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.

Chapter 10

Using gvpr to style graphs

The *C*ascading *S*tyle-*S*heet language, **CSS**, provides a mechanism for "styling" HTML documents. To style an HTML document using CSS, one associates patterns (known as *selectors*) with style attributes. A CSS processor applies the specified style to each element in the HTML document that matches the associated pattern.

For example, the CSS rule:

```
p { border-left: 2px solid black; }
```

instructs the CSS processor to draw a two-pixel wide, solid, black border around each paragraph (`p`) element. The rule:

```
.redtext { font-color: red; }
```

instructs the CSS processor to render the text of any element with the *class* `redtext` in red.

Using `gvpr`, Graphviz's `awk`-like DOT file processor, we can do something quite similar for graphs.

Like a CSS processor, `gvpr` can apply user-specified actions to elements that match a given pattern. All we need to do is define a *predicates* that select the nodes, edges or graphs we're interested in and *actions* that apply the appropriate style attributes.

For example, the `gvpr` program:

```
E [style=="dashed"] { color = "blue"; }
```

will select every edge with the `dashed` style and color it blue, and the `gvpr` program:

```
N [shape=="box"] { fontname = "Helvetica-Oblique"; fontsize=16; }
```

will select every node with the `box` shape and render its label in a `16`-point `Helvetica-Oblique` typeface. (See Figure 10.1 and Figure 10.2.)

```
graph {
  A [label="Lorem" shape="box"]
  B [label="Ipsum"]
  A -- B
}
```

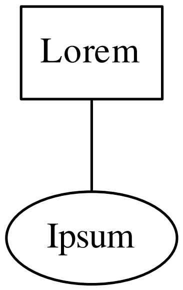

Figure 10.1: *A graph before being styled by* `gvpr`. *(See Figure 10.2 for the "after" version.)*

```
> cat mygraph.gv | gvpr -c
 "N [shape=="box"] {
 fontname="Helvetica-Oblique";
 fontsize=16; }"
graph {
  A [fontname="Helvetica-Oblique",
    fontsize=16,
    label=Lorem,
    shape=box];
  B [label=Ipsum];
  A -- B;
}
```

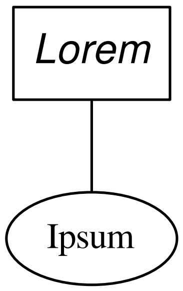

Figure 10.2: *A graph after being styled by* `gvpr`. *(See Figure 10.1 for the "before" version.)*

10.1 More robust selectors

Be careful when applying this technique. Our selectors may not be quite as clever as you imagine.

`gvpr`'s attribute-based patterns compare the given value to the *full* value of the attribute. Hence:

```
E [style="dashed"]
```

will match:

```
A -- B [style="dashed"]
```

but not:

```
A -- B [style="bold,dashed"]
```

One workaround is to enumerate all of the variations that you are interested in.

Another approach is to fall back to a more "manual" form of filtering. For instance:

```
E {
  if(match($.style,"dashed") != -1) {
    color = "blue";
  }
}
```

Figure 10.3: *A* `gvpr` *script containing a rule that matches* all *edges, but then relies on an* `if` *statement to only a modify specific nodes.*

The script in Figure 10.3 matches *every* edge in the input graph, but then uses an `if` condition to determine whether or not the edge's `style` attribute contains the string `dashed` before applying the corresponding action.

10.2 Custom attributes

We can get even more CSS-like behavior by matching against custom attributes. For instance, given the graph:

```
graph { A [class="foo"] }
```

we might apply the `gvpr` script:

```
graph { N [class="foo"] { shape="diamond"; }
```

to change the shape of each node with the specified `class` value.