays in medical risk communication [36, 22, 38]. Figure 1.4 shows an icon array, with patients grouped by disease condition and test results. On the other hand, probability format visualizations, such as density plots, are widely used in scientific communication, and can be more compact to display than icon-based counterparts.

PGoG supports both probability and frequency formats. This feature may help users to quickly explore visualizations of the same probability distribution in either format.

Grammar of Graphics

The power of high-level visualization grammars comes from their coverage of common chart types with a relatively small vocabulary. One notable example of high-level grammars is Wilkinson's *Grammar of Graphics* [58]. In this formalization, a plot is a combination of components, including *data* — "data operations that create variables from datasets" and *elements* — "graphs [*geometries*] and their aesthetic attributes [*aesthetics* or encodings]". As an example, a scatterplot of variable B against A can be specified as:



encoding(*x position* ← *A*, *y position* ← *B*) +
geometry(*point*)

Some widely-used instantiations of the Grammar of Graphics include Vega-lite [47] and Polaris (later Tableau) [50].

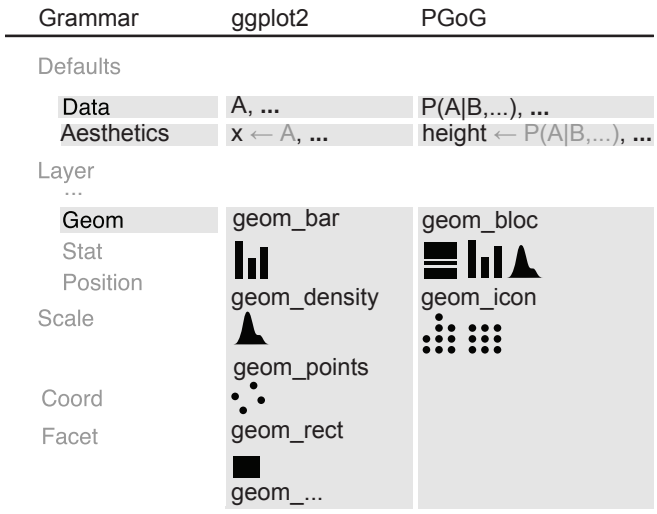


Figure 5. PGoG in the context of the layered grammar of graphics [54]. The leftmost "Grammar" column is adapted from Figure 4 of Wickham's paper [54]. The middle column lists some representative instantiations of layered grammar of graphics components in `ggplot2`. PGoG extends the instances of data, aesthetics, and geometry components from the layered grammar of graphics, shown in the rightmost column.

Most relevant to our work, Wickham's *layered* Grammar of Graphics groups the original components into layers to better embed the grammar into an implementation, namely the `ggplot2` package in R [54]. We reproduce and extend Wickham's illustration of the layered Grammar of Graphics and `ggplot2` components in Figure 5.

One benefit of Grammar of Graphics is that its components leverage structures inherent to visualization to create a modular grammar, instead of using named graphics like "bar chart" [54]. As a result, it is easy to change from one visualization design to another. The previous scatter plot specification can be easily changed to a bar chart:

encoding(*x position* ← *A*, *y position* ← *B*) +
geometry(*bar*)

PGoG conceptually extends the data component in the grammar of graphics, and defines additional aesthetics and geometries to specify probabilistic visualizations.

Convoluted specifications for probabilistic visualizations

While the Grammar of Graphics works well generally, it lacks abstractions for probabilistic visualizations. As a result, specifying probabilistic visualizations can be *ad hoc*. For example, the `ggplot2` package has a proliferation of geometry types for density plot variants alone, such as ones for plain density plots (`geom_density`), violin plots, and ridge plots [21, 57]. Arguably, this *ad hoc* approach diverges from the flexibility and elegance of the Grammar of Graphics; an approach that directly incorporates probability distributions might be more modular and flexible.

Current languages also have *hidden dependencies* that could be inferred from the probabilistic structure of the data, such as the need for manual normalization in the `ggplot2` example

of stacked densities in Figure 2. Similar dependencies exist in another Grammar of Graphics instantiation, Vega-lite [47], where the user must override the default of three different parameters, `extent`, `steps`, and `counts`[2] to create a stacked density. Since parameters set otherwise will not produce a correct stacked density plot, errors are likely; a grammar that understands the underlying probability distributions might prevent such errors.

Specifications for probabilities and visual elements can also be difficult to separate and change. Consider ATOM, a layout-based grammar with a broad coverage for unit visualizations (icon-based probabilistic visualizations) [39]. The ATOM grammar is extremely powerful and creates visualizations by combining low-level layout operations. However, the power of ATOM may come at a cost: its layout operations imply what probability distribution the system computes and visualizes without making it explicit; the probability distribution itself may be hard to interpret from the specification, and it may be hard to ensure that the distribution is preserved if the user changes the layout of the visualization during design exploration. A grammar that directly encodes the desired probability distribution might aid exploration of valid visualizations by inferring layouts from the structure of probability expressions, rather than the other way around.

GRAMMAR DESIGN CONSIDERATIONS AND PROCESS

We derive three design considerations for the PGoG grammar based on the Motivating Example and Related Work:

1. *Guaranteeing correctness:* a grammar for probabilistic visualizations should ensure *correctness* as we define in the Motivating Example. Specifically, the grammar needs to ensure that 1) the probability distributions supplied in the specification are valid, and 2) the visualization accurately reflects the supplied distributions, including any part-to-whole relationships.
2. *Untangling specification:* given that current systems can make specifying probabilistic visualizations convoluted, a grammar should untangle the specification for probabilistic visualizations. Our approach is to center the PGoG grammar around probability expressions, such as $P(A|B)$.
3. *Facilitating exploration:* following other grammars that facilitate exploration, PGoG should be coherent and systematic [11], its components reusable [23]. According to one visualization model, GraphScape, the process of exploring the visualization design space can be thought of as making edits to visualization specifications [30]. From that perspective, easier exploration could be enabled by having shorter edit distances in the space of supported probabilistic visualizations, a property PGoG aims to have.

We defined the PGoG grammar through an iterative process. For several months, we collected and reviewed 100 probabilistic visualizations from academic papers and reputable news outlets, in addition to basic statistical graphics such as density plots and bar charts. The collection is in the Supplemental Materials. We did not intend to reproduce every visualization in the collection. Rather, we used these real-life examples to prototype PGoG components — asking, e.g., *can this aesthetic/geometry describe how this visualization encodes probability?* We then systematically iterated through combinations of the components to refine the grammar rules. Criteria for in-/excluding a particular combinations are mainly based on correctness instead of perceptual or aesthetic properties: *does it produce a probabilistic visualization (as opposed to a non-probabilistic plot)?*, and *does the resulting visualization show the intended distribution?*

[2] https://vega.github.io/vega-lite/docs/density.html

GRAMMAR SPECIFICATION

The PGoG grammar is an extension of the data, aesthetics, and geometry components in the layered *Grammar of Graphics*, see Figure 5. Most notably, probability distributions are made to be first-class citizens as data variables. We also define new aesthetics and geometries to work with probabilities. Not all combinations of data and aesthetics are allowed; we enforce a set of grammar rules to cover existing probabilistic visualizations while ensuring that the generated visualizations can be represented in the 2D plane.

Data variables

We assume that the dataset is in the tidy format [55], *i.e.*, each row represents an observation or a sample from a distribution, and each column is an attribute. Existing high-level visualization libraries such as `ggplot2` and Vega-Lite treat columns as data variables, which we call *simple* variables in PGoG. Simple variables can be discrete or continuous, which later determines the plot type, such as density plot *v.s.* bar charts. In addition, PGoG has *probabilistic* variables, defined in the form $P(A|B,...)$, where $A, B, ...$ are simple variables. Each probabilistic variable is $1D$, with one marginal variable (left of the pipe) and any number of conditionals (right of the pipe). One or more probabilistic variables are used with aesthetics to construct data mappings.

A visualization described by PGoG can represent any probability function $P(A,...|...)$ for discrete variables. In a PGoG specification, this probability function is computed as the product of all probabilistic variables $P(A|...)$ provided. For example, the probability function $P(A,B)$ can be specified with $P(A|B)P(B)$ or $P(B|A)P(A)$. Since there are multiple ways to factor a probability function, PGoG requires the user to supply each factor directly, not just the joint like $P(A,B)$, to eliminate ambiguity in data mapping. We currently have the $1D$ restriction because the probabilistic variables can then correspond to the $1D$ aesthetics (discussed next). Unlike the case with other grammars [56], $2D$ probabilistic variables/aesthetics (e.g. $P(A,B|...)$) are not yet included; we have not yet developed comprehensive rules to ensure the correctness of specifications with such expressions but plan this for future work.

Validating the probabilistic variables

PGoG checks to ensure that the probability factors multiply to a single, valid probability function. The logic for checking follows standard probability rules, outlined in Algorithm 1, which essentially verifies the correctness of the factors using the chain rule for probabilities. PGoG builds and parses a data structure (the "chain") shown in Figure 6. Malformed factorizations, such as $P(A|B)P(A)$, will not pass this check.

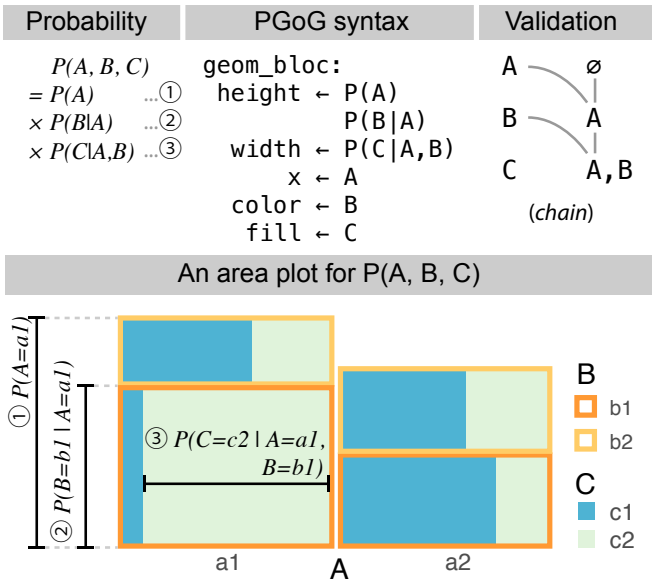


Figure 6. How PGoG turns probabilities into an area plot. PGoG parses aesthetics mappings (middle) with probabilities into a validation data structure ("chain"). The marginal and conditional variables combined from the first level should be the conditional variables of the second level. The PGoG implementation uses this chain data structure to check for malformed probabilities and keep track of visualization layouts.

PGoG orders the factors by the number of conditionals in each factor; each valid specification has only one ordering. The user can thus supply the factors in any order. The probability chain structure is used for checking the factors and later, deriving the visualization layout.

Algorithm 1 Checks if all probability factors are well-formed

```
chain ← order by conditionals length(chain)
legit ← TRUE
for row ∈ chain do
    if next row exists AND legit then
        if next$cond ≠ row$marg ∪ row$cond then
            legit ← FALSE
    if row$marg ∩ row$cond ≠ ∅ then
        legit ← FALSE
assert(legit)
```

Aesthetics

Aesthetics, in the *Grammar of Graphics* model, are plot elements that data variables are *mapped onto* [58]. In PGoG, we add new *probabilistic aesthetics* to accommodate probabilistic variables: `width` and `height`. These $1D$ aesthetics are inspired by Product Plots [56], where the width and height of a rectangle each express a factor of a probability function, and thus the area of the rectangle is the joint probability. During our design process, we find that the concept of `width` and `height` as aesthetics also applies to other probabilistic visualization types. `width` and `height` can be *recursive*, in that we can subdivide one or both dimensions to create partitions

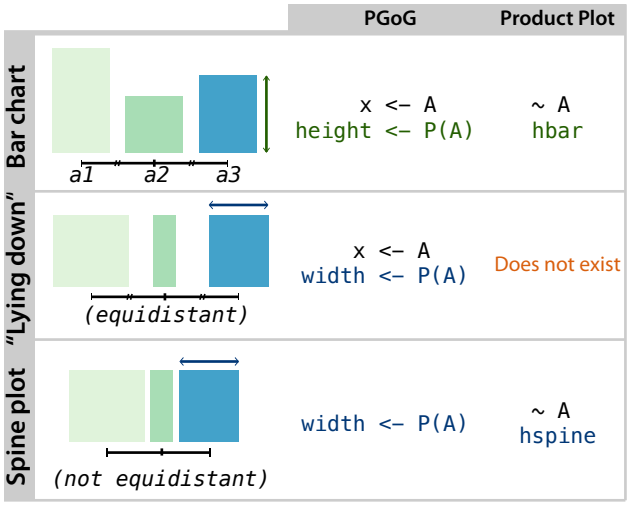


Figure 7. Combinations of aesthetics `x`, `width`, and `height` create different area plots in PGoG. The `x<-A` mapping creates equidistant partitions on the x-axis. Product plot specifications are on the right, which requires the user to learn its visual primitives such as `hbar` (horizontal bar). Figure 3 shows more density plot and dotplot examples in PGoG.

according to marginalization $P(A) = \sum_{b \in B} P(A,B)$. This recursive property of the probabilistic aesthetics makes use of the probability chain structure, as seen in Figure 6: nested conditional probabilities correspond to nested partitions of the area plot. Note how $P(A)$, $P(B|A)$, and $P(C|A,B)$ directly create values that can be read from the chart using the aesthetics they are mapped onto; e.g., $P(B|A)$ is mapped to height, and the recursive layout guarantees that $P(B = b_1 | A = a_1)$ (or other combinations of $B \in \{b_1, b_2\}$ and $A \in \{a_1, a_2\}$) can be read from the chart using height. This even suggests a method to recreate the original TreeMap algorithm [27] using PGoG: alternately mapping conditional probabilities onto height, then width, then height (e.g. `width <- P(A)`, `height <- P(B|A)`, `width <- P(C|B,A)`, `height <- P(D|C,B,A)`, etc). Such a specification reveals the conditional probability structure underlying TreeMaps.

Coordinate aesthetics include the common x and y, the two axes in the Cartesian plane. In traditional visualization specification, we often map two simple variables onto x and y, a combination that can create a scatter plot. In PGoG, using coordinate aesthetics implies conditioning on simple variables. Mapping a simple variable A onto the x-axis, for example, means to condition on A; for discrete variables this creates equal-width partitions along the x-axis, see Figure 7. Unlike traditional specifications where we can only assign one variable to one axis, it is possible to do so with multiple simple variables in PGoG. The coordinate aesthetics could also be easily extended to include polar coordinates in the future; the concept of `width` and `height` can still apply.

It is worth noting that probabilistic and coordinate aesthetics are different. With probabilistic aesthetics, the variation in the visual element (width and height) carries *probabilistic* information. In comparison, coordinate aesthetics lay out *simple* variables on the canvas; they create a uniform partition

on an axis for a discrete variable (equivalent to faceting [58]), or a real-valued axis for a continuous variable.

Visual aesthetics currently include `color`, `fill`, and `alpha`. Each visual aesthetic changes the appearance of visual elements based on the given simple or probabilistic variable. For example in Figure 2, the number of cylinders is mapped onto fill color. We can also map a probabilistic variable onto a visual aesthetic, provided that this probabilistic variable is the base case (the bottom one) in the probability chain. This restriction exists because unlike `width` or `height`, visual aesthetics are not recursive and cannot be partitioned: there is no color palette that conveys marginalization, i.e., $\sum_n P(A,B_n) = P(A)$.

We describe the format of PGoG aesthetics mapping as an EBNF grammar, listed below. However, there are additional checks to ensure that the mapping from data variables to aesthetics is valid. As an example, if the final probability function is a conditional probability such as $P(A|B,C)$, the conditionals B and C are expected to be mapped onto either coordinate or visual aesthetics. These checks are meant for fully specifying each data and visual element and eliminating ambiguity.

```
# ========== data ==========
simple_var = discrete_var|continuous_var;
# example variable names in dataset
discrete_var = "A".."Z";
continuous_var = "x"; # at most one
prob_var = marg cond;
marg = coord_var | visual_var;
cond = {coord_var} {visual_var};
coord_var = simple_var;
visual_var = discrete_var;
# ========== aesthetics ==========
coord_aes = "x" | "y";
visual_aes = "fill" | "color" | "alpha";
prob_aes = "height" | "width";
# ========== mappings ==========
coord_mappings = {coord_aes coord_var};
visual_mappings = {visual_aes visual_var};
prob_mappings = prob_aes prob_var
    {prob_aes | visual_aes prob_var};
mappings = prob_mappings coord_mappings
           visual_mappings;
```

Geometries

Based on the distinction between probability and frequency formats, we designed two separate *geometries* for PGoG, `geom_bloc` for area plots and `geom_icon` for icon-based (or unit) representations.

`geom_bloc` covers product plots [56] for discrete variables and density plots for continuous variables, both using the area of a geometry to convey probability values. Though *geom_bloc* includes visualizations similar to product plots, Figure 7 shows one of their differences. PGoG's aesthetics combination produces more plot types based on $1D$ aesthetics: by using the `x` and `width` aesthetics combination, `geom_bloc` can generate a "lying down" bar chart; despite its resemblance to the spine plot (covered by product plots), it has equidistant bars determined by the `x` mapping. Figure 1.2 also shows such a bar chart.

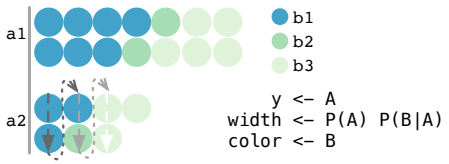


Figure 8. An icon array with a bar chart layout using `geom_icon`. The aesthetics mappings are listed below the legends. The arrows indicate that for `width` aesthetics, icons are colored first from up to down then from left to right.

A density plot variant of `geom_bloc` describes a probability function of one continuous variable and potentially other discrete variables, see Figure 3. Since the density curves are often irregular in shape, the recursive subpartition in product plots does not apply. As a result, only one level of coloring is allowed within each density shape, and thus the density/continuous variable versions of `geom_bloc` can accommodate fewer aesthetics combinations than the discrete-variable version.

For the frequency representation, we use `geom_icon`. The name is from *icon arrays*, a common visualization where each icon (or point) represents one observation in the dataset [15]. `geom_icon` differs from a scatter plot geometry (e.g. `geom_point`) in that the x,y coordinates of an icon are not directly provided by the aesthetics mapping as in `x <- A`. Rather, `geom_icon` combines information about the data variable and the aesthetics to determine how to place the icons:

- `geom_icon` determines the location of each icon with using all relevant x and y mappings and the first probability function factor. This layout rule effectively arranges icons into groups, the outline of which can resemble bars (with coordinate aesthetics) or spines (without coordinate aesthetics) in the `geom_bloc`. In the Figure 8 example, the first probability function with its associated aesthetics `width <- P(A)` and `y <- A` creates a bar chart-shaped icon array.
- Each of the remaining probability function factors creates its subgroups recursively until all probability function factors are processed.
- Within a group, if the `height` aesthetic is provided, `geom_icon` applies other aesthetics (e.g. fill color) from left to right first and then from top to bottom. If `width` is provided, other aesthetics are applied from top to bottom first and then from left to right. Figure 8 visualizes how the `width` aesthetic works within an icon bar chart. It is advisable to combine `width` and `height` with visual aesthetics to differentiate the icon subgroups; the example in Figure 8 uses `fill` to color subgroups.

PROOF-OF-CONCEPT IMPLEMENTATION IN R

We prototype PGoG in the R language as an extension to the popular `ggplot2` package [21], hosted at https://github.com/MUCollective/pgog. To make the prototype self-contained yet compatible with the rest of `ggplot2`, we wrap PGoG data, aesthetics and geometry computation within two new function calls, `geom_icon` and `geom_bloc`. The PGoG geometries compute probabilistic expressions from data and parse aesthetics before returning a `layer` object to core `ggplot2` for plot-building. All visualizations produced by the prototype

Table 1. Visualization types covered by various grammars. "w/ transformation" means that the coordinates for the icons are obtained by either computation beforehand or explicit data transformation in the code. "w/ extension" refers to the `ggmosaic` extension for `ggplot2` [26].

	Type	Icon array	Dotplot	Bar chart	Mosaic plot	Density plot
ggplot2 [21]	High-level	w/ transformation	yes	yes	w/ extension	yes
Vega-lite [47]	High-level	w/ transformation	w/ transformation	yes	yes	yes
ATOM [39]	Layout-based	yes	no	no	no	no
Product Plot [56]	Layout-based	no	no	yes	yes	no
PGoG	High-level	yes	yes	yes	yes	yes

can be replicated with existing `ggplot2` code, though significant data manipulation or lower-level library calls might be necessary. Overall, the R prototype shows that the PGoG abstraction can work alongside an unmodified, existing system that implements the Grammar of Graphics. Therefore, we anticipate few barriers to implementing PGoG in other similar declarative visualization languages, such as Vega-lite [47].

EVALUATION

We first evaluate the expressiveness of the Probabilistic Grammar of Graphics as a visualization grammar. Then, we assess the design of PGoG in terms of heuristics from the Cognitive Dimensions of Notations. We choose these expert evaluation methods because they are appropriate for the type of contribution we make (a formalism) [37] Further, since the current implementation of PGoG requires working knowledge of Grammar of Graphics and the R language, expert evaluation is more practical than user studies [12].

Expressiveness

In line with the convention in recent visualization literature [39, 44, 47], we view an *expressive* specification language as one that covers a wide range of visualizations types. The PGoG grammar can express area plots (including density plots), icon arrays, and dotplots, a meaningful coverage unique to PGoG, see Table 1. We show the breadth of the plot types covered in Figure 3. It is worth noting that Figure 3 does not cover all legal combinations on aesthetics and geometries in PGoG and leaves out all bar chart/spine plot variants. In addition, it is possible to have arbitrarily nested probability expressions for `geom_bloc` with discrete variables. Overall, the expressiveness of PGoG comes from its ability to map probability distributions directly onto aesthetics, and the flexible re-combination of coherent grammar components.

While PGoG unites many types of probabilistic visualizations with various layouts, Figure 1 shows how PGoG is still limited in expressiveness in the wild. We categorize what PGoG cannot cover into two broad categories. First, some probabilistic visualizations use special layouts, such as parliament seating or hierarchy of probabilities; even though PGoG do not cover such layouts, the PGoG concepts often still apply (*e.g.*, use the `width` aesthetics to encode probabilities). The Discussion section will suggest how a future version of PGoG might achieve some of the special layouts informed by the structure of probability expression or even statistical models. The other class of visualizations express probabilities through visual channels not in PGoG; notable examples includes temporal (animation) (Figure 1.5) and geospatial channels (Figure 1.6). In the future, we plan to cover these and other additional channels to express probabilities informed by uncertainty visualization research.

Cognitive dimensions of notations

We designed the PGoG grammar to bring probabilistic visualization specification closer to the existing language of probability expressions, and hopefully therefore closer to the mental model of users familiar with that notation. To formalize this design intuition, we judge the PGoG grammar against the Cognitive Dimensions of Notations, which "describes the usability of notational systems", including programming languages [6].

Below, we highlight several relevant dimensions from Blackwell *et al.* [6]. Dimensions not included below either evaluate similarly to the ones included, or they don't distinguish PGoG from the original Grammar of Graphics.

- *Viscosity: Resistance to Change*. A system should not be *viscous*: the user should not have to unnecessarily take multiple actions to make a change. PGoG avoids viscosity by allowing the user to change (probabilistic) variables, aesthetics, and geometries independently of each other, so there is a shorter *edit distance* to switch between visualizations and the syntax remains *consistent*, see Figure 9.
- *Visibility: Ability to View Components Easily*. PGoG makes probability distributions first-class citizens, helping the user directly express and visualize the probability expressions they want. In other languages, the user needs to translate the probability distributions they want into the corresponding coordinate and visual aesthetics mappings when writing specifications, as shown in Figure 2.d.
- *Premature Commitment: constraints on the Order of Doing Things*. As we note in the Grammar specification Section, PGoG syntax allows the user to compose arbitrary probabilistic variables from column variables. From this angle, PGoG avoids premature commitment in that the user can keep the data *tidy* without having to calculate various proportions in the data before visualizing them.
- *Closeness of Mapping: Closeness of Representation to Domain*. We motivate the design of PGoG with the need for "a closer integration between visualization and statistical algorithms" [24]. As a result, our notations for probability distributions are the same as in statistics. If the user knows the distribution they want to specify, they can do so directly, as in probabilistic programming. Thus, the user may focus more on choosing the probability distribution, or deciding whether the frequency format will be more effective. The PGoG grammar shields user from of configuring a recursive layout or naming the plot they want (*e.g.* violin plot).

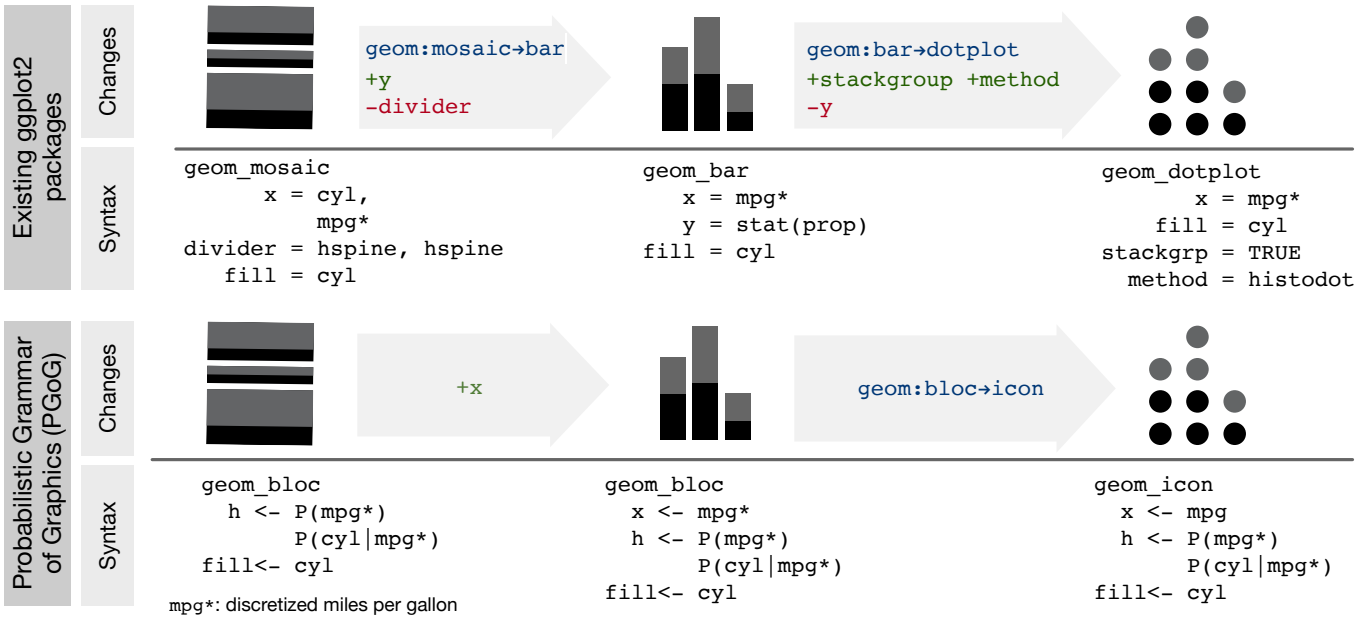


Figure 9. PGoG specification shortens edit distances in the space of probabilistic visualizations. This figure reads from left to right: a spine plot, bar chart, and dotplot with the same underlying distribution $P(cyl, mpg)$. Compared to existing implementations, the user needs far fewer changes to switch from one probabilistic visualization to another.

- *Hidden dependencies: Important Links between Entities Are not Visible*. There are two main hidden dependencies in the PGoG implementation. First, whether the user gets density plots or product plots depends on data: if the marginal of a probabilistic variable is continuous, PGoG will construct a density plot. We choose a more uniform interface (only one geometry for all area plots) over two separate geometries named "density" and "product". Second, if the user modifies the conditionals in the overall probability function, other aesthetics likely need modifying, too. This is because all conditionals in a probabilistic variable need to be *grounded*, or mapped to, a coordinate or visual aesthetic, as determined by the grammar rules.
- *Error-proneness: The Notation Invites Mistakes and the System Gives Little Protection; Hard Mental Operations: High Demand on Cognitive Resources*. PGoG does require the user to factor a desired probability function, which can be cognitively demanding for some users. This is necessary for an unambiguous specification but can result in errors. The user might write factors such as $P(A)P(B)$ instead of $P(A)P(B|A)$, thinking that they want the probability for event A and B. The current system cannot educate users on probability fundamentals; however as mitigation, the implementation checks and throws an error when the probability factors do not multiply to a valid probability function (this ensures, at least, that if a chart *is* generated, it is valid). In addition, PGoG infers layout from the probability structure once they are specified correctly, so it prevents some potential errors and alleviates some cognitive demand for reasoning about (recursive) layouts.

Overall, PGoG evaluates favorably against the cognitive dimensions of notations: its elegant syntax allows it to stay close to the notation of probability distributions. We do identify potential hidden dependency and error-proneness problems, issues that future user studies might investigate.

Validation through a visualization algebra

To show that PGoG can avoid making some misleading visualizations, we apply the algebraic process for visualization design model [31]. Among other things, this model defines a symmetry ("invertible transformations") in the data space ($\alpha : D \rightarrow D$) and a symmetry in the visualization space ($\omega : V \rightarrow V$). These symmetries lead to the Principle of Unambiguous Data Depiction for a visualization design: a substantial change in data (α) should result in a substantial change in the resulting visualization (ω). In our motivational example in Figure 2, the naive `ggplot2` version always calculates the density plot regions to have equal area. This leads to a violation of the Principle of Unambiguous Data Depiction: if we remove half of the data points for 8-cylinder cars, the visible proportion of 8-cylinder cars does not change in the ggplot chart; in the PGoG chart, it does (see Figure 10).

DISCUSSION AND FUTURE WORK

The Evaluation section demonstrates that PGoG covers a wide range of probabilistic visualizations with a descriptive syntax close to statistics, while facilitating design exploration through combinations of simple aesthetics. In addition to its current power and benefits, PGoG has potential to improve how we communicate a wider range of uncertainty data.

Defining visualizations with uncertainty concepts

Though the current PGoG can express one probability distribution per visualization, we wish to extend the coverage of PGoG to uncertainty data in general. We are interested in uncertainty

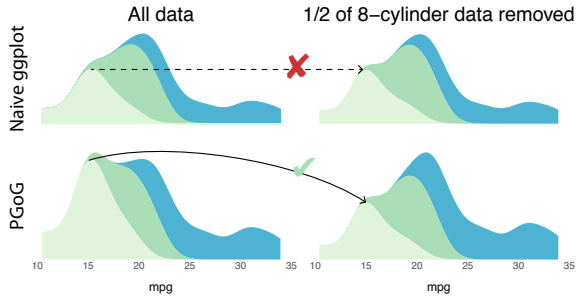


Figure 10. Evaluating naive `ggplot2` and PGoG against the Unambiguous Data Depiction principle in the algebraic process for visualization design [31]. For a naive `ggplot2` spec (Figure 2.b), removing data (transformation $\alpha \neq 1_D$) is hardly reflected in the visualization (transformation $\omega = 1_V$); in PGoG, removing the same data shrank the shaded region proportionately, indicated by the arrowhead.

because it is important for decision-making but also difficult to communicate; in particular, one of the barriers of visualizing uncertainty is a lack of effective visualization techniques and tools [14]. The motivation behind PGoG should be transferable to uncertainty visualizations: a Grammar of Graphics for *uncertainty* data may help users externalize uncertainty into visualizations.

There exist many taxonomies for uncertainty data [52, 40, 48]. In addition to the single probability distribution PGoG handles, the taxonomies cover more complex data structures and semantics, such as the epistemic *v.s.* aleatory distinction [48] and errors [52]. Below, we show how one category, uncertainty lineage [52], can translate to effective visualizations with a future PGoG extension.

Leveraging uncertainty lineage and model structures

Uncertainty data can be hierarchical or sequential ("lineage" in Thomson *et al.*) [52]. For example, a Bayesian mixed linear model can produce parameter estimates that can be organized in a hierarchy (see Figure 16.3 in *Doing Bayesian data analysis: A tutorial with R, JAGS, and Stan* [33]); in machine learning, an event sequence prediction model can generate probabilities of state transitions [20]. These structures in data (or even in statistical models) can be captured with formal specifications, such as probabilistic programs.

Then, a future version of PGoG can use the structures in data (or models) to inform and expedite visualization specifications. In addition to maintaining a close mapping between user mental models (uncertainty concepts and models), this capability can hide the messiness of common statistical model outputs. In the case of Stan, the model output is a combination of scalar and vector parameters of different data types drawn from thousands of iterations[3]. Since these messy parameters are defined in the model already, given reasonable defaults, PGoG might be able to infer plots from model specifications. For instance, a hierarchical model $response \sim (1|condition)$ might lead to two plots: one for condition v.s. posterior predictions, and another for condition v.s. posterior distribution of the means.

[3]Example of inspecting Stan outputs: https://cran.r-project.org/web/packages/rstan/vignettes/stanfit-objects.html

With sensible defaults, a future version of PGoG can map uncertainty data and model structures onto more visualization layouts. These can include as Sankey diagrams and elements such as links [44], but they are not yet systematically described with a formalization such as the Grammar of Graphics. One example of such mapping can be from the summation in the law of total probability $P(A) = \sum_B P(A \cap B)$ to a forking layout, such as the one in a New York Times article in Figure 1.3 [4].

Enabling uncertainty visualization best practices

There are effective uncertainty visualization techniques we have not included in PGoG. Wilkinson discusses uncertainty intervals and several aesthetics to convey uncertainty ("error"), such as transparency and blur [58]. Recently, Correll *et al.* proposed a color palette that maps both probability (uncertainty) and data values [10]. Animation can reflect uncertainty through sampling, as in the New York Times election needle [51]. A future version of PGoG can have new aesthetics and geometries to represent these sorts of visualizations; e.g., aesthetics for temporal frequency could be used to create animated probabilistic visualizations, such as HOPs [25].

Informing uncertainty visualization research

Beyond the potential for PGoG to facilitate specification of uncertainty visualizations, it can provide a theoretical framework to more systematically study people's understanding of different uncertainty visualization (and probabilistic visualization) types. For example, by giving a theoretical definition of an equivalent frequency format visualization for any given probability format visualization (by moving from `geom_bloc` to `geom_icon` while keeping aesthetics constant), we could more comprehensively study the effect of frequency formats across a range of visualization types. Or, to better understand how people interpret different depictions of the same conditional probability distributions, we could study a set of visualizations that all use the same probability function expression. This would allow us to better understand how well people understand conditional probabilities from different visual depictions of the same distribution.

CONCLUSION

In this paper, we introduce a new abstraction for visualizing probability data: the Probabilistic Grammar of Graphics (PGoG), a set of new Grammar of Graphics components and specification rules. We instantiate this visualization grammar in R. PGoG treats probabilistic variables as first class citizens in visualization specifications. This design guarantees the correctness of intended probabilities, stays close to users' mental models, and facilitates exploration of probabilistic visualization designs. By instantiating probability concepts within visualization, PGoG has the potential to further uncertainty visualization research through allowing systematic study of probability distributions and their representations.

ACKNOWLEDGEMENTS

This work is supported by the National Science Foundation, Award Number 1910431. Many thanks to Puhe Liang for helping build the visualization collection, as well as Dominik Moritz and Eytan Adar for their valuable feedback.

REFERENCES

[1] 2020. PAIR-Code/Facets. PAIR code. (Jan. 2020). https://github.com/PAIR-code/facets.

[2] Corinne Abrams. 2019. 'You Have to Actually Cut Open Mumbai's Belly'—Inside One of the World's Most Audacious Transit Projects. *Wall Street Journal* (Jan. 2019). https://www.wsj.com/articles/through-monsoons-around-slums-under-templesmumbai-builds-its-first-subway-11546803877.

[3] Sarah Almukhtar, Mike Andre, Wilson Andrews, Matthew Bloch, and et al. 2018. Live Forecast: Who Will Win the House? *The New York Times* (Nov. 2018). https://www.nytimes.com/interactive/2018/11/06/us/elections/results-house-forecast.html,https://www.nytimes.com/interactive/2018/11/06/us/elections/results-house-forecast.html.

[4] Emily Badger, Claire Ann Miller, Adam Pearce, and Kevin Quealy. 2018. Income Mobility Charts for Girls, Asian-Americans and Other Groups. Or Make Your Own. *New York Times* (March 2018). https://www.nytimes.com/interactive/2018/03/27/upshot/make-your-own-mobility-animation.html

[5] Karin Binder, Stefan Krauss, and Georg Bruckmaier. 2015. Effects of visualizing statistical information – an empirical study on tree diagrams and 2 × 2 tables. *Frontiers in Psychology* 6 (2015). DOI: http://dx.doi.org/10.3389/fpsyg.2015.01186

[6] A. F. Blackwell, C. Britton, A. Cox, T. R. G. Green, C. Gurr, G. Kadoda, M. S. Kutar, M. Loomes, C. L. Nehaniv, M. Petre, C. Roast, C. Roe, A. Wong, and R. M. Young. 2001. Cognitive Dimensions of Notations: Design Tools for Cognitive Technology. In *Cognitive Technology: Instruments of Mind*, Meurig Beynon, Chrystopher L. Nehaniv, and Kerstin Dautenhahn (Eds.). Springer Berlin Heidelberg, 325–341.

[7] Aaron Bycoffe and Rachael Dottle. 2019. The 2020 Endorsement Primary. https://projects.fivethirtyeight.com/2020-endorsements/democratic-primary/. (Feb. 2019).

[8] Sean Carmody. 2010. Risk Characteriztion Theatres (RCT). https://github.com/seancarmody/stubborn-mule. (2010).

[9] Bob Carpenter, Andrew Gelman, Matthew D. Hoffman, Daniel Lee, Ben Goodrich, Michael Betancourt, Marcus Brubaker, Jiqiang Guo, Peter Li, and Allen Riddell. 2017. Stan: A probabilistic programming language. *Journal of statistical software* 76, 1 (2017).

[10] Michael Correll, Dominik Moritz, and Jeffrey Heer. 2018. Value-Suppressing Uncertainty Palettes. *Conference on Human Factors in Computing Systems - CHI '18* (2018). DOI: http://dx.doi.org/10.1145/3173574.3174216 ISBN: 9781450356206.

[11] D. R. Cox. 1978. Some remarks on the role in statistics of graphical methods. *Journal of the Royal Statistical Society: Series C (Applied Statistics)* 27, 1 (1978), 4–9.

[12] Alan Dix (Ed.). 2004. *Human-computer interaction* (3rd ed ed.). Pearson/Prentice-Hall, Harlow, England ; New York.

[13] Michael Fernandes, Logan Walls, Sean Munson, Jessica Hullman, and Matthew Kay. 2018. Uncertainty Displays Using Quantile Dotplots or CDFs Improve Transit Decision-Making. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems (CHI '18)*. ACM, New York, NY, USA, 144:1–144:12. DOI:http://dx.doi.org/10.1145/3173574.3173718 event-place: Montreal QC, Canada.

[14] Baruch Fischhoff, Noel Brewer, and Julie Downs. 2012. *Communicating Risks and Benefits: An Evidence-Based User's Guide*. Technical Report. 242 pages.

[15] Mirta Galesic, Rocio Garcia-Retamero, and Gerd Gigerenzer. 2009. Using Icon Arrays to Communicate Medical Risks: Overcoming Low Numeracy. *Health Psychology* 28, 2 (2009), 210–216. DOI: http://dx.doi.org/10.1037/a0014474 ISBN: 0278-6133.

[16] A.W. Geiger. 2018. *18 striking findings from 2018*. Technical Report. https://www.pewresearch.org/fact-tank/2018/12/13/18-striking-findings-from-2018/

[17] Gerd Gigerenzer and Ulrich Hoffrage. 1995. How to improve Bayesian reasoning without instruction: Frequency formats. *Psychological Review* 102, 4 (1995), 684–704. DOI: http://dx.doi.org/10.1037/0033-295X.102.4.684 ISBN: 0033-295X\r1939-1471.

[18] Andrew D. Gordon, Thore Graepel, Nicolas Rolland, Claudio Russo, Johannes Borgstrom, and John Guiver. 2014a. Tabular: A Schema-driven Probabilistic Programming Language. In *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '14)*. ACM, New York, NY, USA, 321–334. DOI: http://dx.doi.org/10.1145/2535838.2535850 event-place: San Diego, California, USA.

[19] Andrew D. Gordon, Thomas A. Henzinger, Aditya V. Nori, and Sriram K. Rajamani. 2014b. Probabilistic Programming. In *Proceedings of the on Future of Software Engineering (FOSE 2014)*. ACM, New York, NY, USA, 167–181. DOI: http://dx.doi.org/10.1145/2593882.2593900 event-place: Hyderabad, India.

[20] Shunan Guo, Fan Du, Sana Malik, Eunyee Koh, Sungchul Kim, Zhicheng Liu, Donghyun Kim, Hongyuan Zha, and Nan Cao. 2019. Visualizing Uncertainty and Alternatives in Event Sequence Predictions. (2019), 12.

[21] Hadley Wickham. 2016. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York. https://ggplot2.tidyverse.org

[22] Sarah T. Hawley, Brian Zikmund-Fisher, Peter Ubel, Aleksandra Jancovic, Todd Lucas, and Angela Fagerlin. 2008. The impact of the format of graphical presentation on health-related knowledge and treatment choices. *Patient Education and Counseling* 73, 3 (2008), 448–455. DOI: http://dx.doi.org/10.1016/j.pec.2008.07.023 ISBN: 0738-3991 (Print)\r0738-3991 (Linking).

[23] Jeffrey Heer, Stuart K. Card, and James A. Landay. 2005. Prefuse: A Toolkit for Interactive Information Visualization. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '05)*. ACM, New York, NY, USA, 421–430. DOI: http://dx.doi.org/10.1145/1054972.1055031 event-place: Portland, Oregon, USA.

[24] Jeffrey Heer and Ben Shneiderman. 2012. Interactive Dynamics for Visual Analysis. *Queue* 10, 2 (Feb. 2012), 30:30–30:55. DOI: http://dx.doi.org/10.1145/2133416.2146416

[25] Jessica Hullman, Paul Resnick, and Eytan Adar. 2015. Hypothetical Outcome Plots Outperform Error Bars and Violin Plots for Inferences about Reliability of Variable Ordering. *PLOS ONE* 10, 11 (Nov. 2015), e0142444. DOI:http://dx.doi.org/10.1371/journal.pone.0142444

[26] Haley Jeppson, Heike Hofmann, and Di Cook. 2019. *ggmosaic: Mosaic Plots in the 'ggplot2' Framework*. http://github.com/haleyjeppson/ggmosaic R package version 0.2.1.

[27] Brian Johnson. 1992. TreeViz: treemap visualization of hierarchically structured information. In *CHI*, Vol. 92. 369–370.

[28] Matthew Kay, Tara Kola, Jessica R Hullman, and Sean A Munson. 2016. When (ish) is My Bus?: User-centered Visualizations of Uncertainty in Everyday, Mobile Predictive Systems. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems - CHI '16*. 5092–5103. DOI: http://dx.doi.org/10.1145/2858036.2858558

[29] Matthew Kay, Dan Morris, Mc Schraefel, and Julie A Kientz. 2013. There's No Such Thing as Gaining a Pound: Reconsidering the Bathroom Scale User Interface. *Proceedings of the 2013 ACM international joint conference on Pervasive and ubiquitous computing - UbiComp '13* (2013), 401–410. DOI: http://dx.doi.org/10.1145/2493432.2493456 ISBN: 9781450317702.

[30] Younghoon Kim, Kanit Wongsuphasawat, Jessica Hullman, and Jeffrey Heer. 2017. 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)*. ACM, New York, NY, USA, 2628–2638. DOI:http://dx.doi.org/10.1145/3025453.3025866 event-place: Denver, Colorado, USA.

[31] G. Kindlmann and C. Scheidegger. 2014. An Algebraic Process for Visualization Design. *IEEE Transactions on Visualization and Computer Graphics* 20, 12 (Dec. 2014), 2181–2190. DOI: http://dx.doi.org/10.1109/TVCG.2014.2346325

[32] Niko Kommenda, Caelainn Barr, Josh Holder, Niko Kommenda, Caelainn Barr, and Josh Holder. 2018. Gender Pay Gap: What We Learned and How to Fix It. *The Guardian* (April 2018). https://www.theguardian.com/news/ng-interactive/2018/apr/05/women-are-paid-less-than-men-heres-how-to-fix-it.

[33] John Kruschke. 2014. *Doing Bayesian data analysis: A tutorial with R, JAGS, and Stan*. Academic Press.

[34] L. Liu, L. Padilla, S. H. Creem-Regehr, and D. H. House. 2019. Visualizing Uncertain Tropical Cyclone Predictions using Representative Samples from Ensembles of Forecast Tracks. *IEEE Transactions on Visualization and Computer Graphics* 25, 1 (Jan. 2019), 882–891. DOI: http://dx.doi.org/10.1109/TVCG.2018.2865193

[35] Jock Mackinlay. 1986. Automating the design of graphical presentations of relational information. *ACM Transactions on Graphics (TOG)* 5, 2 (April 1986), 110–141. DOI:http://dx.doi.org/10.1145/22949.22950

[36] Michelle McDowell, Gerd Gigerenzer, Odette Wegwarth, and Felix G. Rebitschek. 2019. Effect of Tabular and Icon Fact Box Formats on Comprehension of Benefits and Harms of Prostate Cancer Screening: A Randomized Trial. *Medical Decision Making* 39, 1 (Jan. 2019), 41–56. DOI: http://dx.doi.org/10.1177/0272989X18818166

[37] Tamara Munzner. 2008. Process and Pitfalls in Writing Information Visualization Research Papers. In *Information Visualization*. Vol. 4950. Springer Berlin Heidelberg, Berlin, Heidelberg, 134–153. DOI: http://dx.doi.org/10.1007/978-3-540-70956-5_6

[38] Jurriaan P. Oudhoff and Daniëlle R. M. Timmermans. 2015. The Effect of Different Graphical and Numerical Likelihood Formats on Perception of Likelihood and Choice. *Medical Decision Making* 35, 4 (May 2015), 487–500. DOI: http://dx.doi.org/10.1177/0272989X15576487

[39] Deokgun Park, Steven Mark Drucker, Roland Fernandez, and Niklas Elmqvist. 2017. ATOM: A Grammar for Unit Visualizations. *IEEE Transactions on Visualization and Computer Graphics* 2626, c (2017). DOI:http://dx.doi.org/10.1109/TVCG.2017.2785807

[40] M.Elisabeth Paté-Cornell. 1996. Uncertainties in risk analysis: Six levels of treatment. *Reliability Engineering & System Safety* 54, 2 (1996), 95–111. DOI: http://dx.doi.org/10.1016/S0951-8320(96)00067-1 ISBN: 0951-8320.

[41] Martyn Plummer. 2003. JAGS: A program for analysis of Bayesian graphical models using Gibbs sampling. *Working Papers* (2003), 8.

[42] Nadja Popovich. 2019. America's Light Bulb Revolution. *The New York Times* (March 2019). https://www.nytimes.com/interactive/2019/03/08/climate/light-bulb-efficiency.html,https://www.nytimes.com/interactive/2019/03/08/climate/light-bulb-efficiency.html.

[43] R Core Team. 2019. *R: A language and environment for statistical computing*. Vienna, Austria. https://www.R-project.org/ R Foundation for Statistical Computing.

[44] D. Ren, B. Lee, and M. Brehmer. 2019. Charticulator: Interactive Construction of Bespoke Chart Layouts. *IEEE Transactions on Visualization and Computer Graphics* 25, 1 (Jan. 2019), 789–799. DOI: http://dx.doi.org/10.1109/TVCG.2018.2865158

[45] Valentina Romei, James Politi, Cale Tilford, and Billy Ehrenberg-Shannon. 2018. Italian Election Poll-Tracker 2018: Who Is Running and Why It Matters. *Financial Times* (Jan. 2018). https://ig.ft.com/italy-poll-tracker/.

[46] Alastair Rushworth. 2019. Inspectdf: Inspection, Comparison and Visualisation of Data Frames. https://CRAN.R-project.org/package=inspectdf. (2019).

[47] Arvind Satyanarayan, Dominik Moritz, Kanit Wongsuphasawat, and Jeffrey Heer. 2017. Vega-Lite: A Grammar of Interactive Graphics. *IEEE Transactions on Visualization and Computer Graphics* 23, 1 (2017), 341–350. DOI: http://dx.doi.org/10.1109/TVCG.2016.2599030

[48] David Spiegelhalter. 2017. Risk and Uncertainty Communication. *Annual Review of Statistics and Its Application* 4, 1 (March 2017), 31–60. DOI:http://dx.doi.org/10.1146/annurev-statistics-010814-020148

[49] David J. Spiegelhalter, Andrew Thomas, Nicky G. Best, Wally Gilks, and D. Lunn. 1996. BUGS: Bayesian inference using Gibbs sampling. *Version 0.5,(version ii) http://www. mrc-bsu. cam. ac. uk/bugs* 19 (1996).

[50] 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 (Jan. 2002), 52–65. DOI: http://dx.doi.org/10.1109/2945.981851

[51] The Upshot Staff. 2018. What Is the Needle? *The New York Times* (Nov. 2018). https://www.nytimes.com/2018/11/05/upshot/needle-election-night-2018-midterms.html

[52] Judi Thomson, Elizabeth Hetzler, Alan MacEachren, Mark Gahegan, and Misha Pavel. 2005. A typology for visualizing uncertainty. In *Visualization and Data Analysis 2005*, Vol. 5669. International Society for Optics and Photonics, 146–157. DOI: http://dx.doi.org/10.1117/12.587254

[53] Martin Wattenberg, Fernanda Viégas, and Moritz Hardt. 2016. Attack Discrimination with Smarter Machine Learning. https://research.google.com/bigpicture/attacking-discrimination-in-ml/. (2016). Retrieved on 1/5/2020.

[54] Hadley Wickham. 2010. A Layered grammar of graphics. *Journal of Computational and Graphical Statistics* 19, 1 (2010), 3–28. DOI: http://dx.doi.org/10.1198/jcgs.2009.07098 ISBN: 1061-8600.

[55] Hadley Wickham. 2014. Tidy Data. *Journal of Statistical Software* 59, 1 (Sept. 2014), 1–23. DOI: http://dx.doi.org/10.18637/jss.v059.i10

[56] Hadley Wickham and Heike Hofmann. 2011. Product Plots. *IEEE Transactions on Visualization and Computer Graphics* 17, 12 (Dec. 2011), 2223–2230. DOI:http://dx.doi.org/10.1109/TVCG.2011.227

[57] Claus O. Wilke. 2018. *ggridges: Ridgeline Plots in 'ggplot2'*. https://CRAN.R-project.org/package=ggridges R package version 0.5.1.

[58] Leland Wilkinson. 2005. *The Grammar of Graphics*. Springer-Verlag, New York. http://link.springer.com/10.1007/0-387-28695-0

[59] Jingxiong Zhang, Michael F. Goodchild, and Michael F. Goodchild. 2002. *Uncertainty in Geographical Information*. CRC Press. DOI: http://dx.doi.org/10.1201/b12624