An excerpt from

The Graphviz Cookbook

Rod Waldhoff
rwaldhoff@gmail.com
http://heyrod.com/

ABOUT THIS BOOK

The Graphviz Cookbook, like a regular cookbook, is meant to be a practical guide that *shows you how to create something tangible* and, hopefully, *teaches you how to improvise your own creations* using similar techniques.

The book is organized into four parts:

Part 1: Getting Started introduces the Graphviz tool suite and provides "quick start" instructions to help you get up-and-running with Graphviz for the first time.

Part 2: Ingredients describes the elements of the Graphviz ecosystem in more detail, including an in-depth review of each application in the Graphviz family.

Part 3: Techniques reviews several idioms or "patterns" that crop up often when working with Graphviz such as how to tweak a graph's layout or add a "legend" to a graph. You might think of these as "micro-recipes" that are used again and again.

Part 4: Recipes contains detailed walk-throughs of how to accomplish specific tasks with Graphviz, such as how to spider a web-site to generate a sitemap or how to generate UML diagrams from source files.

6.7 `patchwork`

`patchwork` is a relatively new addition to the Graphviz family.

`patchwork` is a layout engine like `dot` or `neato`, but rather than creating the conventional “boxes and arrows” view of a graph, `patchwork` generates a type of visualization known as a treemap.

Treemaps

Most of the visualizations in this book—indeed most of the visualizations one might create with Graphviz—are designed to illustrate the *connections* between data points (in the form of the edges between nodes).

Treemaps are a little bit different. While a treemap can represent certain associations between nodes, a treemap is *designed* to illustrate the distribution of a quantifiable value across several, potentially hierarchically-nested, categories.

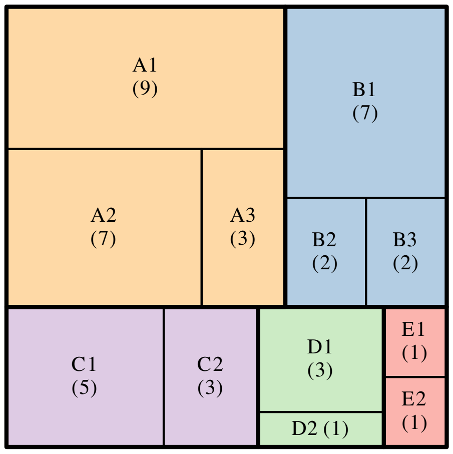

Figure 6.1: *A **treemap** uses rectangular tiles and clusters of tiles to represent the distribution of some value across hierarchical categories. The relative* area *of each tile (and cluster) is proportional to the relative* magnitude *of the corresponding category (and “super-category”).*

Specifically, a ***treemap*** is composed of rectangular *tiles*. The *area* of each tile is proportional to the value of the corresponding data point. Some tiles are further subdivided into even smaller tiles. These meta-tiles are called clusters, and represent “super-categories” in the hierarchical data, while the tiles within a cluster represent the “sub-categories” nested beneath that super-category. In this way a treemap partitions a rectangle into tiles and clusters of tiles (and other clusters) such that the *area* of each tile (and cluster) is proportional to the *magnitude* of the corresponding value (and collection of values) in the original data set.

Compared to other charts

Like a bar or pie chart, a treemap offers a way to compare the relative size of some numerical value across several categories.

Consider the stereotypical bar chart data set, sales by region:

Region	Revenue (YTD)
Northeast	$68,794
Southeast	$8,000
Central	$16,000
Northwest	$54,311
Southwest	$61,842

Table 6.1: *Sales By Region (Year-to-Date)*

In a *bar chart* each sales region is represented by a rectangle. The *width* of each rectangle is proportional to the revenue earned in that region. For example, since the Central region's revenue is twice that of the Southeast region, the bar that represents the Central region should be twice as wide as that of the Southeast.

In a *pie chart*, each sales region is represented by a wedge of circle. The internal *angle* of each wedge is proportional to the revenue earned in that region. Here, since the Central region's revenue is twice that of the Southeast region, the *internal angle* of the wedge that represents the Central region is twice that of the wedge that represents the Southeast.

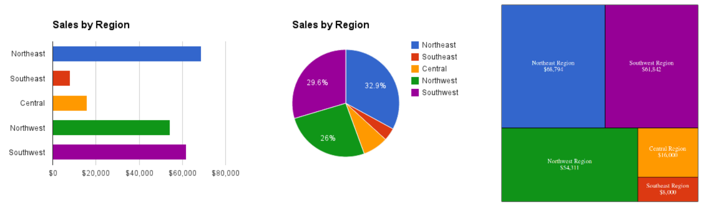

Figure 6.2: *Comparing bar chart, pie chart and treemap visualizations of the data in Table 6.1. The bar chart uses the* width *of each bar to represent the magnitude of the corresponding data point. The pie chart uses the* angle *of each wedge. The treemap uses the* area *of each tile.*

In a *treemap*, each region is again represented by a rectangle, but it is the *area* of each rectangle that is proportional to revenue. Since the Central region's revenue is twice that of the Southeast region, the total *area* of the Central region's rectangle is twice that of the Southeast.

Strengths and Weaknesses

The unusual rules by which treemaps are constructed introduces some properties that set treemaps apart from more conventional diagrams like pie charts and bar charts.

Treemaps excel at illustrating hierarchical distributions. Since treemaps cluster the tiles that belong to a particular group together, one can easily compare the relative size of each group, the relative size of the items *within* a given group, and the relative size of elements regardless of their level in the hierarchy.

Treemaps are well suited for visualizing large data sets. Since they make use of both dimensions, and with little or no gap between tiles, treemaps are much more space-efficient than bar and column charts.

Despite their initial novelty, treemaps may be more intuitively understandable than some of the alternatives. Viewers intutively look to the *area* of the corresponding region when comparing values on chart. But the area of each wedge (on a pie chart) or rectangle (on a bar or column chart) can be misleading.

On a pie chart, it's not the area but the *angle* of the wedge that is determined by the source data. Since the area of each wedge grows with the *square* of the radius, doubling the radius of the circle will *quadruple* the difference in area between two wedges.[1]

Even bar and column charts are susceptible to this kind of distortion. Changing the *height* of each bar or the *width* of each column can increase or decrease the *absolute* difference in size between two regions on the chart, dramatically changing the impression that that the chart creates.

That said, when the size of the data set is reasonably small, it is much easier to compare values in one dimension (on a bar or column chart) than in two dimensions (on a treemap). While a quick scan will tell you the largest or smallest rectangle on a bar or column chart, the difference may be less obvious on a treemap.

Moreover, the effectiveness of a treemap rapidly diminishes when the individual tiles have dramatically different aspect ratios. People aren't very good at estimating the area of very tall (or very wide) rectangles. `patchwork` tries to create roughly square tiles, but this isn't always possible, especially when clustering is taken into account.

[1] Doubling the radius of the circle doesn't change the *ratio* between the area of two wedges—if wedge A was twice the size of wedge B it will still be twice the size of wedge B after the doubling—but it changes the raw *difference* between the areas.

Example of Use

`patchwork` is invoked like all of the other Graphviz layout engines and accepts the same common command line parameters. The command:

```
> patchwork -Tpng mygraph.gv -o mygraph.png
```

uses `patchwork` to creates a PNG image named `mygraph.pdf` from the DOT file `mygraph.gv`.

`patchwork` friendly graphs

Just as treemaps are a bit unique among graph visualizations, `patchwork` is a bit unique among the Graphviz layout engines.

Consider the simple digraph in Figure 6.3. As you can see, `patchwork` has ignored the edge relationships between the nodes and partitioned the canvas into one equally sized tile per node.

```
digraph {
  A -> B
  B -> C
  B -> D
  D -> E
}
```

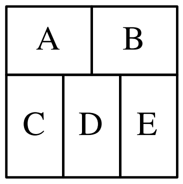

Figure 6.3: *A simple digraph, rendered by* `patchwork`*. Note that* `patchwork` *ignores the edge relationships between nodes and simply partitions the canvas into one tile per node.*

To use `patchwork` effectively, we should need keep three key points in mind:

1. `patchwork` completely ignores the edge relationships between nodes.[2]
2. In order for `patchwork` to do anything interesting, a `patchwork`-specific node attribute named `area` must be associated with each node. (The default `area` value is `1`, which is why the the canvas in Figure 6.3 was paritioned into equally-sized tiles.)
3. Like the other layout engines, `patchwork` will keep the nodes within a cluster together. In particular, just as `patchwork` converts each node into a tile, it converts a cluster of nodes within the graph into a cluster of tiles within the treemap.

[2] This is described as a bug on the `patchwork man` page, but it doesn't seem likely that this will change in the foreseeable future. It's not even clear *what* role edges would play given the logic of the current `patchwork` algorithm.

A “flat” treemap

Since `patchwork` ignores edges, we might as well use an un-directed graph and leave out the edge declarations. Introducing an explict `area` attribute to each node, we’re left with a basic treemap, as illustrated in Figure 6.4.

```
graph {
  A [area=1]
  B [area=2]
  C [area=3]
  D [area=2]
  E [area=5]
}
```

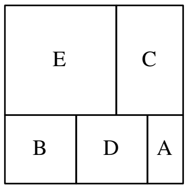

Figure 6.4: `patchwork` *paritions the canvas into tiles, one for each node. The relative area of each tile is proportional to the relative* `area` *of each node.*

Absent any clusters, `patchwork` simply partitions the canvas into tiles representing each node. The ratio of the area of each tile to the total area of the treemap is the same as the ratio of the `area` of each node to the total area of all nodes in the graph

A “hierarchical” treemap

In order to group the tiles in the treemap into clusters, we group the nodes in the graph into clusters.

```
graph {
  subgraph cluster1 {
    A [area=1]
    B [area=2]
  }
  subgraph cluster2 {
    C [area=3]
    subgraph cluster3 {
      D [area=2]
      E [area=5]
    }
  }
}
```

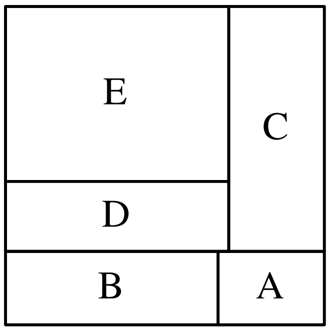

Figure 6.5: `patchwork` *arranges the tiles on the treemap such that each cluster of nodes is grouped into a contiguous rectangle on the treemap.*

As seen in Figure 6.5, `patchwork` arranges the tiles on the treemap such that each cluster of nodes is grouped into a contiguous rectangle on the treemap.

To make the clusters easier to identifty, we can use the `penwidth` graph attribute to draw a border around each cluster, as demonstrated in Figure 6.5.

```
graph {
  penwidth=3
  subgraph cluster1 {
    A [area=1]
    B [area=2]
  }
  subgraph cluster2 {
    C [area=3]
    subgraph cluster3 {
      D [area=2]
      E [area=5]
    }
  }
}
```

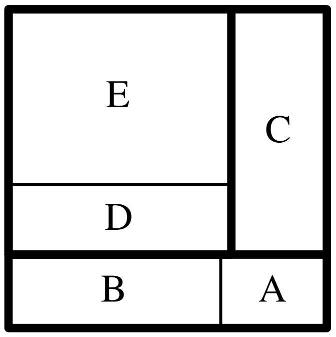

Figure 6.6: *Adding the* `penwidth` *graph attribute makes it easier to see the cluster boundaries within the treemap.*