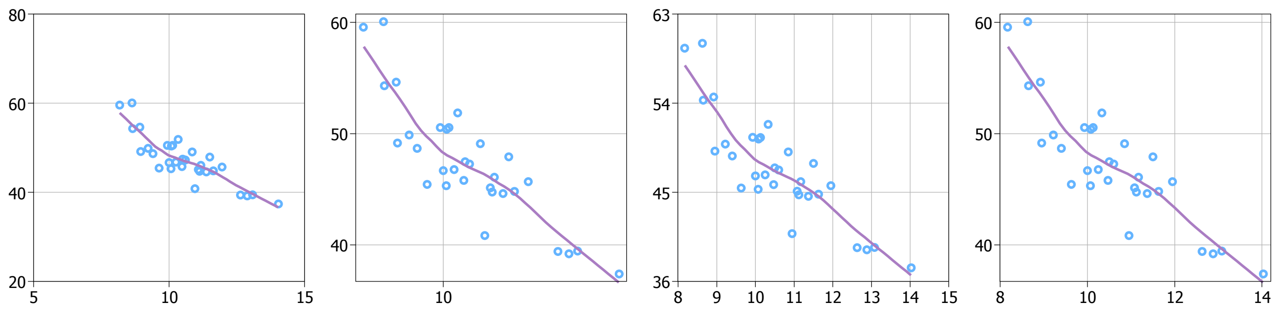
# An Extension of Wilkinson's Algorithm for Positioning Tick Labels on Axes

Justin Talbot, Sharon Lin, Pat Hanrahan



(a) Heckbert (b) R's pretty (c) Wilkinson (d) Extended

Fig. 1: Comparison of axis labeling algorithms. Our extension of Wilkinson's optimization approach produces nice labelings while maintaining good visual density of the labels and coverage of the data.

**Abstract**—The non-data components of a visualization, such as axes and legends, can often be just as important as the data itself. They provide contextual information essential to interpreting the data. In this paper, we describe an automated system for choosing positions and labels for axis tick marks. Our system extends Wilkinson's optimization-based labeling approach [11] to create a more robust, full-featured axis labeler. We define an expanded space of axis labelings by automatically generating additional nice numbers as needed and by permitting the extreme labels to occur inside the data range. These changes provide flexibility in problematic cases, without degrading quality elsewhere. We also propose an additional optimization criterion, *legibility*, which allows us to simultaneously optimize over label formatting, font size, and orientation. To solve this revised optimization problem, we describe the optimization function and an efficient search algorithm. Finally, we compare our method to previous work using both quantitative and qualitative metrics. This paper is a good example of how ideas from automated graphic design can be applied to information visualization.

**Index Terms**—Axis labeling, nice numbers.

## 1 Introduction

Axis tick labels are an important component of most basic plot types. By providing cognitive context for the data, they support analysis tasks, such as value lookup, range computation, and slope estimation. Good axis labelings should make the mental computation involved in these tasks easier. Additionally, tick labels contribute substantially to the visual appearance of the plot and therefore graphic design properties such as the density of labels or the whitespace they introduce around a plot require careful consideration. We tackle the problem of automatically producing a sequence of labels for an axis.

Many automatic labeling schemes are inspired by Heckbert's labeling algorithm published in *Graphics Gems 1* [3]. These approaches use a simple formula to compute a nice step size between labels, taking the range of the data and the desired number of ticks as input. One problem in Heckbert's approach is that for small numbers of labels the range of the labels can be much larger than the range of the data (Figure 1a). In R [5], this problem is avoided by dropping labels which overlap or fall outside the data range. However, this can result in axes with unevenly spaced labels or axes with only one label (Figure 1b), making it impossible to understand the extent of the data.

More recently, Wilkinson [11] proposed a labeling approach based on an optimization search over a space of possible labelings. By imposing an optimization constraint during the search, it can find labelings with less problematic whitespace. One shortcoming is that the whitespace constraint may force the algorithm to choose labelings that are overly dense, such that the axis distracts from the data (Figure 1c).

We extend Wilkinson's optimization framework to create a more robust axis labeler that can produce labelings with both good coverage and good label density (Figure 1d). This paper makes four contributions:

1. We describe an expanded space of nice labelings based on algorithmically generating a ranked set of tick intervals and phase offsets. We also recommend using flexible labelings where the extreme labels may occur inside or outside of the data range to increase the robustness of the algorithm.

2. We add three label appearance dimensions to the optimization space which allow us to also optimize over label format, font size, and orientation. This permits better labelings in cases where the labels are long compared to the size of the axis.

3. We modify all three components of Wilkinson's optimization formula, *simplicity*, *coverage*, and *granularity* to make them applicable to our new space, and we add a new optimization component, *legibility*, to evaluate the appearance dimensions.

4. We describe a search algorithm for finding solutions to this optimization problem.

We start by describing previous work in this area, including a brief explanation of Wilkinson's algorithm to provide necessary background for our extensions. Next, we explain our expanded labeling space, the associated optimization criteria, and the search function we use to efficiently find solutions. Finally, we demonstrate the effectiveness of

• *Justin Talbot is with Stanford University, E-mail: jtalbot@stanford.edu.*
• *Sharon Lin is with Stanford University, E-mail: sharonl@cs.stanford.edu.*
• *Pat Hanrahan is with Stanford University, E-mail: hanrahan@cs.stanford.edu.*

our extended optimization approach using both quantitative and qualitative metrics.

## 2 Previous Work

Given the importance of axis labeling in all forms of information visualization, it is somewhat surprising that there has been little work in this area.

There is substantial evidence that nice numbers are much easier to work with in a variety of mental estimation tasks [6]. However, we are not aware of any experimental research to determine why certain numbers are nice. Nor are we aware of any research looking specifically at the benefit of nice numbers on plot axes.

A few authors have published practical advice on labeling axes. Bill Cleveland gives a number of suggestions for effective visual layout of axes. In particular he recommends, "...choose the scales so that the data rectangle fills up as much of the scale-line rectangle as possible" [1]. This recommendation drives our desire to find axis labelings that cover the data range, but do not go significantly beyond it. He also suggests that one "not force tickmarks and their labels to occur at the frame corners" [2], which influences our choice to allow flexible labeling. In *The Visual Display of Quantitative Information* [10], Edward Tufte suggests that labels should be used to convey semantic information about the data set, an idea he terms *data-based labels*.

A sequence of papers from the late 60's through early 80's, by Thayer and Storer [9], Sparks [7], Nelder [4], and Stirling [8], establishes the basis for algorithmic axis labeling. They describe a variety of formulas that directly produce good labelings with a given number of ticks by selecting consecutive integer multiples of an element from a set of nice numbers, $Q$. For example, Nelder proposes using $Q = \{1.2, 1.6, 2, 2.5, 3, 4, 5, 6, 8, 10\}$. (Where larger or smaller numbers are needed as labels, $Q$ can be multiplied by an appropriate power of ten.)

Heckbert's 1990 algorithm [3] differs from this previous work by potentially returning more or less than the requested number of ticks; this extra flexibility allows him to use a smaller set of nice numbers, $Q = \{1, 2, 5\}$. Additionally, Heckbert's algorithm only finds "loose" labelings, where the extreme tick locations are outside the range of the data. This ensures that data values can be looked up by interpolating between labels. A widely-used variant of Heckbert's algorithm is the *pretty* function in the R statistical environment [5] (source code at https://svn.r-project.org/R/trunk/src/appl/pretty.c). R's pretty allows the user to favor different components in $Q$ by supplying weights. This can allow the user to encourage steps of size 5 over steps of size 2, if she perceives those to be "nicer".

In contrast to these simple formula-based approaches, Wilkinson has proposed using an optimization search over a space of possible loose labelings (described in *The Grammar of Graphics* [11] and in a recent Java implementation [12]). These labelings are derived from a preference-ordered list of nice numbers—$Q = [1, 5, 2, 2.5, 3]$ in the book and $Q = [1, 5, 2, 2.5, 3, 4, 1.5, 7, 6, 8, 9]$ in the implementation.

Wilkinson's optimization function is defined as the sum of three components. If the user requests $m$ labels and a possible labeling has $k$ labels, then the components are:

$$simplicity = 1 - i/|Q| + v/|Q|$$

$$coverage = \frac{d_{max} - d_{min}}{l_{max} - l_{min}}$$

$$granularity = 1 - |k - m|/m^1$$

(Other variables are defined in Table 2.) *Simplicity* encourages nicer labeling sequences by preferring step sizes that appear earlier in $Q$. It also rewards labelings that include 0 as a way to ground the sequence. The *coverage* component encourages labelings that do not extend far beyond the range of the data, penalizing the bad whitespace cases that arise in Heckbert's approach. In addition to the cost function, Wilkinson employs a hard minimum on the coverage which he recommends setting to ensure no more than 20% whitespace. Finally, *granularity* encourages returning roughly the number of labels that were requested.

In his implementation, Wilkinson also rewards labelings that cover nice ranges, such as 0–1 or 0–100. We have not yet experimented with including this component in our extension, but believe it could be useful.

## 3 Expanded Labeling Space

We have found that the space of possible labelings considered by both Wilkinson and Heckbert is overly constrained. Our extension begins by expanding the space of possible labelings in two ways—by permitting flexible labelings and by algorithmically expanding the list of nice numbers.

Then we add dimensions to the space that capture variations in the label appearance. Incorporating these dimensions into the optimization framework permits us to ensure that the labels remain legible even in difficult labeling scenarios, in particular, when the label length is large compared to the available space.

### 3.1 Flexible labeling

As suggested by Cleveland, instead of limiting ourselves to loose labelings, where the extreme labels are outside the data range, we consider placement of the extreme labels both inside and outside of the range of the data. We do require that the labeling "fill" the data range. That is, it must not be possible to extend the labeling sequence in either direction and produce an additional label that is also within the data range.

This flexible placement is not just an aesthetic preference; as we will show in the evaluation, it provides additional labeling options that can greatly increase the quality of the optimization results while meeting label density and coverage constraints. However, we must be careful that extreme labels inside the data range do not get too far from the limits of the data since extrapolation is inherently more error prone than interpolation.

### 3.2 Algorithmic generation of nice numbers

Rather than using only the step sizes in the finite-size list $Q$, we use $Q$ to generate an infinite preference-ordered list of possible step sizes and phase offset amounts. Conceptually, our proposed infinite list is generated by "skipping" elements in a base labeling sequence, taking only every $j$th label. For example, starting from this nice sequence which consists of sequential multiples of $q = 2 \in Q$,

$$\ldots, 0, 2, 4, 6, 8, 10, 12, \ldots$$

we can take every other label ($j = 2$), which results in the sequence

$$\ldots, 0, 4, 8, 12, 16, \ldots$$

with a new step size of 4, the product of $j$ and $q$. Table 1 shows the first 18 entries of the infinite sequence of step sizes generated by repeatedly iterating over the generating list, $Q$, while increasing $j$. We use $Q = [1, 5, 2, 2.5, 4, 3]$ which, we assert, generates a reasonable preference ordering for step sizes. The bottom row shows the formula used to generate the step size. Like previous work, we implicitly include all power of ten multiples of the step sizes in the set of nice numbers, so, for comparison purposes, we keep the step sizes in the range 1–10 by dividing out the nearest smaller power of ten.

When $j > 1$, we have a choice in which numbers to skip. Continuing the previous example, where $q = 2$ and $j = 2$, the sequence

$$\ldots, 2, 6, 10, 14, 18, \ldots$$

is also a valid alternative labeling produced by starting the skipping on a different element. The result is to offset, or phase, the entire sequence. The final column of Table 1 shows the possible offsets for each element in the step size sequence. When step sizes are repeated in the sequence, they are accompanied by an increasing number of offset possibilities. We consider the *first* occurrence of a step size-offset pair to define its preference ordering. Again, the bottom row shows the formula used to generate the offsets.

[1] Unlike the definition in *The Grammar of Graphics*, Wilkinson's implementation does not clamp *granularity* to non-negative numbers.

Table 1: Algorithmically Generated Nice Step Sizes and Offsets

| $j$ | $q$ | step size | offsets |
|---|---|---|---|
| 1 | 1 | 1 | 0 |
| | 5 | 5 | 0 |
| | 2 | 2 | 0 |
| | 2.5 | 2.5 | 0 |
| | 4 | 4 | 0 |
| | 3 | 3 | 0 |
| 2 | 1 | 2 | 1 |
| | 5 | 1 | 0.5 |
| | 2 | 4 | 2 |
| | 2.5 | 5 | 2.5 |
| | 4 | 8 | 0, 4 |
| | 3 | 6 | 0, 3 |
| 3 | 1 | 3 | 1, 2 |
| | 5 | 1.5 | 0, 0.5, 1 |
| | 2 | 6 | 2, 4 |
| | 2.5 | 7.5 | 0, 2.5, 5 |
| | 4 | 1.2 | 0, 0.4, 0.8 |
| | 3 | 9 | 0, 3, 6 |
| ⋮ | ⋮ | ⋮ | ⋮ |
| $j$ | $q$ | $jq/10^{\lfloor \log_{10} jq \rfloor}$ | $q * [0..j-1]/10^{\lfloor \log_{10} jq \rfloor}$ |

## 3.3 Appearance dimensions

It is important for the labels to be easy to read, but this obvious criterion can be hard to meet for visually dense axes where the labels are long relative to the axis size. We add three new appearance dimensions to the optimization space that consider variations in formatting, font size, and orientation to increase legibility of labels in difficult labeling cases.

**Formatting.** The formatting dimension captures the possible textual representations of the labels. For instance, numbers can be formatted as decimals "5,000,000", in scientific notation "$5 \times 10^6$", or in other common formats, e.g. "5M". Depending on the range of the data and the space available for each label, different formats will maximize legibility.

Many formats have common elements that can be pulled out from the individual tick labels and placed with the axis title in order to conserve space. We call this a "factored" label. Additionally, for all numeric formats, we consider 0-extending the labels to have the same number of decimal places. This can increase legibility, but reduces available space.

**Font size.** Since font size contributes greatly to the visual appearance and implied importance of the axis we would like to respect the user's setting if possible; however, dense axes can make this difficult. Reducing the font size may be the only way to include non-overlapping labels. We include a fixed number of font size variations, $\{5\text{pts}, 7, 9, 10, 12, 14, 18, 20, 24\}$ (the LaTeX defaults), as a dimension in the search space.

**Orientation.** Finally, to handle extremely dense axis cases, we include horizontal/vertical label orientation as a binary dimension.

## 4 Optimization Function

We adapt each of Wilkinson's three components (renaming *granularity* to *density*) to work in the expanded labeling space resulting from flexible labeling and automatic generation of nice numbers, and we introduce a fourth component, *legibility*, to score the new appearance dimensions discussed in the previous section. The overall score is a weighted average of the four components. Depending on the scenario, different weighted combinations may be appropriate. However, we have tried to find a weighting that provides a reasonable compromise

Table 2: Commonly Used Variables

| | Name | Description |
|---|---|---|
| Input | $Q$ | preference-ordered list of nice step sizes |
| | $(d_{min}, d_{max})$ | data range |
| | $\rho_t$ | target label density |
| | $fs_t$ | target font size |
| | *Formats* | set of label formats (defaults shown in Table 3) |
| Other | $q$ | element of $Q$ |
| | $j$ | skip amount |
| | $i$ | index of $q$ in $Q$ |
| | $(l_{min}, l_{max})$ | labeling sequence range |
| | $v$ | equals 1 if labeling includes 0, 0 otherwise |

for general use. We suggest using

$$\begin{aligned} score = {} & 0.2 \cdot simplicity + \\ & 0.25 \cdot coverage + \\ & 0.5 \cdot density + \\ & 0.05 \cdot legibility. \end{aligned}$$

Each component is bounded by $[-\infty, 1]$, so the total maximum score is 1; scores above 0.75 are quite good. The next four subsections describe our components in more detail. Common variables are summarized in Table 2.

### 4.1 Simplicity

$$simplicity = 1 - \frac{i-1}{|Q|-1} - j + v$$

The major change to the simplicity component is the inclusion of, $j$, the skipping amount that parameterizes the algorithmically generated list of nice numbers. As seen in Table 1, larger values of $j$ result in less nice step sizes and offset amounts, so larger $j$s are penalized.

Additionally, we have slightly changed the role of $v$, the indicator variable which is 1 if the labeling includes 0. In Wilkinson's simplicity score, $v$ is divided by $|Q|$, reducing its influence if a larger $Q$ is used. This behavior doesn't seem necessary and we removed the division.

### 4.2 Coverage

$$coverage = 1 - \frac{1}{2} \frac{(d_{max} - l_{max})^2 + (d_{min} - l_{min})^2}{[0.1 \cdot (d_{max} - d_{min})]^2}$$

Our coverage function is substantially different from Wilkinson's. Rather than basing it on the ratio of the data and label ranges, we base it on the average squared distance between the extreme labels and the extreme data points. This has two advantages. First, it can be evaluated on extreme labels located both inside and outside the data range; Wilkinson's only makes sense for labels outside the data range. This is necessary since we've removed the loose labeling constraint. Second, when one extreme tick mark is located inside the data range and the other is outside the data range, it is possible for the axis and data to become "unbalanced" (where the axis is pushed to one side and the data to the other), especially on plots with few labels. The squared error form encourages balanced labelings with roughly equal amounts of whitespace on both ends of the range.

Dividing by one-tenth the data range normalizes the component and makes the value more interpretable. It will be 0 when there is about 20% whitespace, Wilkinson's maximum recommended value.

Unlike Wilkinson, we don't rely on an optimization constraint to get a good amount of coverage. Instead, the squared error form of this component makes excessive whitespace very costly and the component weights have been tuned to ensure that almost all labelings have less than 20% whitespace. However, since this is not a hard constraint, our approach can be more flexible in particularly difficult labeling scenarios.

### 4.3 Density

$$density = 1 - \max(\frac{\rho}{\rho_t}, \frac{\rho_t}{\rho})$$

The density component is controlled by a user-provided target density, $\rho_t$, given in convenient units, such as labels per inch. The density of a candidate labeling, $\rho$, is computed in the same units. If the extreme labels are outside the data range, then the density is just the number of labels divided by the physical space available for the axis (accounting for margins, etc.). However, if the extreme labels are inside the range, the label density will be higher since the label extent is less than the axis extent.

Unlike previous work, we do not directly control the number of labels on an axis. Rather, inspired by automated graphic design ideas, the number of labels is controlled indirectly through this density component. While either parameterization will produce similar labelings, density is much more convenient. For example, it makes sense to require the x- and y-axes to have the same target density, even though, depending on the aspect ratio, the numbers of labels may be very different. Similarly, a single axis undergoing interactive resizing may gain or lose labels, but the target density can remain the same.

The form of our density metric is chosen to treat relative deviations from the target density, either increasing or decreasing, equally. Thus, in contrast to Wilkinson's granularity metric, under our metric a labeling that doubles the target density and a labeling that halves it will both receive the same score.

We also tried using scoring functions based on the minimum of $\rho/\rho_t$ and $\rho/\rho_t$ or on $|\log \rho/\rho_t|$. Both have the same property of treating density changes equally. However, the first derivative of both functions goes to zero as $\rho$ increases. These relatively flat regions permitted too much variation in density and reduced the opportunities for pruning the search space.

### 4.4 Legibility

$$legibility = \frac{format + font\text{-}size + orientation + overlap}{4}$$

The simplicity, coverage, and spacing scores only consider the abstract point locations at which labels will be placed. In practice, finite-size labels have to be displayed, raising text layout issues. This is handled by the new *legibility* component which includes scoring terms for each of the three new appearance dimensions added to the optimization space. Changes in the optimization-selected appearance are driven by a soft penalty term which disallows label overlap.

#### 4.4.1 Format

Each format, $f \in Formats$, provides its own scoring function, $leg(n)$, that rates the legibility of the formatting as a function of a particular tick number, $n$. Our default set of formats is shown in Table 3, where the Iverson bracket notation, $[condition]$, indicates the function that is 1 if *condition* is true and 0 otherwise. $n = 0$ is a special case not shown in the table; since 0 is rendered the same in all the formats, $leg(0) = 1$ for any format.

The scoring functions in Table 3 were derived by trial and error on a large set of labeling tasks. We have found that factoring is generally not preferable since it increases interpretation overhead. One exception is in the case of scientific format, where leaving the exponents in the tick labels implicitly communicates that the exponents are changing (as in a log scale). A visual scan is necessary to discover that the exponents are, in fact, not changing. Therefore, for scientific format, we favor factoring out the exponent, if possible.

We assume that the same format is used on all tick labels. The extra flexibility which might be provided in difficult spacing scenarios does not seem worth the legibility challenges associated with showing multiple formats.

The format term of the legibility score for a sequence with $k$ labels, all in format $f$, is just the average of the format's legibility score across all the labels, plus a small term accounting for the whether the numbers have been 0-extended.

Table 3: Label Formats and Associated Scoring Functions

| Format | | Legibility score–$leg(n)$ |
|---|---|---|
| Decimal | 5000 | $[10^{-4} < \lvert n\rvert < 10^6]$ |
| Factored decimal | 5 (*thousands*) | 0.5 |
| K | 5K | $[10^3 \le \lvert n\rvert < 10^6] * 0.75$ |
| Factored K | 5 (K) | $[10^3 \le \lvert n\rvert < 10^6] * 0.4$ |
| M | 1M | $[10^6 \le \lvert n\rvert < 10^9] * 0.75$ |
| Factored M | 1 (M) | $[10^6 \le \lvert n\rvert < 10^9] * 0.4$ |
| Scientific | $5 \times 10^6$ | 0.25 |
| Factored scientific | 5 $(10^6)$ | 0.3 |

$$format = 0.9 * \left(\frac{1}{k}\sum_{i=1}^{k} f.leg(label_i)\right) + 0.1 * [\text{0-extended}]$$

#### 4.4.2 Font size

If the font size of a labeling is $fs$, the target font size is $fs_t$, and the minimum font size is $fs_{min}$, then the font size component is

$$font\text{-}size = \begin{cases} 1 & fs = fs_t \\ 0.2 * \frac{fs - fs_{min} + 1}{fs_t - fs_{min}} & fs_{min} \le fs < fs_t \\ -\infty & \text{otherwise.} \end{cases}$$

We do not allow the font size to go above the target font size or below a minimum size. The minimum size could be supplied by the user or by the plotting system, perhaps depending on the intended presentation format (e.g. display, print, presentation slide). However, we currently simply set it to 5pts.

The weight 0.2 in this component was chosen to be smaller than any of the legibility scores in Table 3. This favors changing format variants before decreasing the font size. This seems to be a reasonable choice in scenarios where the font size of the plot should match fonts in other plots or in accompanying text. If the plot is appearing by itself, small variations in font size may be more acceptable and this weight could be increased.

#### 4.4.3 Orientation

We consider orientation changes to be a last resort. The resulting labelings can be quite difficult to read, so we penalize them heavily.

$$orientation = \begin{cases} 1 & \text{if horizontal} \\ -0.5 & \text{otherwise.} \end{cases}$$

#### 4.4.4 Overlap

The *density* component encourages well-spaced labels. However, as a plot gets smaller, the optimization will space the labels closer than the desired density either to favor the simplicity or coverage components or to have at least two labels (our lower bound). In general, this is desirable. However, if the resulting labels are too tightly packed or overlapping they will be difficult, if not impossible, to read. The overlap score acts as a soft minimum spacing constraint, but rather than specified in terms of the abstract tick locations, it considers the actual physical extents of the labels.

We begin penalizing labels if they are closer than 1.5em apart and we forbid overlapping labels. For a labeling with $k$ labels,

$$overlap = \min_{i \in [1..k-1]} \begin{cases} 1 & \text{dist}(i, i+1) \ge 1.5\text{em} \\ 2 - \frac{1.5\text{em}}{\text{dist}(i,i+1)} & 0 < \text{dist}(i, i+1) < 1.5\text{em} \\ -\infty & \text{otherwise.} \end{cases}$$

## 5 Search Algorithm

We divide the optimization search into two parts. An outer part, Extended, performs the optimization over potential label locations. An inner portion optimizes the *legibility* component including format, font size, and orientation. The legibility portion is currently optimized by exhaustive search, since the total number of possibilities is relatively small. To search the label location portion, we leverage knowledge about the scoring function to order the search so good solutions are found earlier, allowing us to prune whole branches.

Our search takes the standard form of nested loops over the dimensions of the space. First, we iterate over $q$ within $j$, producing the step sizes in Table 1 in descending order of preference. We then iterate over $k$, the number of labels. We use the fact that the $k$ labels must at least fill the data range to compute a lower bound on $z$, the power of ten that the step size should be multiplied by. Finally, we iterate over possible starting positions of the sequence, accounting for possible phase offsets.

Within each of these loops we compute a maximum possible score given the information available (lines 1, 2, and 3). If the maximum possible score is less than the best score found so far, we can safely exit the loop, pruning that branch of the search. To guarantee correctness, we have to find a conditional maximization of each component that results in a measure that monotonically decreases as the corresponding loop variable increases. This is straightforward for *simplicity*, just set $v = 1$. Once the label range is larger than the data range, the *coverage* metric can be safely maximized by centering the label range on the data range. The maximization of the *density* metric depends on how the system chooses to layout the plot, but typically it is straightforward to find a maximization that necessarily decreases as $k$, the number of labels, increases for all values $k > k_0$, for some value of $k_0$.

If no constraints are placed on the solution, this algorithm will find a good solution very quickly. However, if constraints are applied, then it is possible that no solution exists. The strict no overlap constraint is an example of such a constraint. In these scenarios, the algorithm could keep searching for a solution indefinitely. To avoid this, we initialize *best-score* to a finite value, $-2$. This will guarantee termination and act as a bound on the worst-case labeling that can be returned. If no labeling is returned, we simply don't display one. In our implementation this only occurs when the plot is so small that having no labeling is visually preferable.

We have found that the above loop ordering gives the best early pruning results. Over a run of typical use cases, we found that the body of the inner-most loop was visited an average of 41 times, with a maximum of 158. The inner optimization over the legibility dimensions, OptLegibility, was only called on average 4 times, with a maximum of 13.

This is fast enough to be used in our research system that recomputes the axes during interactive resizing. With the legibility optimization disabled, our C# implementation (on a 2.13GHz Intel Core 2 Duo) computes an axis in 1–20ms. The lower end occurs on small axes with 2–4 labels. The high end occurs on a full screen axis with over 30 labels. Adding the legibility optimization (over 128 appearance variations) increases runtimes to 10–200ms, again depending on the number of labels in the axis. The most expensive portion of the search is in determining the pixel size of the label strings, but caching string sizes can address this problem, particularly in an interactive system where the strings will likely be reused during a session. If lots of formatting variations or font sizes are to be tried, our current exhaustive search of the appearance space could be replaced with one that permits pruning of search branches.

```
EXTENDED($d_{min}, d_{max}, \rho_t, Q = \{1,5,2,2.5,4,3\}$)
    $\mathbf{w} = [0.2, 0.25, 0.5, 0.05]$
    $result$ = NULL
    $best\text{-}score = -2$
    for $q, j \in Q \times [1..\infty]$
1       $\hat{s} = \text{simplicity}_{\max}(q, j)$
        if $[\hat{s}, 1, 1, 1] \cdot \mathbf{w} < best\text{-}score$
            break
        for $k$ in 2 to $\infty$
2           $\hat{d} = \text{density}_{\max}(k)$
            if $[\hat{s}, 1, \hat{d}, 1] \cdot \mathbf{w} < best\text{-}score$
                break
            $delta = (d_{max} - d_{min})/(k+1)/(j * q)$
            for $z$ in $\lceil log_{10}\ delta \rceil$ to $\infty$
                $l_{step} = q * j * 10^z$
3               $\hat{c} = \text{coverage}_{\max}(k-1, l_{step})$
                if $[\hat{s}, \hat{c}, \hat{d}, 1] \cdot \mathbf{w} < best\text{-}score$
                    break
                for $start$ in $\lfloor d_{max}/l_{step} \rfloor - (k-1)$
                    to $\lceil d_{min}/l_{step} \rceil$ by $1/j$
                    $l_{min} = start * l_{step}$
                    $l_{max} = l_{min} + (k-1) l_{step}$
                    $s = \text{simplicity}(q, j, l_{min}, l_{max}, l_{step})$
                    $d = \text{density}(l_{min}, l_{max}, l_{step})$
                    $c = \text{coverage}(l_{min}, l_{max}, l_{step})$
                    if $[s, c, d, 1] \cdot \mathbf{w} < best\text{-}score$
                        continue
                    $l, l_{format}$ = OPTLEGIBILITY($l_{min}, l_{max}, l_{step}$)
                    $score = [s, c, d, l] \cdot \mathbf{w}$
                    if $score > best\text{-}score$
                        $best\text{-}score = score$
                        $result$ = LABEL($l_{min}, l_{max}, l_{step}, l_{format}$)
    return $result$
```

## 6 Results

Though evaluating labelings can be somewhat subjective, we provide both quantitative and qualitative results to demonstrate the improvement of our algorithm over the previous work. First, we demonstrate, at an aggregate level, that our algorithmically expanded space of nice numbers and our use of flexible labelings allow us produce labelings with better density and coverage than the previous work. Second, we provide examples of our formatting variations maintaining legibility under changing labeling scenarios. Third, we compare labeling results produced by our algorithm to those produced by Heckbert's algorithm, R's pretty function, and Wilkinson's optimization approach on a small number of real data sets.

### 6.1 Quantitative comparison with other algorithms

By gathering aggregate statistics from runs on random labeling tasks, we provide a quantitative comparison of our algorithm to Wilkinson's and Heckbert's techniques, as well as to a limited version of our algorithm that only generates loose labelings. Each algorithm used the defaults recommended by the authors, with the exception that we disabled our legibility component for this test.

We produced a set of random labeling tasks by generating data ranges ($n = 10,000$) in the interval $(-100, 400)$ which were generated by drawing the minimum and maximum uniformly at random from the range and swapping them if necessary. We then generated the desired number of tick labels from a uniform distribution between 2 and 10.

#### 6.1.1 Expanded space of nice numbers

Figure 2 shows the distributions of coverage, $(l_{max} - l_{min})/(d_{max} - d_{min})$, and relative density, $\rho/\rho_t$, of the labelings produced by the four algorithms on the random labeling tasks.

As shown in the top-left histogram, Heckbert's algorithm produces a wide range of coverages. For the subset of cases where three or fewer labels were requested, over 50% of the labelings have more than 50% whitespace, and 3% have more than 66% whitespace. As Cleveland suggests, we should be wary of introducing so much whitespace. Wilkinson tackles this problem by substantially expanding the list $Q$, which should improve the worst-case coverage, and by enforcing a minimum coverage constraint in the optimization search. The left plot

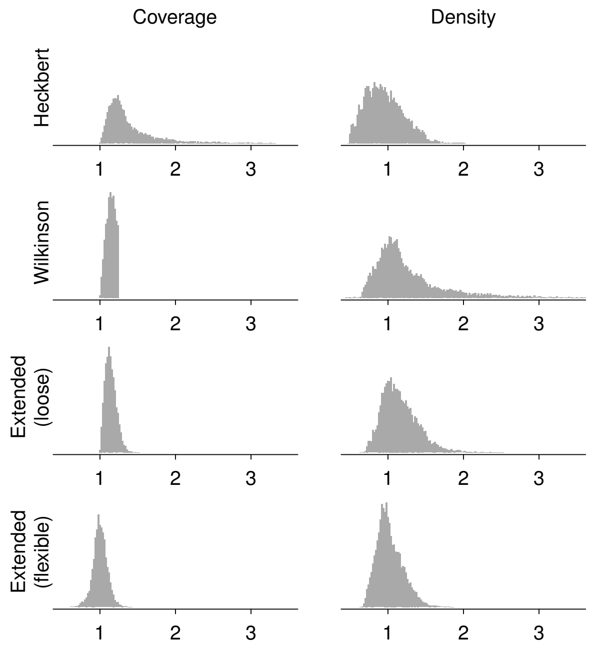


Fig. 2: The distributions of coverage and density of 10,000 random labelings generated by Heckbert's, Wilkinson's, and our approach. Our approach is able to maintain both good coverage (near 1) and good relative density (near 1).

demonstrates the effectiveness of this approach; the coverage distribution is now clustered near 1. However, there is a cost. As shown in the right-hand histogram, Wilkinson's method is not able to respect the density of tick marks requested by the user. There is a substantial tail off to the right, indicating cases where the algorithm returned too many labels. In 10% of the cases more than double the requested number of labels was returned and in 1.6% of the cases three times as many labels were returned. These dramatic changes in density will negatively affect the visual quality of the plots.

The third row shows the effect of our expanded space of nice numbers (without flexible labeling). Unlike Heckbert's or Wilkinson's approaches, we more effectively control both the coverage and density properties of the labeling, with the distributions clustered around 1 and more moderate tails.

We gain this control by algorithmically generating step sizes and sequence offsets. Figure 3 shows the distribution of step sizes chosen by each algorithm (very small non-zero counts have been exaggerated to make them visible). Our algorithm has expanded the initial list, $Q = [1,5,2,2.5,4,3]$, with new step sizes, e.g. 1.2 and 1.25, to solve the more difficult labeling scenarios. The generated numbers are reasonably nice—the number of decimal places is small and they are relatively amenable to mental manipulation. The algorithm has chosen to offset about 16% of the labeling sequences. Because we permit offsetting, it is possible to produce labelings that cross 0 but do not include it. This happens about 4% of the time in the loose labeling case.

### 6.1.2 Flexible labeling

Figures 2 and 3 also show the aggregate results of our algorithm when flexible labeling is allowed. The coverage and density distributions improve substantially and our algorithm can use nicer numbers more often. Importantly, we were also able to reduce the number of labelings that crossed 0 but did not include it from 4% to 1%. In addition, only 4% of sequences were offset. Example differences are shown in Figure 4. By allowing flexible labeling, we can get nicer numbers and sequences that include 0 in situations where loose labeling cannot.

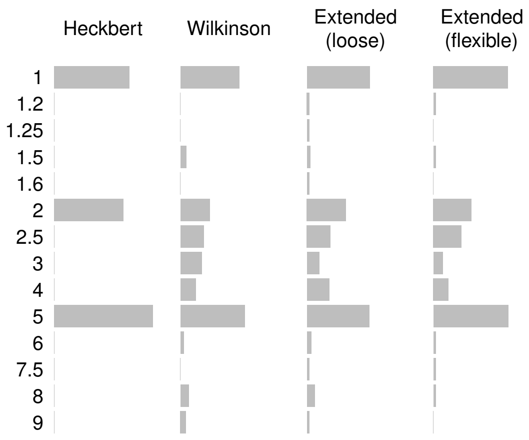


Fig. 3: Step sizes used by each algorithm in the 10,000 random labelings. Heckbert and Wilkinson both use a static set of possible step sizes. Our skipping approach generates additional step sizes (e.g. 1.2, 1.25, etc.) that were not in the original set in order to produce labelings with good density and coverage.

## 6.2 Appearance dimensions

Our appearance dimensions were selected to support better labeling for dense axes, where the label text length is a significant fraction of the axis length. Dense axes can arise from small plot sizes, long label text (for very large or very small numbers), or larger requested font sizes.

Figure 5 demonstrates the typical behavior of our optimization-driven labeling on smaller and smaller plot sizes while maintaining a fixed font size. By choosing abbreviated label formats, splitting labels, and finally rotating labels, our optimization is able to find reasonable labeling solutions. Note that in the next-to-last axis the optimization avoids overlap or undesirable rotation by decreasing the niceness of

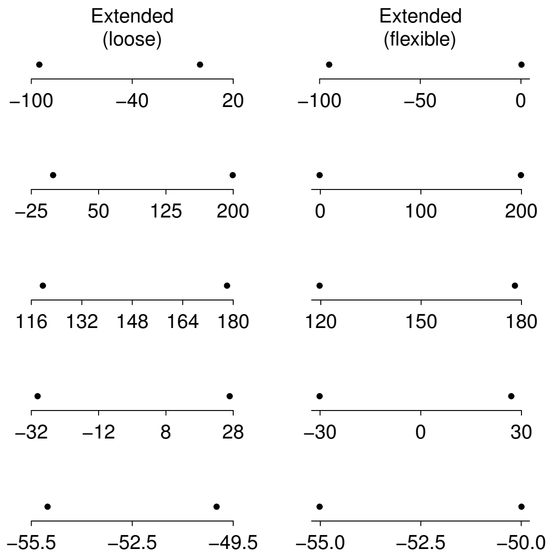


Fig. 4: Permitting flexible placement of the extreme labels, inside or outside of the data range (shown with black dots), permits better labelings. Nicer numbers can be selected, the target density (in these examples, about 3 labels) can more easily be matched, and 0 can be included more frequently in ranges that cross 0.

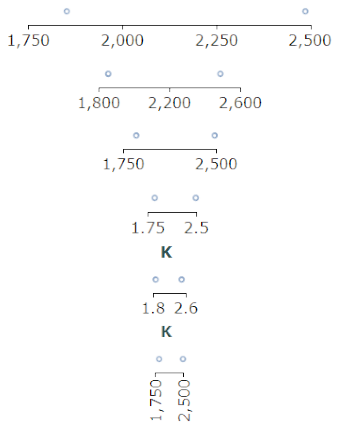


Fig. 5: Our formatting dimensions allow us to keep good labelings, even as the plot size shrinks to 30 pixels.

the numbers. This is only possible because our approach considers label location and formatting in the same optimization search.

In Figure 6, we show labelings of the same data set for three different display scenarios, generated by varying the target density and target font size. Again, variation in the label appearance dimensions ensures that labels do not overlap.

### 6.3 Real data sets

We now qualitatively compare our extended optimization to current algorithms with several concrete example plots. Table 7 shows the resulting labels after running our algorithm, Wilkinson, Heckbert, and R's pretty on three datasets. Axis sizes are around 300 pixels. The desired density was adjusted to be about 1 tick per 100 pixels, so each axis should have 3–4 labels.

Heckbert's labeling extends far beyond the data range in all the examples, which squashes the data points into a small portion of the plot. In the *students* example, this squashing hides the interesting vertical banding pattern in the data. R's plotting function reduces unwanted whitespace by dropping labels outside the data range. This can make it more difficult to estimate the data range when an extreme label is far from the extreme data point (y-axis of both *pearson* and *stimulus*). Wilkinson's algorithm can either produce too many labels (x-axis of *pearson*) or too few (x-axis of *students*).

In comparison, our extended algorithm produces the right number of ticks, a small amount of whitespace, and relatively nice numbers. Additionally, as shown in *stimulus*, our appearance dimensions improve the display of long labels.

## 7 Discussion and Future Work

We have described a number of extensions to Wilkinson's optimization-based labeling approach that result in better labelings across a wide range of axis scenarios. Implementations of our algorithm are available at `http://www.justintalbot.com/research/axis-labeling`. However, our approach still has a number of limitations and there remains substantial future work.

**Limitations**. Our density component is parameterized by a target number of labels per unit of length. However, the visual density is also a function of the length of the label text. It may be more useful to express the density in terms of a graphic design-inspired measure, such as blackness, to better capture the actual visual effect of a labeling. Doing so may obviate the need for our separate overlap score.

Our appearance dimensions work well for horizontal axes. But for vertical axes (with horizontal labels), overlap is not principally determined by the length of the label, so our appearance variations are seldom used. For vertical axes, the length of the labels determines how big the left- or right-hand margins must be, and therefore how much space is available for the data portion of the plot. It may be useful to add an optimization criteria that tries to maximize the space available for data. A related limitation is that currently we optimize a single axis at a time. In reality, the labelings of the x- and y-axes interact and joint optimization may be preferable.

Our algorithm currently only works for linear scales and it's not clear how to extend it to non-linear scales.

**Future work**. To some extent, our proposed optimization components and the chosen weights are *ad hoc*. More work remains to be done in grounding these in information visualization theory. In particular, controlled user studies of axis-based analytic tasks would contribute greatly to our understanding of what makes effective labelings and would lead to refined optimization functions.

So far we have only considered simple numeric data. An obvious extension is to apply the optimization framework to other data types. We've made good initial progress on incorporating date-time labeling into our extended optimization system and are currently working on defining the interface between the type definition and the optimization system in order to support type extensibility. Categorical labels are particularly challenging and leveraging text layout algorithms might be profitable.

Automatic axis labeling is just one component of the more general area of automated graphic design. The optimization approach applied here could be applied to the layout of an entire plot, including the placement of titles, axis titles, legends, data labels, or aspect ratio (via banking to 45°). Such an approach could eventually enable fully automated reuse of plots across a wide range of presentation formats, including static images in documents, interactive plots on computer screens, large slide show-style plots for communication, or very small plots presented on mobile devices.

## Acknowledgments

The authors would like to thank Lee Wilkinson for providing an implementation of his labeling algorithm and John Gerth for careful editing. The stimulus data set was gathered by Veronique de Rugy. This work was supported by FODAVA grant 0937123.

## References

[1] W. S. Cleveland. *The Elements of Graphing Data*. Wadsworth Publ. Co., Belmont, CA, USA, 1985.

[2] W. S. Cleveland and R. McGill. The many faces of a scatterplot. *Journal of the American Statistical Association*, 79(388):807–822, 1984.

[3] P. Heckbert. Nice numbers for graph labels. In A. Glassner, editor, *Graphics Gems*, pages 61–63 657–659. Academic Press, Boston, 1990.

[4] J. A. Nelder. Algorithm AS 96: A simple algorithm for scaling graphs. *Applied Statistics*, 25(1):94–96, 1976.

[5] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2009. ISBN 3-900051-07-0.

[6] R. E. Reys, J. F. Rybolt, B. J. Bestgen, and J. W. Wyatt. Processes used by good computational estimators. *Journal for Research in Mathematics Education*, 13(3):183–201, 1982.

[7] D. N. Sparks. Algorithm AS 44: Scatter diagram plotting. *Applied Statistics*, 20(3):327–331, 1971.

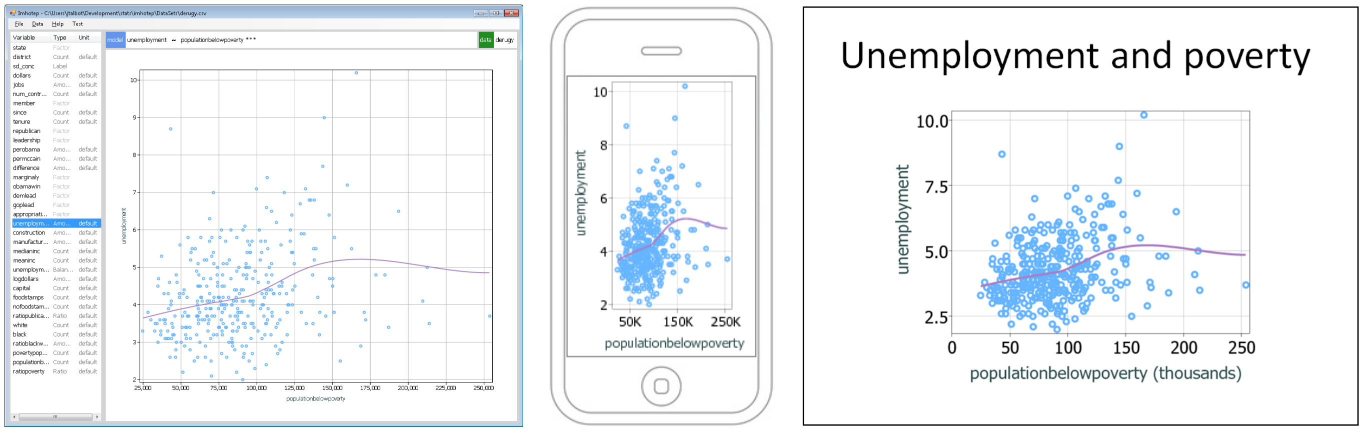
[8] W. D. Stirling. Algorithm AS 168: Scale selection and formatting. *Applied Statistics*, 30(3):339–344, 1981.

[9] R. P. Thayer and R. F. Storer. Algorithm AS 21: Scale selection for computer plots. *Applied Statistics*, 18(2):206–208, 1969.

[10] E. R. Tufte. *The Visual Display of Quantitative Information*. Graphics Press, Cheshire, CT, USA, 1986.

[11] L. Wilkinson. *The Grammar of Graphics (Statistics and Computing)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[12] L. Wilkinson. Personal communication, March 2010.



| Variable | Type | Unit |
|---|---|---|
| state | Factor | |
| district | Count | default |
| sd_conc | Label | |
| dollars | Count | default |
| jobs | Amo... | default |
| num_contr... | Count | default |
| member | Factor | |
| since | Count | default |
| tenure | Count | default |
| republican | Factor | |
| leadership | Factor | |
| perobama | Amo... | default |
| permccain | Amo... | default |
| difference | Amo... | default |
| marginaly | Factor | |
| obamawin | Factor | |
| demlead | Factor | |
| goplead | Factor | |
| appropriati... | Factor | |
| unemploym... | Amo... | default |
| construction | Amo... | default |
| manufactur... | Amo... | default |
| medianinc | Count | default |
| meaninc | Count | default |
| unemploym... | Balan... | default |
| logdollars | Amo... | default |
| capital | Count | default |
| foodstamps | Count | default |
| nofoodstam... | Count | default |
| ratiopublica... | Ratio | default |
| white | Count | default |
| black | Count | default |
| ratioblackw... | Amo... | default |
| povertypop... | Count | default |
| populationb... | Count | default |
| ratiopoverty | Ratio | default |

(a) Interactive analytic display (b) Mobile device (c) Presentation slide

Fig. 6: By changing the target font size and target density input parameters to our algorithm, we can format axes for three different display scenarios. Optimization over the appearance dimensions ensures that labels remain legible in each.

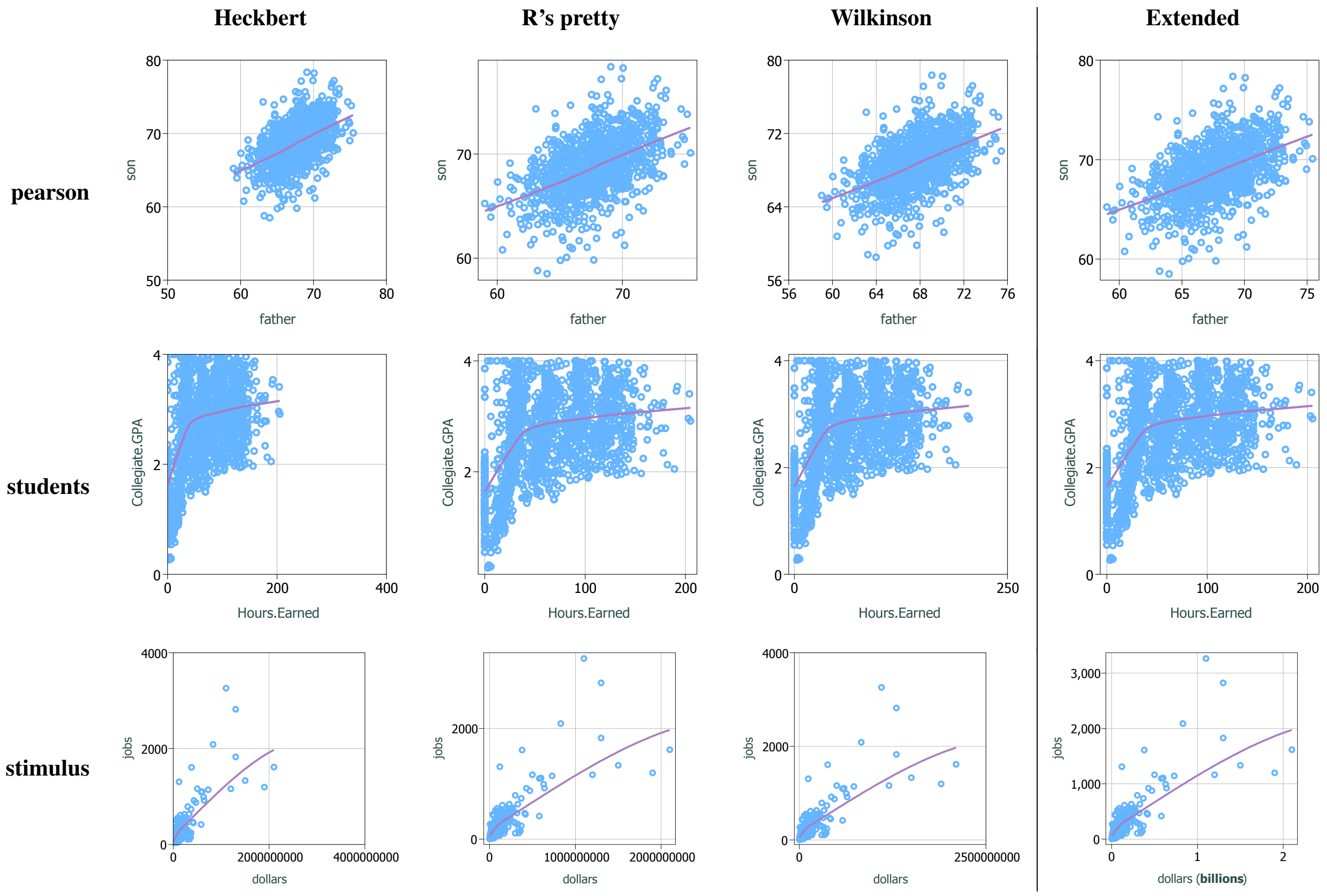


Fig. 7: Comparison of our extended algorithm with Heckbert, R's pretty, and Wilkinson on four data sets. Our extended algorithm better manages label density and ensures that the labels cover the data range well without introducing too much whitespace in the plots.