Paper 267-2012

Off the Beaten Path: Create Unusual Graphs with GTL

Prashant Hebbar, SAS Institute Inc., Cary, NC

ABSTRACT

You open the latest edition of Time magazine and see a unique spiral bar chart of revenue shortfall by state, or your coworker sends you a link to an interesting population pyramid graph on a Web site and you ask yourself: “Can I do that using SAS®”? Often, as it is for these two graphs, the answer is “Yes.” With a little bit of data transformation and some creativity, you can generate spiral plots, conditional regression trees, and population pyramids using ODS Graphics, without a shred of annotate code.

This presentation covers many tricks and tips to create unique graphs that grab the reader’s attention, and also deliver the information effectively. Learn how to stretch the boundaries of what is possible with ODS Graphics.

INTRODUCTION

The ODS Graphics System allows you to create modern, analytical graphs. It comprises the SG Procedures (also called the ODS Graphics Procedures), the Graph Template Language (GTL), the ODS Graphics Designer, and the ODS Graphics Editor.

GTL is used to define a STATGRAPH template using the TEMPLATE Procedure. The SGRENDER procedure lets you associate the data for the graph with a template and creates the graph. GTL offers a ‘building-block’ approach to specifying your graph template. You can combine the following in numerous ways to create different graphs:

- *Layouts*: Overlay: OverlayEquated, Gridded, Region, Lattice, DataLattice, and DataPanel.
- *Plots*: Scatter: Series, Histogram, Bubble, HighLow, BoxPlots, BarCharts, Fit plots, and more.
- *Text and Legends*: EntryTitle, EntryFootnote, Entry, DiscreteLegend, ContinuousLegend, and more.
- *Other features*: Expressions, conditional blocks, dynamic, and macro variables.

This system is now a part of Base SAS starting with SAS 9.3. In this presentation, you will see how we use GTL to create some atypical graphs. Many of these techniques can also be used to great effect in the SG Procedures.

You can download a copy of the code and data used in this paper from the SAS Technical Papers and Presentations site at http://support.sas.com/rnd/papers/index.html. Find the entry "Off the Beaten Path: Create Unusual Graphs with GTL" under the section for SAS Presentations at SAS Global Forum 2012, and download the examples. The code in this paper was tested using SAS® 9.3 software.

POPULATION PYRAMID

Traditionally, horizontal bar charts have been used to show population variation by age. One of the novel presentations of this graph uses multiple overlaid series plots to create a gossamer-like effect. (See ExcelCharts Blog in the References section.)

Here is a GTL version of the population pyramid for USA:

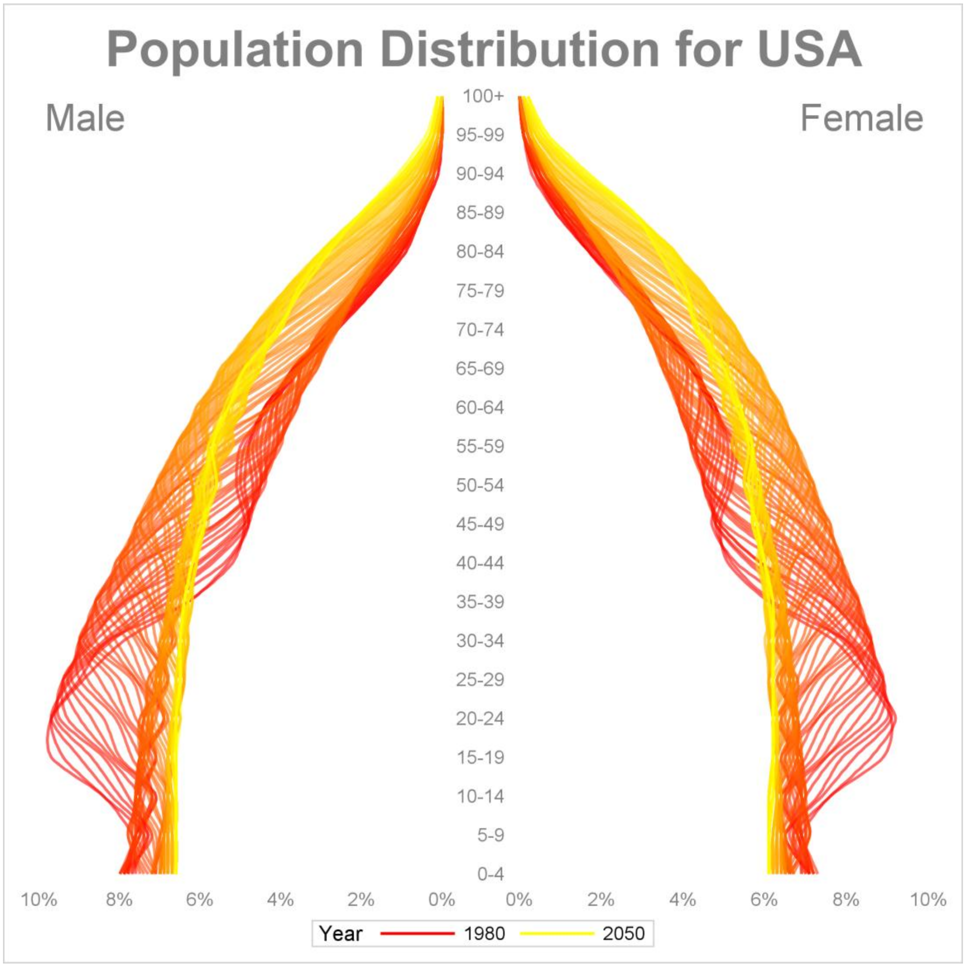


Figure 1: Population Pyramid for USA

In the next few sections, you will see the details about how this graph was implemented.

DATA PREPARATION

The data for this plot comes from the US Census Bureau in CSV format and was imported into a SAS data set. Here are the relevant columns:

Obs	Country	Year	Age	Percent_Male	Percent_Female	...
1	United States	1980	Total	100	100	
2	United States	1980	0-4	7.6	6.9	
3	United States	1980	5-9	7.7	6.9	
4	United States	1980	10-14	8.4	7.6	
...						

Table 1: Population Data

After we filter out any observations where age = Total or age = (NA), we need to scale the columns PERCENT_MALE and PERCENT_FEMALE to [0, 1] as columns PCT_MALE and PCT_FEMALE respectively, to allow us to use the *percent.* format.

TEMPLATE CODE

To implement this in GTL, you can use the lattice layout with two cells arranged in one row. Each cell contains an overlay layout with a series plot of age against gender-specific population percentages, grouped by year. Here is the program snippet:

```
proc template;
  define statgraph pop_pyramid;
  ...
  layout lattice / columns=2 columnDataRange=unionAll columnWeights=(0.47 0.53);

    layout overlay / wallDisplay=none yAxisOpts=(display=none)
                     xAxisOpts=(reverse=true display=(tickvalues)
                           linearOpts=(tickValueFormat=percent. viewMin=0) ...);
      entry hAlign=left 'Male' / vAlign=top ... ;
      seriesPlot x=pct_male y=age / group=year_attrvar smoothConnect=true
                       lineAttrs=(pattern=solid thickness=2) dataTransparency=0.5;
    endLayout;

    layout overlay / wallDisplay=none yAxisopts=(display=(tickValues)...)
                     xAxisOpts=(display=(tickValues)
                           linearOpts=(tickValueFormat=percent. viewMin=0) ...);
      entry halign=right 'Female' / vAlign=top ...;
      seriesplot x=pct_female y=age / group=year_attrvar smoothConnect=true
                       lineAttrs=(pattern=solid thickness=2) dataTransparency=0.5;
    endLayout;

  endLayout;
  ...
```

The first cell (on the left) has a reversed X axis to give you the 'reflection' effect. In addition, we turned off the Y axis display here, but retained Y axis tick values in the second cell (on the right). We have set the column weights explicitly to even out the space allocation for the wall area of the two cells. This compensates for the axis of the right cell taking up some space, compared to the entire space being occupied by the wall in the left cell.

The series plots have a data transparency of 0.5 to help provide a 'silky' feel for the lines. We have also set the SMOOTHCONNECT option to true in order to reduce sharp transitions.

If you observe carefully, the group column YEAR_ATTRVAR is not in the original data! This is a discrete attribute variable that is being used to map the colors for the year variable using a discrete attribute map as shown below:

```
...
  discreteAttrMap name='year';
    value '1980' / lineAttrs=(color=cxff0000);
    value '1981' / lineAttrs=(color=cxff0300);
    value '1982' / lineAttrs=(color=cxff0700);
    ...
    value '2050' / lineAttrs=(color=cxffff00);
  endDiscreteAttrMap;

  discreteAttrVar attrVar=year_attrvar var=year attrMap="year";;
...
```

The 'year' discrete attribute map has been coded to map every year to a unique, stepped line color starting from 1980 (color = red) and ending with 2050 (color = yellow).

Next, let us look at how the legend was constructed. As shown below, the discrete legend directly references two legend items, one for each end range of colors for the YEAR column. If we had referenced the series plot instead, we would have ended up with too many legend items to be meaningful.

```
...
  legendItem name='1980' type=line / label='1980'
                                    lineAttrs=(color=cxFF0000 thickness=2);
  legendItem name='2050' type=line / label='2050'
                                    lineAttrs=(color=cxFFFF00 thickness=2);
  ...
  discreteLegend '1980' '2050' / title='Year';
...
```

We are finished with the data and template. All that is left is to generate the graph by submitting the SGRENDER procedure below:

```
proc sgrender data=pop_clean template=pop_pyramid;
run;
```

A FURTHER TWIST

The above construction can be extended by overlaying the data for another country on the first one so that we can compare them. Below is an example of such an overlay for USA and Qatar:

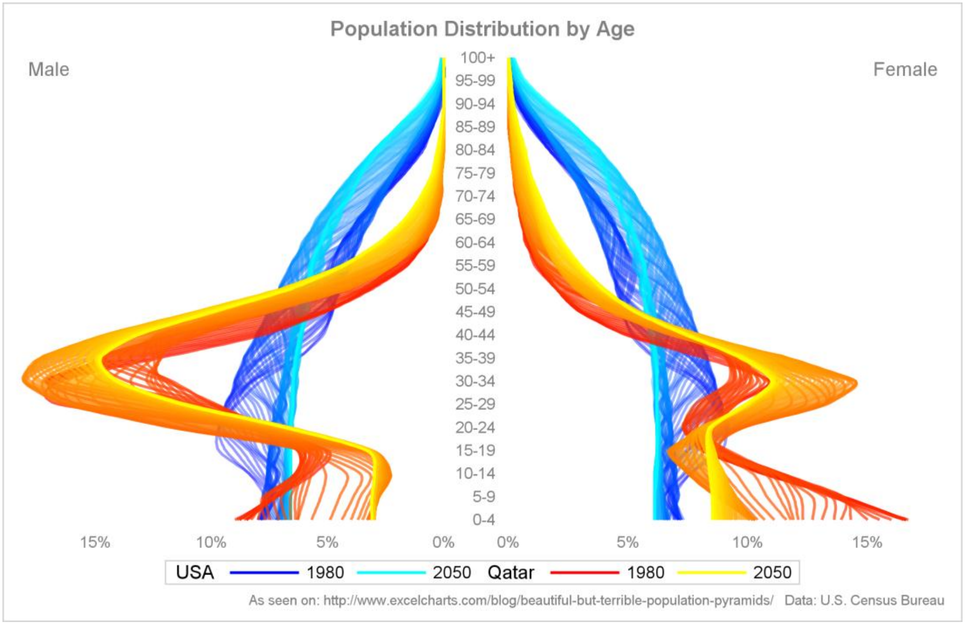


Figure 2: Populations of USA and Qatar

SPIRAL BAR CHART

If you had browsed the July 25th 2011 issue of Time magazine, you would have seen an interesting presentation of a bar chart that seemed to spiral out, drawing attention to the revenue shortfall for some states in the USA even after accounting for online sales tax collection.

The bars were sorted clockwise in descending order with stacked (cumulative) bars for taxes from online sales and out-of-state catalog sales overlaid against the budget shortfall.

Here is a GTL version of that display:

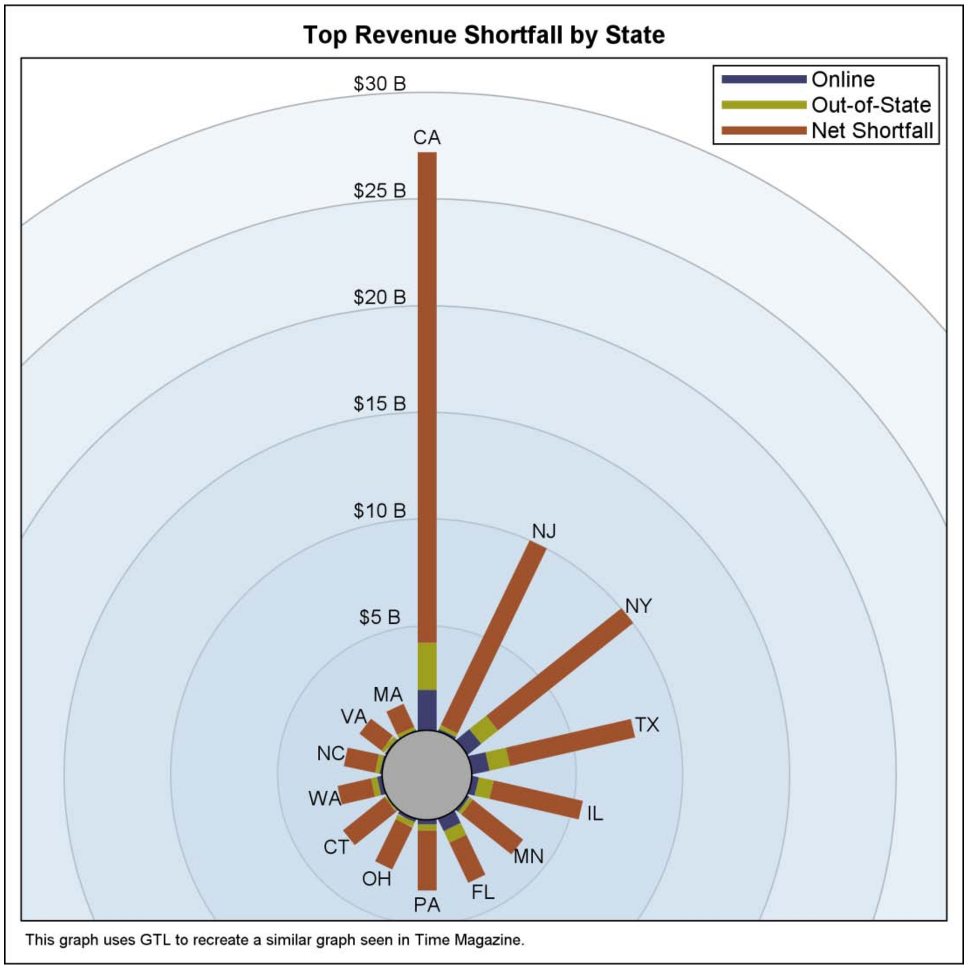


Figure 3: Spiral Bar Chart Showing Revenue Shortfall

Let us now dive into the details of this implementation.

DATA PREPARATION

The data here is approximate and for illustration-purposes only. We start with the raw numbers 'eyeballed' from the original graph. Here is a snippet of the DATA step:

```
data state_online;
  input State $1-10 stateAbbr $15-20 Case $21-35 Amount;
  format Amount dollar6.0;
  label amount='Amount in Billions';
  datalines;
California    CA    Online          2.0
California    CA    Out-of-State    2.2
California    CA    Net Shortfall   23.0
New Jersey    NJ    Online          0.2
New Jersey    NJ    Out-of-State    0.2
New Jersey    NJ    Net Shortfall   9.6
...
```

Our goal here is to draw the bar chart in a radial fashion, in polar coordinates. A vector plot without arrow heads lends itself well to drawing angled bars.

We will first compute the vector's radial length and an equidistant angle for each state. We will use a grouped vector plot with the CASE column as our group. Each vector tags on to the end of the previous group value, in the same order as shown in the DATA step above. Let us assume we want to plot fourteen states and offset the bars from the pole on a circle of radius = 2, with the bars being equally spread out over 2π radians.

```
%let numTopStates = 14;
%let baseRadius = 2.0;

/* Compute vector end points prev, this and angle theta */
data state1;
  retain theta 0 this &baseRadius prev;
  set state_online (obs= %eval(3 * &numTopStates));

  if mod(_N_, 3) NE 0 then stateAbbr="";

  prev = this;
  this = prev + amount;
  output;

%let PI = %sysfunc(constant(PI));
  if mod( N , 3) = 0 then do;
    theta = theta + 2 * &PI / &numTopStates;
    this = &baseRadius;
  end;

run;
```

Next, we convert the polar coordinate data into Cartesian coordinates to allow us to draw the graph in GTL. In this step, we project the data into X and Y values using the cosine and sine functions, respectively. The columns X1, Y1 are the origin of the vectors and X2, Y2 are the end points.

```
data revenue;
  set state1 end=last;
  length circLabel $7;
  keep state case x1 y1 x2 y2 stLabelX stLabelY cLabelX
                              stateAbbr radius circLabel;

  xProject = cos(theta); yProject = sin(theta);

  x1=prev*xProject; /* vector origin */
  y1=prev*yProject;

  x2=this*xProject;  /* vector end */
  y2=this*yProject;

  /* State labels at the end of each 'bar' */
%let labelOffset = 0.7;
  stLabelX=(this + &labelOffset)*xProject;
  stLabelY=(this + &labelOffset)*yProject;

  radius = &baseRadius;
  output;

  ...
```

We also calculate the positions for the abbreviated state labels at the end of each bar as STLABELX, STLABELY, after adding a small offset from the end value.

Continuing with this DATA step, we generate some 'reference circles', akin to reference lines in a Cartesian plot, at the end of the DATA step. The RADIUS column contains the radii for these circles. The X coordinate for the reference labels, CLABELX, is fixed at -2.2, slightly to the left of center, whereas the Y coordinate can be slightly offset from the radius value itself. The label text itself is generated from the radius value, with 'B' suffix (for billions).

```
...

  /* Reference circles */
  if last=1 then
  do;
    state=''; stateAbbr=''; case='';
    x1=.; y1=.; x2=.; y2=.; stLabelX=.; stLabelY=.;
    do radius=7 to 32 by 5;
      clabelX=-2.2;
      circLabel=put(radius - &baseRadius, 2.);
      circLabel=cat('$', strip(cat(strip(circLabel),' B')));
      output;
    end;
  end;
...
```

This is all the data we need to draw the graph.

TEMPLATE CODE

It is best to use a layout that uses the same rendered scale for the X and Y coordinates when displaying a polar coordinate visual. Hence, we start with an OVERLAYEQUATED layout and suppress the axis displays as they are not needed here, as shown in the template code snippet below.

First, we put down our reference circles, using parametric ellipses with their SEMIMAJOR and SEMIMINOR parameters set to the RADIUS column. As a bonus, these overlapping circles also give us a stepped color effect due to the transparencies being used.

The ‘bars’ are specified next as a vector plot with the X1, Y1 columns as their origin and X2, Y2 columns for their ends. The second parametric ellipse that follows the vector plot adds a ‘cap’ to the center of the plot to smooth out the ends of the bars.

The next two scatter plots add the state labels at the end of the bars and the labels for the reference circles. The discrete legend references the vector plot to complete the template.

```
...
  layout overlayEquated / xaxisopts=(display=none offsetmin=0 offsetmax=0)
                          yaxisopts=(display=none offsetmin=0 offsetmax=0.03);

    ellipseParm semimajor=radius semiminor=radius slope=0 xorigin=0 yorigin=0 /
                          display=all clip=true datatransparency=0.7
                          fillAttrs=(transparency=0.8);
    vectorPlot x=y2 y=x2 xorigin=y1 yorigin=x1 / group=case
                          arrowheads=false name='v' includeMissingGroup=false
                          lineattrs=(pattern=solid thickness=12);
    ellipseParm semimajor=2.1 semiminor=2.1 slope=0 xorigin=0 yorigin=0 /
                          display=all fillAttrs=(color=darkGray);

    scatterPlot x=stLabelY y=stLabelX / markercharacter=stateAbbr ... ;
    scatterPlot x=cLabelX y=eval(radius + 0.4) / markercharacter=circLabel ... ;

    referenceLine y=-6.8 / datatransparency=1;

    discreteLegend 'v' / location=inside valign=top halign=right across=1 ... ;

  endLayout;
...
```

Notice the reference line lurking just before the discrete legend. Its purpose is purely to nudge the graph’s minimum Y range below the lowest bar label (‘PA’ in this case). Without this, the label can be clipped due to the fact that we have explicitly set the Y axis OFFSETMIN option.

We have used a custom style to change the first three group colors shades of red, yellow, and blue. All the pieces are now in place and the graph is rendered using PROC SGRENDER code similar to the previous graph.

TWEAKED VERSION

The above presentation can be improved by adding a 'skin' effect to the vectors. This is done by overlaying several vectors with decreasing thicknesses and lighter colors. A shopping cart icon has been placed at the origin of the bars using the DRAWIMAGE statement. Adding a text box using DRAWTEXT statements to explain the analysis adds the final touch.

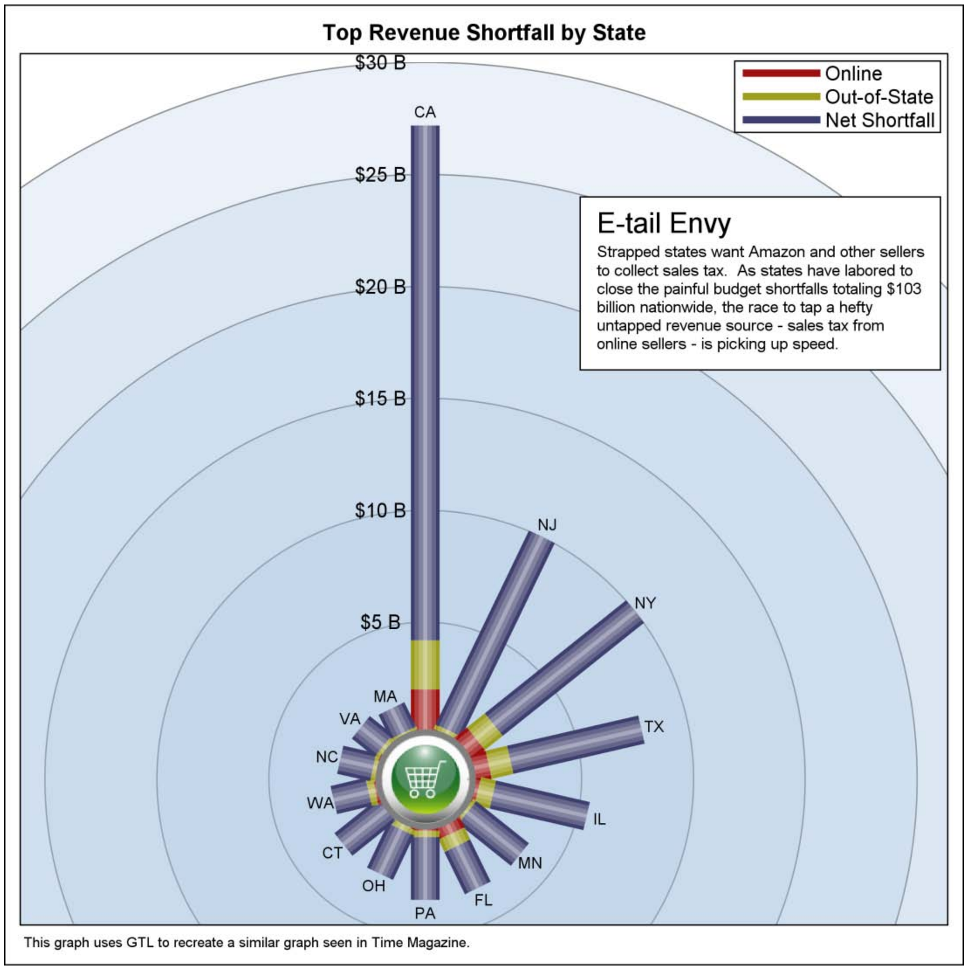


Figure 4: Spiral Bar Chart with Simulated Skin

This graph now has an *Infographics* feel to it, which demonstrates one more facet of GTL's capabilities.

CONDITIONAL REGRESSION TREE

Next, we will show how we can use GTL to display tree-structured graphs. Conditional trees show the binary recursive partitions used for statistical inference. Many of the examples on the Web were created using *R*.

The example below is a regression tree (box plots for leaf nodes) created with GTL. Tree vertices and edges need to be explicitly positioned, but the leaf nodes with box plot fit nicely into a lattice layout. Let us proceed to the data preparation for the graph.

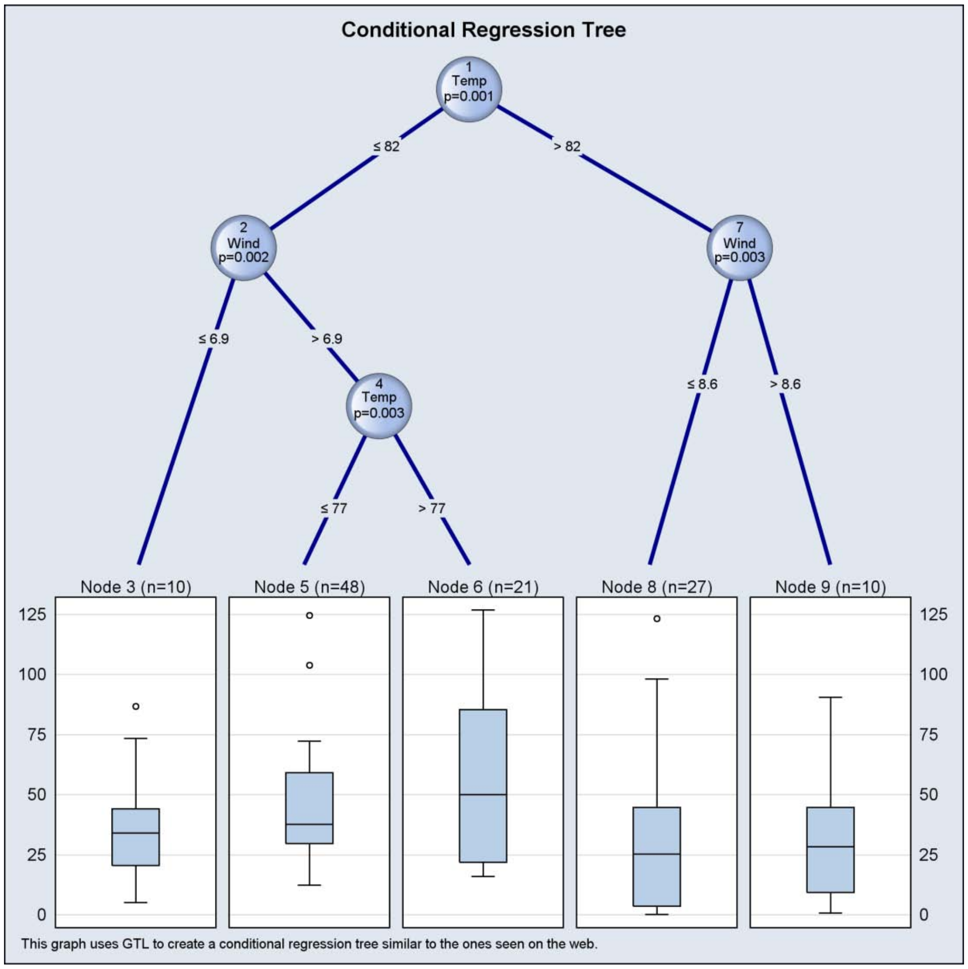


Figure 5: A Conditional Regression Plot

.

DATA PREPARATION

Assuming that we know the regression tree structure, we first generate the positions and size for the nodes as the nodes data set and the node labels as the nodeLabels data set.

In this process, it helps to visualize plotting these inside a rectangle with an approximate width of 5 and height of 25. This gives us an X range of [0, 5] and y range of [0, 25], with the leaf nodes being toward the Y = 0 side. The choice of these ranges is not crucial, but a larger Y range allows us to position the numerous labels and nodes vertically in larger steps than if the range were small.

```
data nodes;
 input xvar yvar size nodeId $3.;
 datalines;
3.0  21 30 1
1.5  14 30 2
4.8  14 30 7
2.4   7 30 4
;
run;

data nodeLabels;
 input xscat yscat label1 $15.;
 datalines;
3.0  22.0 1
3.0  21.4 Temp
3.0  20.7 p=0.001
1.5  14.9 2
1.5  14.2 Wind
1.5  13.7 p=0.002
4.8  14.9 7
4.8  14.2 Wind
4.8  13.7 p=0.003
2.4   8.0 4
2.4   7.4 Temp
2.4   6.7 p=0.003
;
run;
```

Next, we generate the position of the edges as the edges data set:

```
data edges;
 input x y xor yor;
 datalines;
0.8   0 1.5  14
1.5  14 3.0  21
2.4   7 1.5  14
1.9   0 2.4   7
3     0 2.4   7
4.8  14 3.0  21
4.2   0 4.8  14
5.4   0 4.8  14
;
run;
```

Lastly, we merge the box variable data with all the above data sets. We are using randomly generated data as the box data set for this example.

```
data box;
  do i=1 to 20;
    y1=abs(rannor(9273)*50);
    y2=abs(rannor(9273)*50);
    y3=abs(rannor(9273)*50);
    y4=abs(rannor(9273)*50);
    y5=abs(rannor(9273)*50);
    output;
  end;
run;

data cr_tree;
  merge nodes nodeLabels edges box;
run;
```

TEMPLATE CODE

The conditional regression tree can be thought of as two cells arranged in a single column. The top cell contains the tree part and the bottom cell has the box plot leaf nodes. The ratio of top cell to the bottom cell can be adjusted with the ROWWEIGHTS option, and here we have used a ratio of 3:2 to give enough space for the tree part.

Let us now inspect the top cell. We have set the VIEWMIN and VIEWMAX options for the axes to help register the tree with the box plots by trial and error. The edges are drawn first as a vector plot and the nodes are overlaid on them as a bubble plot. Reversing this order would expose the edges being drawn to the center of the node bubbles!

Next, we use the scatter plot to label the node with the node ID and other information, as shown below. The block for edge labels and the lattice of box plots will be discussed separately.

```
...

  layout lattice / rows=2 rowweights=(0.6 0.4);

    layout overlay / walldisplay=none
                    yaxisopts=(display=none offsetmin=0.02
                                            linearopts=(viewmin=0 viewmax=21))
                    xaxisopts=(display=none offsetMin=0 offsetmax=0
                                            linearopts=(viewmin=0 viewMax=6.2));
      vectorplot x=x y=y xorigin=xor yorigin=yor / arrowheads=false ... ;
      bubbleplot x=xvar y=yvar size=size /  display=(outline fill)
                       bubbleRadiusMin=24 bubbleRadiusMax=28 dataSkin=gloss ... ;
      scatterplot x=xscat y=yscat / markercharacter=label1 ... ;

      /* << Draw text statements go here >> */

    endlayout;

    /* << Lattice layout with box plots goes here >> */

  endLayout;
...
```

We could have used a scatter plot to label the edges as well, but if you prefer to use Unicode for the symbol '≤', we cannot use column-based text data. In SAS 9.3, GTL supports layout-based draw statements that support in-line text data, similar to the ENTRY statements. These draw statements are similar to the annotate facility in traditional SAS/GRAPH, but give you more control.

Here we have used the DRAWTEXT statement to place the edge labels. This statement has a fully transparent background by default. By setting the transparency to zero (opaque), we get a 'wipe out' effect underneath the edge labels.

```
...

%let unicodeLTEq = {unicode '2264'x}; /* Unicode <= character */
%macro drawTextStmt(symbol, txt, x=, y=);
  drawText textAttrs=(size=7) &symbol &txt / x=&x y=&y
            drawSpace=datavalue backGroundAttrs=(transparency=0);
%mend drawTextStmt;

      %drawTextStmt(&unicodeLTEq, " 82", x=2.45, y=18.5);
      %drawTextStmt(">",          " 82", x=3.65, y=18.5);

      %drawTextStmt(&unicodeLTEq, " 6.9", x=1.30, y=10);
      %drawTextStmt(">",          " 6.9", x=2.05, y=10);

      %drawTextStmt(&unicodeLTEq, " 8.6", x=4.55, y=8);
      %drawTextStmt(">",          " 8.6", x=5.1,  y=8);

      %drawTextStmt(&unicodeLTEq, " 77", x=2.1, y=2.5);
      %drawTextStmt(">",          " 77", x=2.75, y=2.5);
...
```

The last piece in this template puzzle is the bottom cell of the lattice layout containing the box plot leaf nodes. We use a five-column lattice layout, each containing a box plot. The leaf node ID and labels have been implemented as COLUMN2HEADERS, as shown below.

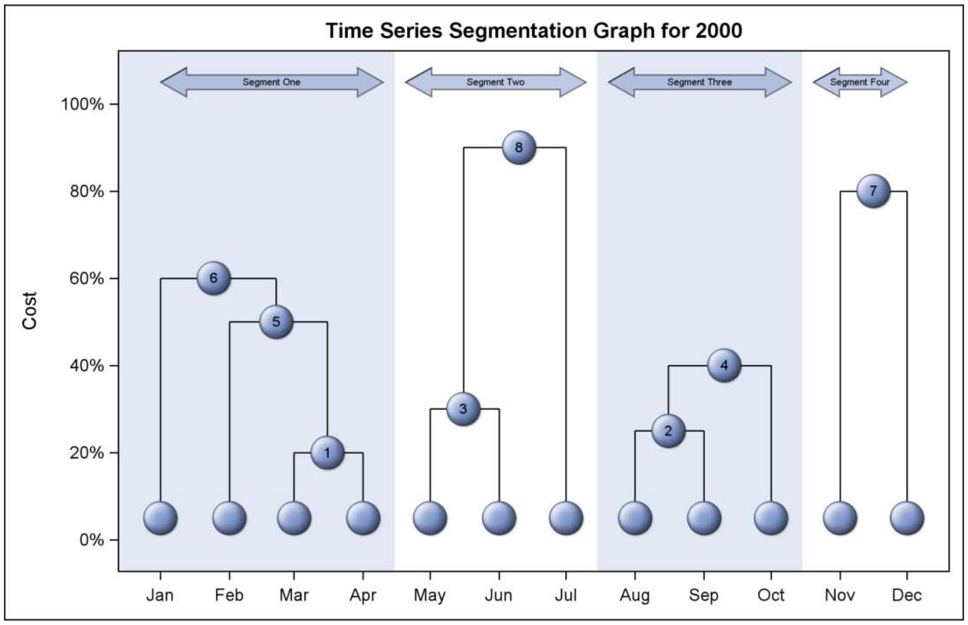
```
...
  layout lattice / columns=5 rowDataRange=union columnGutter=10;
    column2Headers;
      entry "Node 3 (n=10)";
      entry "Node 5 (n=48)";
      entry "Node 6 (n=21)";
      entry "Node 8 (n=27)";
      entry "Node 9 (n=10)";
    endColumn2Headers;

    rowAxes;
      rowAxis / display=(tickValues) displaySecondary=(tickValues)
                                                   gridDisplay=on;
    endRowAxes;

%let boxDisplay=(caps fill median outliers);
      boxplot y=y1 / display=&boxDisplay ;
      boxplot y=y2 / display=&boxDisplay ;
      boxplot y=y3 / display=&boxDisplay ;
      boxplot y=y4 / display=&boxDisplay ;
      boxplot y=y5 / display=&boxDisplay ;
    endLayout;
  endLayout;

...
```

With this, the template for the conditional regression tree is complete. The graph is generated using the SGRENDER procedure as shown in the earlier sections.

TIME SERIES SEGMENTED TREE

This is a type of visualization for presenting similarity of time series segments. The graph displays several time period (regime) segments, and the time series is clustered within those segments using dendrogram-like trees.

Off the Beaten Path: Create Unusual Graphs with GTL, continued



Let us look at the data and template for this graph.

DATA PREPARATION

First we create the tree data:

```
data TimeSeriesTree;
  length Name $3 Label $13;
  format Date x xo xl xh xlabel date9.;
  format Cost percent.;
  retain id 1;
  x=.; y=.; xo=.; yo=.;

  /* Leaf nodes at x=date, y=cost */
  do date='01Jan2000'd, '01Feb2000'd, '01Mar2000'd, '01Apr2000'd,
          '01May2000'd, '01Jun2000'd, '01Jul2000'd, '01Aug2000'd,
          '01Sep2000'd, '01Oct2000'd, '01Nov2000'd, '01Dec2000'd;
    Cost=0.05; Size=1; Name=' ';
    output;
    id = id + 1;
  end;

  /* Each non-leaf node is generated. Its left and right links are created as an
   * inverted ‘U’ with 3 segments.
   */
  id=13; name='1'; date='16Mar2000'd; cost=0.20;
  output;
  id=.;  name='V1'; x='01Mar2000'd; y=0.2; xo=x; yo=0.05; cost=.;
  output;
  id=.;  name='V2'; x='01Apr2000'd; y=0.2; xo=x; yo=0.05; cost=.;
  output;
  id=.;  name='V3'; x='01Mar2000'd; y=0.2; xo='01Apr2000'd; yo=0.2; cost=.;
  output;

  ...

  /* Left and right extents for double-ended arrows */
  id=.; y2=1.05; xl='01Jan2000'd; xh='10Apr2000'd; label='Segment One';
  xlabel=(xl+xh)/2; /* Label for arrows */
  output;
  ...

run;
```

The above DATA step creates the positions and labels for each node. It also creates the node links composed of three segments in the shape of an inverted ‘U’. The last few lines are for horizontal double

Then we create the regime data and merge the two data sets:

```
data regimes;
  input date2 regime;
  informat date2 date.;
  format date2 date.;
  datalines;
'01Jan2000'd  1
'15Apr2000'd  1
'15Apr2000'd  2
'15Jul2000'd  2
'15Jul2000'd  3
'15Oct2000'd  3
'15Oct2000'd  4
'01Dec2000'd  4
;
run;

data merged;
  set timeseriestree regimes;
run;
```

TEMPLATE CODE

The template for this graph is rather simple compared to the previous conditional regression tree example. We have an overlay layout with a block plot for demarcating the regimes. A high-low chart with filled arrow end caps is used to draw the arrows at the top. These are then labeled using a scatter plot.

The dendrogram-like tree is rendered by using a vector plot to draw the node links (as inverted ‘U’s) and overlaying a bubble plot on top for the nodes. This is similar to the previous example. The nodes are once again labeled using a scatter plot.

```
...
  layout overlay / xaxisopts=(label='Time' offsetmax=0.05 offsetmin=0.05
                              display=(ticks tickvalues)
                              timeopts=(splittickvalue=false interval=month
                                        tickvalueformat=monname3.))
        yaxisopts=(label='Cost' linearopts=(viewmin=0 tickvalueformat=percent.));

    blockplot x=date2 block=regime / display=(fill) filltype=alternate
                        altfillattrs=(color=white) datatransparency=0.3;

    highlowplot y=y2 low=xl high=xh / type=bar intervalbarwidth=11
                highcap=filledarrow lowcap=filledarrow datatransparency=0.5 ... ;
    scatterplot y=y2 x=xlabel/ markercharacter=label ... ;

    vectorplot x=x y=y xorigin=xo yorigin=yo / arrowheads=false;
    bubbleplot x=date y=cost size=size / bubbleradiusmin=12px ... ;
    scatterplot x=date y=cost / markercharacter=name;

  endlayout;
...
```

Now that we have the data and the template, we can generate the time series segmentation graph using the SGRENDER procedure as shown earlier.

MAP MAGIC

GTL does not have standard map support in SAS 9.3. But for simple outline maps, you can get the desired visual using projected data and series plots. Shown below is a graph created using GTL. The graph shows the populations of the four largest cities in the USA on a map. An inset provides the ethnicity breakdown for each of the four cities. Note that the states of Alaska and Hawaii have also been displayed on the map. Note that this example requires SAS/GRAPH for the data and projection.

Let us now see how this was achieved in GTL.

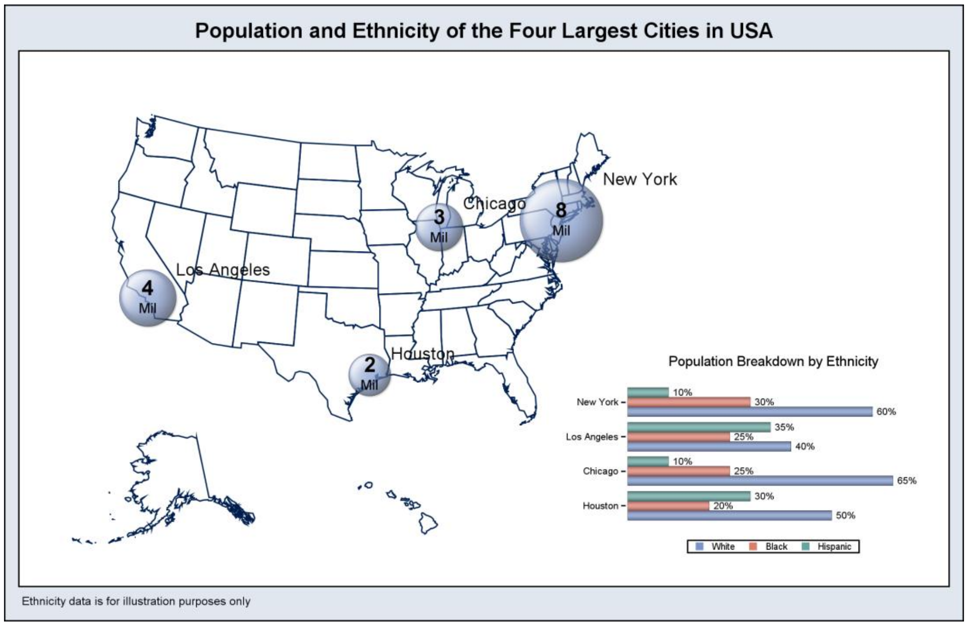


Figure 6: A GTL Map Using Series Plot

DATA PREPARATION

We start with the MAPS.STATES data set that comes with SAS/GRAPH. Since we will be using a series plot to draw the map outline, we need to close the map polygons by repeating the first observation for each segment (state). We have selected a lower level of detail (density ≤ 1) as that is sufficient for our purpose here. Puerto Rico has been omitted to allow space for the bar chart inset.

In order to reposition Alaska and Hawaii, we have split the data set into three data sets as shown below:

```
data usa alaska hawaii;
  /* Use less detailed data; remove Puerto Rico to allow barchart inset */
  set maps.states( where=(( density <= 1) and (state ne 72)) );
  length id $8;
  retain xFirst yFirst;
  keep state segment x y id;
  by state segment;

  /* Make unique Id for each state+segment combination */
  id=put(state, 3.0) || put(segment, 3.0);

  /* Save away the first vertex */
  if first.segment=1 then do;
    xFirst=x; yFirst=y;
  end;

  if state=2 then output alaska;
  else if state=15 then output hawaii;
  else output usa;

  /* Close the segment polygon: write first vertex again */
  if last.segment=1 then do;
    x=xFirst; y=yFirst;
    if state=2 then output alaska;
    else if state=15 then output hawaii;
    else output usa;
  end;

run;
```

Next, we append the information about the four cities to the end of our USA data set:

```
data usa_map_pop;
  set usa end=last;
  drop deg2Rad lat long;
  format pop 1.0;
  length city $12;
  deg2Rad=atan(1)/45; /* Degree to radian conversion factor */
  output;

  /* Append obs for city population values to USA map data */

  if last=1 then do; /* Convert lat,long from deg,min --> x, y radians */
    segment=1;
    state=102; City='New York';   lat=40+47/60; long=73+58/60;
        x=long*deg2Rad; y=lat*deg2Rad; pop=8.1; output;
    state=103; City='Houston';    lat=29+45/60; long=95+31/60;
        x=long*deg2Rad; y=lat*deg2Rad; pop=2.1;  output;
    state=104; City='Chicago';    lat=41+50/60; long=87+37/60;
        x=long*deg2Rad; y=lat*deg2Rad; pop=2.7;  output;
    state=105; City='Los Angeles';lat=34+3/60;  long=118+15/60;
        x=long*deg2Rad; y=lat*deg2Rad; pop=3.8;  output;
  end;

run;
```

Our original data set MAPS.STATES had unprojected coordinates in radians. We now project the data sets into Cartesian coordinates using the SAS/GRAPH GPROJECT procedure with the default projection (Albers):

```
/*--Project each map separately--*/
proc gproject data=usa_map_pop out=usap;
  id state;
run;
proc gproject data=alaska out=alaskap;
  id state;
run;
proc gproject data=hawaii out=hawaiip;
  id state;
run;
```

Next, we generate the ethnicity data. Note that this data is for illustration purposes only.

```
data race;
  input City2 $1-12 Race $ Value;
  format Value percent.;
  datalines;
New York     White       0.6
Los Angeles  White       0.4
Chicago      White       0.65
Houston      White       0.5
New York     Black       0.3
Los Angeles  Black       0.25
Chicago      Black       0.25
Houston      Black       0.2
New York     Hispanic    0.1
Los Angeles  Hispanic    0.35
Chicago      Hispanic    0.1
Houston      Hispanic    0.3
;
run;
```

In the final steps, we merge the ethnicity data with the map data after renaming the X, Y and ID columns for Alaska and Hawaii to new columns. We then separate out the city population data coordinates we added earlier to different columns. We also add a scale label of ‘Mil’ for labeling the population values. This completes the data preparation phase.

```
data merged;
  merge usap
        alaskap(rename=(x=ax y=ay state=astate segment=aseg id=aid))
        hawaiip(rename=(x=hx y=hy state=hstate segment=hseg id=hid))
        race;
run;

/* Separate the city population data coord columns from the map coord */
data mergedFinal;
  set merged;
  if city ne ' ' then do;
    xc=x; yc=y; x=.; y=.; segment=.; state=.;
  scale='Mil';
  end;
run;
```

We will go over the template next.

TEMPLATE CODE

The best way to render map data is with an equated overlay layout because it uses the same ‘data to drawing scale’ for both X and Y axes. A series plot is used to draw the map outlines with the projected X and Y columns and group set to the ID column. We then overlay filled circles centered on the city coordinates, sized according to their population values using a bubble plot. The bubbles are then labeled with their population values using two scatter plots. Note that the Y values of the scatter plots have been adjusted around the center to allow for a two-line label.

An interesting feature of this template is the use of nested overlays. As you can see below, the insets for Alaska and Hawaii have been created by two nested equated overlay layouts. Each overlay contains a series plot with specific data for its state, similar to the parent layout.

Note that these nested overlays have been explicitly sized using the WIDTH and HEIGHT options specified as percentages of the output size. These values have to be set with some care to get the relative dimensions of the states to look right. We position these nested layouts inside the parent layout using the numeric forms of the HALIGN and VALIGN options.

```
...
  /* << Legend items go here >> */

  layout overlayEquated / xAxisOpts=(offsetMax=0.3 display=none)
                              yAxisOpts=(offsetMin=0.3 display=none);
    seriesPlot x=x y=y / group=id ... ;
    bubblePlot x=xc y=yc size=pop / dataTransparency=0.3  dataLabel=city
                                 bubbleRadiusMin=15 bubbleRadiusMax=30 ...;
    scatterPlot x=xc y=eval(yc + 0.015) / markerCharacter=pop ... ;
    scatterPlot x=xc y=eval(yc - 0.015) / markerCharacter=scale ... ;

    /* Inset Alaska */
    layout overlayEquated / xAxisOpts=(display=none) yAxisOpts=(display=none)
                            width=25pct height=20pct hAlign=0.05 vAlign=0.1
                            wallDisplay=none opaque=false;
      seriesPlot x=ax y=ay / group=aid ... ;
    endLayout;

    /* Inset Hawaii */
    layout overlayEquated / xAxisOpts=(display=none) yAxisOpts=(display=none)
                            width=10pct height=10pct hAlign=0.4 vAlign=0.1
                            wallDisplay=none  opaque=false;
      seriesPlot x=hx y=hy / group=hid ... ;
    endLayout;

  /* << Inset bar chart layout >> */

  endLayout;
...
```

The last bit of template code is for the inset bar chart showing the ethnicity data. Once again, we employ a nested overlay layout for this. A simple overlay suffices here since we do not require the coordinate drawing scales for a bar chart to be equal. We have used the clustered group display in the bar chart for easy comparison between the group values.

```
...
  layout overlay / width=40pct height=30pct hAlign=0.95 vAlign=0.1
                   wallDisplay=none xAxisOpts=(display=none)
                   yAxisOpts=(reverse=true display=(tickValues) ... ) ;

    entry textAttrs=(size=7) "Population Breakdown by Ethnicity" /
                                            location=outside vAlign=top;
    barChart x=city2 y=value / group=race orient=horizontal name='a'
                    groupDisplay=cluster includeMissingGroup=false
                   barLabel=true dataSkin=pressed ... ;
    discreteLegend ... ;

  endLayout;
...
```

Both data and the template for this graph are now complete. The SGRENDER procedure is then used to render this graph.

A TRANSIT MAP FOR THE MILKY WAY

Are you lost in your galactic travels? Samuel Arbesman came up with a stylized map for our Milky Way galaxy, akin to the ubiquitous subway transit maps. (See Arbesman, S. in the References section.) The map shows different transit 'lines' with stations and terminals. It also has transfer stations between the different transit lines. Note that this stellar map has been augmented with some definitely terrestrial locations, just to see if you are paying attention!

The GTL implementation of this map is shown below:

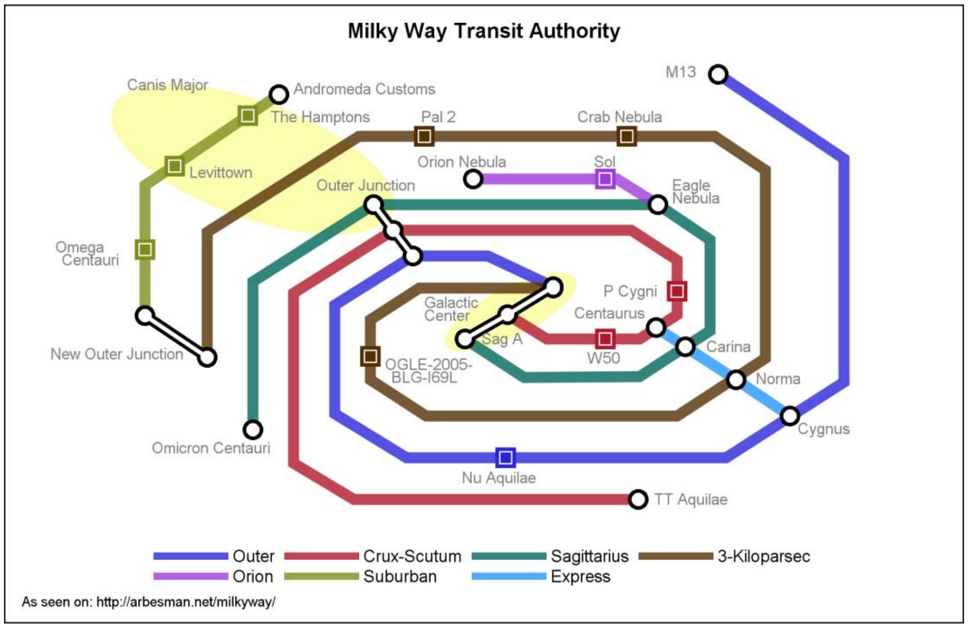


Figure 7: Milky Way Transit Map

Next, we will go over the data and the template needed for this visual.

DATA PREPARATION

First we create the transit lines with columns X and Y for vertex positions on the transit line. The ROUTE column tags vertices belonging to the same line with their route name. The TERMINUS column is set to 1 for stations that are terminals. The STATIONGRP column marks the vertices that are transfer stations with common non-missing values.

```
data mw_routes;
    input route $14. x y stationgrp terminus;
    datalines;
Outer          415     16  .     1
Outer          492     95  .     .
Outer          492    304  .     .
Outer          420    375  .     .
Outer          224    375  .     .
Outer          180    334  .     .
Outer          180    232  .     .
Outer          228    186  3     1
Outer          274    186  .     .
Outer          314    215  4     1
Crux-Scutum    366    414  .     1
Crux-Scutum    192    414  .     .
Crux-Scutum    155    380  .     .
Crux-Scutum    155    221  .     .
Crux-Scutum    216    162  3     1
Crux-Scutum    362    162  .     .
Crux-Scutum    390    190  .     .
Crux-Scutum    390    241  .     .
Crux-Scutum    367    264  .     .
Crux-Scutum    310    264  .     .
Crux-Scutum    286    241  4     1
...
```

Next we generate the positions and text for all the labels in the DATA step below:

```
data mw_names;
    input xn yn name $ 13-31 ;
    datalines;
152      30  Andromeda Customs
138      56  The Hamptons
 88     108  Levittown
  6     178  Omega
 10     190  Centauri
  3     278  New Outer Junction
 65     366  Omicron Centauri
...
```

We now get to stations that are neither terminals or transfer stations (sub stations). These are grouped according to the transit line they belong to. Notice that we have an observation for SUBSTATION_GRP = 3 with missing XS and YS. This is because there are no substations on this line (Sagittarius) and this serves as a place-holder so that the group colors stay in sync.

```
data mw_substations;
  input xs ys substation_grp ;
  datalines;
285     375  1
390     219  2
346     264  2
  .       .  3
235      74  4
```

The last part of the data generation is to merge the above three data sets. We also reverse the Y coordinate values to account for the fact that our data generation assumed the coordinate origin to be at the top-left, whereas GTL uses a bottom-left origin.

```
data mw_merged;
    merge mw_routes mw_names mw_substations;
    y = 430 - y;
    if (yn ne .) then
      yn = 430 - yn;
    if (ys ne .) then
      ys = 430 - ys;
run;
```

We will inspect the template next.

TEMPLATE CODE

The template has a single layout overlay with the wall and both X and Y axis displays turned off. We have used two parametric ellipses to show the galaxy center and Canis Major constellation. The series plot with columns X, Y for position and ROUTE as the group is overlaid next. This forms the transit lines, which we also reference from the discrete legend, as shown below:

```
...
    layout overlay / walldisplay=none yaxisopts=(display=none)
        xaxisopts=(display=none offsetmax=0.1);

      /* Galactic Center */
      ellipseparm semimajor=51 semiminor=20 slope=.88 xorigin=288 yorigin=191 /
          display=(fill) fillAttrs=(color=cxFDFDB0);
      /* Canis Major */
      ellipseparm semimajor=100 semiMinor=50 slope=-.70 xorigin=130 yorigin=334 /
          display=(fill) fillattrs=(color=cxFDFDB0);

      /* Main routes */
      seriesplot x=x y=y / name="routes" includemissinggroup=false
          group=route lineattrs=(thickness=1.8% pattern=solid)
          dataTransparency=0.2;

      /* << Transfer stations, Terminals and Sub-Stations go here >> */

      discretelegend "routes" / border=false;

    endlayout;
...
```

We will go over the details of the terminals, transfer stations and sub stations next. In order to get the connected outline effect on the transfer stations, we draw the transfer lines as a series plot with thick lines and a scatter plot with larger markers. We then overlay a slightly thinner series plot and a scatter plot with smaller markers in the background color to ‘hollow out’ these features, leaving us with an outline effect. This is shown below:

```
...
      /* Transfer outline */
      seriesplot x=x y=y / group=stationgrp includemissinggroup=false display=all
          lineattrs=(thickness=1.8% pattern=solid color=black)
          markerAttrs=(size=3.0% symbol=circlefilled color=black);
      /* Terminal outline */
      scatterplot x=x y=y / group=terminus includemissinggroup=false
          markerattrs=(size=3.0% symbol=circlefilled color=black);

%let bgColor=white;
      /* Transfer fill */
      seriesplot x=x y=y / group=stationgrp includemissinggroup=false
          lineattrs=(thickness=0.8% pattern=solid color=&bgColor);
      /* Terminal fill */
      scatterplot x=x y=y / group=terminus includemissinggroup=false
          markerattrs=(size=2.0% symbol=circlefilled color=&bgColor);

...
```

Lastly we draw the sub stations using scatter plots and then place all the labels using a scatter plot.

```
...

      /* Substations fill */
      scatterplot x=xs y=ys / group=substation_grp includemissinggroup=false
          markerattrs=(size=3.0% symbol=squareFilled);
      /* Substations inlay */
      scatterplot x=xs y=ys / group=substation_grp includemissinggroup=false
          markerattrs=(size=2.0% symbol=square color=&bgColor);

      /* Terminal/Sub Labels */
      scatterplot x=xn y=yn / group=name includemissinggroup=false
          markerattrs=(size=0) datalabelattrs=(color=grey size=8)
          datalabel=name datalabelposition=right;
...
```

Note that we first draw the sub stations with filled markers (SQAUREFILLED) and then overlay smaller SQUARE markers to achieve the inlay effect. One of the drawbacks of this trick of overlaying markers and lines is that we cannot get the same superimposed representation in the discrete legend. But since we have labeled the markers and have a legend for the series plot transit lines, we can do without a legend for the sub stations.

We render this graph the same way as the previous graphs, using the SGRENDER procedure.

PARALLEL AXES PLOT

Parallel axes plot (or parallel coordinates plot) is a good visualization for detecting patterns in multivariate data. After seeing the previous examples in this presentation, it should come as no surprise to see a parallel axis plot implemented in GTL!

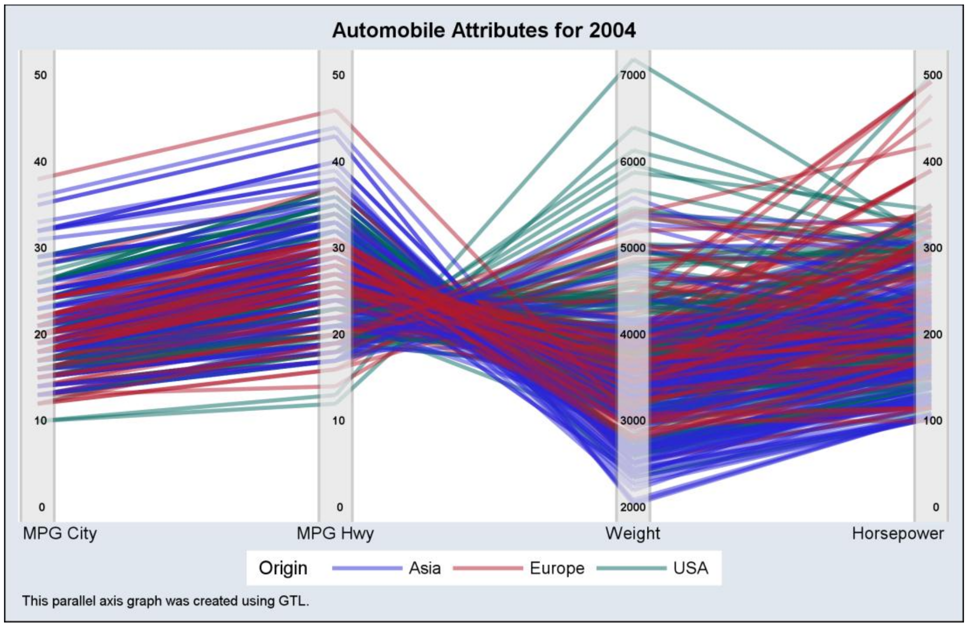


Figure 8: A Parallel Axis Plot for Year 2004 Automobiles

The above graph shows how automobiles in 2004 varied in their miles per gallon for city and highway, weight and horsepower values, depending on their origin. We have omitted vehicles using hybrid technology from this example.

Let us proceed to the data and template for this graph.

DATA PREPARATION

We will use the SASHELP.CARS data set as our starting point and we will look at four columns: MPG_CITY, MPG_HIGHWAY, WEIGHT and HORSEPOWER, for this example. This graph simulates the Y axes (as you will see in the template section), so we cannot rely on the built-in support for generating the axis tick values. Instead, we need to know the data ranges for our columns of interest. Here we have set the minimum and maximum values for each column after some experimentation, starting from their true ranges. We create macro variables for the min, max and range values for each column, as shown below:

```
/* Set min, max for each variable */
%let min_v1 = 0;    %let max_v1 = 50;   /* mpg_city */
%let min_v2 = 0;    %let max_v2 = 50;   /* mpg_highway */
%let min_v3 = 2000; %let max_v3 = 7000; /* weight */
%let min_v4 = 0;    %let max_v4 = 500;  /* horsepower */

/* Calculate the ranges */
%let range_v1 = %sysevalf(&max_v1 - &min_v1);
%let range_v2 = %sysevalf(&max_v2 - &min_v2);
%let range_v3 = %sysevalf(&max_v3 - &min_v3);
%let range_v4 = %sysevalf(&max_v4 - &min_v4);
```

Next, we filter out cars of TYPE='Hybrid' from our input data set. We setup up four X axis coordinates X1 to X4, one for each parallel axis column and also create new normalized [0, 1] columns for these columns.

```
/* Normalize the values */
data par_axis_norm;
  set sashelp.cars(where=(type ne 'Hybrid'));
  x1 = 1; x2 = 2; x3 = 3; x4 = 4;
  mpg_city_pct = (mpg_city - &min_v1) / &range_v1;
  mpg_highway_pct = (mpg_highway - &min_v2) / &range_v2;
  weight_pct = (weight - &min_v3) / &range_v3;
  horsepower_pct = (horsepower - &min_v4) / &range_v4;
run;
```

We now calculate the coordinates for the Y axis tick values for each of the columns. Note that AXIS_X stays constant for a given column while AXIS_Y varies from 0 to 1 in steps of 0.2 to give us five intervals. The actual tick value to label this coordinate is obtained by linearly interpolating the column values from its min to max.

```
data y_axis_ticks;
  drop i increment;
  attrib tvalue format=Best4.;
  axis_x = 1; axis_y = 0;
  tvalue=&min v1;
  increment = 0.2 * &range_v1;
  output;
  do i=1 to 5;
    axis_y = axis_y + 0.2;
    tvalue = tvalue + increment;
    output;
  end;

  /* << Repeat for other columns >> */
  ...
run;
```

Lastly, we create the X coordinates and labels for the reference lines which simulate the axis lines and merge all the data sets:

```
/* X axis labels */
data x_axis_labels;
  input pos label $15.;
  datalines;
 1   MPG City
 2   MPG Hwy
 3   Weight
 4   Horsepower
;
run;

/* Final merged data */
data par_axis_final;
  merge par_axis_norm y_axis_ticks x_axis_labels;
run;
```

At this point we have all the data we need for the graph. Let us see how the template is structured next.

TEMPLATE CODE

The template for this graph is simpler than the data generation steps. We start with an overlay layout and turn off the X and Y axis displays. We then draw a set of lines between the first pair of parallel axes as a vector plot using the X1 and MPG_CITY_PCT as the origin and X1 and MPG_HIGHWAY_PCT as the end, grouped by the ORIGIN column. The arrow heads for the vector plot has been turned off and we also use a data transparency of 0.5 to enable us to see the clusters of lines on the parallel axes. We specify such a vector plot for remaining two pairs of axes.

To simulate Y axis lines, we overlay two thick reference lines with slightly different thicknesses to create an outlined bar effect. The curve label of second reference line functions as the X axis tick values.

As the last step, we label the Y axis tick values for the parallel axes using a scatter plot with AXIS_X and AXIS_Y columns as their X and Y coordinates with the MARKERCHARACTER option set to the TVALUE column.

```
...
layout overlay / wallDisplay=(fill)
                  xaxisOpts=(display=none offsetMin=0.02 offsetMax=0.02)
                  yAxisOpts=(display=none offsetMin=0.03 offsetMax=0.02);

%let vectorOpts=lineAttrs=(pattern=solid thickness=3) arrowHeads=false
                                         dataTransparency=0.5;
    vectorPlot xOrigin=x1 yOrigin=mpg_city_pct x=x2 y=mpg_highway_pct /
                                         group=origin  &vectorOpts;
    vectorPlot xOrigin=x2 yOrigin=mpg_highway_pct x=x3 y=weight_pct /
                                         group=origin  &vectorOpts;
    vectorPlot xOrigin=x3 yOrigin=weight_pct x=x4 y=horsepower_pct / name="p1"
                                         group=origin &vectorOpts;
     /* Y axis lines */
    referenceLine x=pos / lineAttrs=(color=gray thickness=26)
                                         dataTransparency=0.6 ;
    referenceLine x=pos / lineAttrs=(color=white thickness=22)
                  dataTransparency=0.4 curveLabel=label curveLabelPosition=min;

    /* Y axis tick values */
    scatterPlot x= axis_x y = axis_y / markerCharacter=tvalue
                                    markerCharacterAttrs=(size=6 weight=bold);

    discreteLegend "p1" / ... ;

endlayout;
...
```

This completes the template for the parallel axes plot. We render this graph the same way as the previous graphs, using the SGRENDER procedure.

CONCLUSION

The Graph Template Language in the ODS Graphics System offers you an extensive set of building blocks. You can combine these along with some data manipulation to generate graphs that might not, at first, appear possible in GTL.

When you break a graph down into its components, it is fairly easy to take GTL 'off the beaten path' to generate unusual graphs.

REFERENCES

- SAS Technical Papers. Available at http://support.sas.com/rnd/papers/index.html
- ExcelCharts Blog. Available at http://www.excelcharts.com/blog/beautiful-but-terrible-population-pyramids/
- Arbesman, S. "Milky Way Transit Authority". Available at http://arbesman.net/milkyway/

ACKNOWLEDGMENTS

This paper would not have been possible without the insight, ideas and support offered by Sanjay Matange. Thanks are also due to Rich Hogan (Population Pyramid), Debpriya Sarkar (Conditional Regression Tree) and Jason Smith (Milky Way Transit) for their SAS programs.

RECOMMENDED READING

- Cartier, Jeff. 2006. "A Programmer's Introduction to the Graphics Template Language." *Proceedings of the Thirty-first Annual SAS Users Group International Conference*. Cary, NC: SAS Institute Inc. Available at http://support.sas.com/events/sasglobalforum/previous/index.html.
- Kuhfeld, W. 2010. *Statistical Graphics in SAS: An Introduction to the Graph Template Language and the Statistical Graphics Procedures*, Cary NC: SAS Press.
- Matange, S. and Heath, D. 2011 *Statistical Graphics Procedures by Example: Effective Graphs Using SAS,* Cary NC: SAS Press.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Prashant Hebbar
SAS Institute Inc.
SAS Campus Dr.
Cary, NC 27513
E-mail: Prashant.Hebbar@sas.com