Paper 266-2012

Visualizing Spatial Data using SAS® and Google Static Maps

Jizhou Fu, NORC at the University of Chicago, Chicago, IL
Yanwei Zhang, CNA Insurance Company, Chicago, IL

ABSTRACT

We present a new tool that enables SAS users to overlay various spatial patterns on Google Static Maps in a straightforward manner. The tool relies on the graphical functionality from the DATA Step Graphical Interface, and is structured in a way that a variety of graphical forms or spatial information from different data sources can be combined effectively. We give an extensive discussion of the design of the tool and related background information, and provide detailed guidance on its use. Also, the usefulness and the flexibility of this tool are demonstrated through separate applications to survey field listing maps and crime distribution visualizations.

INTRODUCTION

Google Maps has become one of the most recognized and comfortable means of handling location data and providing mapping service. The popularity of Google Maps and its additional map-based services has helped many people become familiar with concepts like mapping, geocoding, routing and so on. Recent years have seen a growing interest in integrating Google Maps and statistical software to visualize spatial data in various applied statistical fields. The primary motivation is to enable end users of statistical software to utilize the comfortable and familiar geographic context Google Maps offers in order to improve the presentation or the analysis of spatial data. For example, Massengill (2010) presents a SAS Google Map Generator that will convert SAS/GRAPH map data sets and Annotate data sets to Google Maps polygons, and generate the HTML file and necessary Java code to display these on the Google Map; Drukenbrod and Mintz (2008) and Azimaee (2010) introduce approaches to writing Google Earth's Keyhole Markup Language (KML) within SAS so that SAS users can overlay their own data on Google Earth. While these methods provide solutions to displaying SAS spatial data on various Google maps, their reliance on other programming or markup languages such as Java and KML can increase the difficulty for SAS users not familiar with these languages, at least for the purpose of debugging, customizing or extending the existing functionalities.

Unlike the above tools in SAS, this paper presents a tool that integrates SAS and the Google Static Maps API, one component of the Google Maps API family. Google Static Maps API differs from Google Maps API in that it does not require JavaScript, thus providing a simpler way of interacting with Google Maps. One example of incorporating Google Static Maps in the open-source statistical package R can be found in Loecher (2010). Another distinct feature of the tool is that customized visualization products are created via SAS DATA Step Graphics Interface (DSGI), eliminating the need for users to understand the programming language from another domain.

With this toolkit, SAS users now possess an easy and efficient method to download the static map image containing the location of interest. Also, the tool implements a variety of customized spatial graphics, allowing users to overlay these graphics on previously retrieved images. Another important feature of this tool is that it can construct the desired figure as a sequence of layered images- one call of the tool generates one layer of the final image, and this layer of image can be imported as background image before the processing of another layer. Therefore, this process allows the final image to be built up gradually, layer by layer, and the plotting characteristics of different layers can possibly come from different data sources. As a result, the users have substantial control of the graphical appearance, and can produce sophisticated graphs in a straightforward and efficient manner.

In the following, we discuss the philosophy we adopted in designing the tool, and give a brief introduction to the Google Static Maps API and the DATA Step Graphics Interface. Then, two applications of the tool will be presented: the visualization of geographical boundaries and housing units in the field listing and the geographical distribution of different violent crimes in Chicago.

INTEGRATING GOOGLE STATIC MAPS WITH SAS

The tool offers a macro `%SASGoogleMaps` that allows SAS users to display various spatial patterns on Google Static Maps. The prototype of the macro, along with some brief explanations of each argument, is shown in the following. We only list the most relevant arguments here, and have left out arguments related to the detailed control of the graphical appearance (e.g., the color or the size of the plotted dots).

```
%SASGoogleMaps(
    center = ,                  /* center coordinates of map      */
    zoom = 15,                  /* zoom level of map              */
    size = 640x640,             /* size of map                    */
    format = png,               /* format of output image         */
    maptype = roadmap,          /* type of map                    */
    input = ,                   /* data to be plotted             */
    lat = Y,                    /* the latitude to be used        */
    lon = X,                    /* the longitude to be used       */
    plot_type = ,               /* type of spatial pattern        */
    out_dir = ,                 /* output directory               */
    out_file = ,                /* output image name              */
    group = ,                   /* pattern (polygon) group        */
    add = 0 ,                   /* add on an input image?         */
    map_in = ,                  /* path for an input map          */
    read_meta = 1,              /* read meta info?                */
    convert_coord = 1,          /* convert coordinate?            */
    ... )                       /* other control parameters       */
```

In generating a graph that overlays a SAS spatial pattern with a Google static map, the macro `%SASGoogleMaps` implements the following three fundamental steps internally:

1. Obtaining a static map image for a specified location of interest from Google Statics Maps;
2. Preparing the SAS data set to be plotted by converting the location coordinates to their pixel representations;
3. Importing the static map as a background image and then plotting the geographical information contained in the SAS data set.

However, to make the macro flexible for different purposes, we have structured it with some flags so that it could implement only a subset of the above three steps. For example, when the argument `plot_type` is missing, `%SASGoogleMaps` only downloads the required map without overlaying additional patterns; when `convert_coord=0`, the coordinate transformation is prohibited; when `add=1`, no map is downloaded, and instead an existing graph is read in as background image in the overlaying process. To help users better understand how the tool works, we now explain the details of each of the above three steps.

RETRIEVING GOOGLE STATIC MAPS

To utilize a Google static map, the first step is to retrieve the map from the Google Static Maps server. The server responds to a standard HTTP request via a special URL, parameter-by-parameter, and returns an image displaying the geographic location of interest. Parameters affecting the requested map are the center of the location (`center`), the zoom level (`zoom`), the size of the image (`size`), the type of the map (`maptype`), the format of the image (`format`). To retrieve a map without plotting additional information from a SAS data set, the `%SASGoogleMaps` macro needs to be called with the argument `plot_type` missing (the default). A simple example image showing the downtown Chicago area can be downloaded via the following call, and the resulting image is shown in Figure 1a:

```
%SASGoogleMaps(center = %str(41.878114,-87.629798),
                zoom = 14,
                size = 320x320,
                maptype = roadmap,
                format = png)
```

In addition to the exact location coordinates, the macro also recognizes input of the address of the location of interest as a simpler and friendlier query method. In this case, an intermediate step is implemented within the macro to translate the address to geographic coordinates via the geocoding tool provided by Google Geocoding API. Consequently, the following call is also legal, and perhaps more frequently used (this retrieves the map of the campus of the University of Chicago):

```
%SASGoogleMaps(center = University of Chicago,
                maptype = hybrid,
                zoom = 15,
                size = 320x320)
```

In the above, we have made use of the default value for the argument `format`, which is `png`. The resulting image is displayed in Figure 1b.

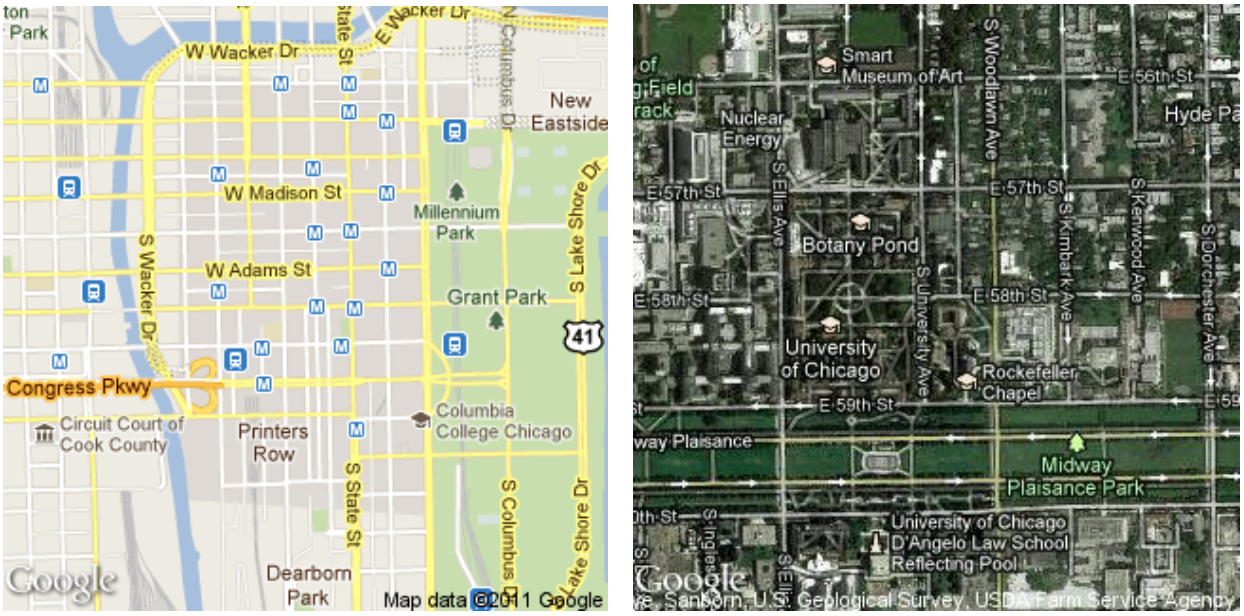

Figure 1. (a) Static map with latitude/longitude inputs, (b) Static map with address inputs

A third and even more general way to call the macro is to input a SAS data set, whose geographic information is then used to calculate the center and the maximum zoom level allowed. For example, if the user wants to overlay the set of geographical locations defined by an ESRI shape file imported into SAS, then it is necessary for the center and the zoom level of the map to be dependent on these location coordinates to ensure that the resulting map includes all these locations. The following code illustrates how to download an image for an imported ESRI shape files by specifying the imported shape files as the input data set (image not displayed). In this case, the `center` argument has to be missing. More details of this type of specification will be discussed in the application section.

```
%SASGoogleMaps(input = input-dataset,
               lat = variable-name-containing-the-latitude,
               lon = variable-name-containing-the-longitude,
               size = size-of-image,
               ... )
```

COORDINATES TRANSFORMATION AND SCALING

In the second step, we convert the values of the latitude and the longitude to map tile and pixel coordinates that are used by Google to make static maps images. These map tile coordinates are then scaled to fit in the default (0, 0) to (100, 100) coordinate scheme (corresponding to the relative portion of the viewport) adopted by the DSGI. Without the coordinate transformation and scaling, the spatial data could not be displayed accurately on the background static map image. The argument `convert_coord` in `%SASGoogleMaps` indicates whether coordinate transformation is to be performed in the macro call. The default is `convert_coord=1`, meaning that the data contain the latitude and the longitude and these values are to be converted to the DSGI coordinates. However, when the input is already on the appropriate plotting scale, e.g., a manually inputted data on the DSGI scale for adding textual labels, it is necessary to turn off the default conversion by setting `convert_coord=0`. For more details about coordinate transformation, we refer interested readers to the Google documentation or Loecher (2010).

DISPLAYING SAS DATA USING DATA STEP GRAPHICS INTERFACE (DSGI)

In the last step, we import the retrieved static map and plot additional spatial information available in SAS data set using functions from the SAS DATA Step Graphics Interface (DSGI). We adopt the DSGI here because of its flexibility to read in an existing image and add more graphical features. Also, since DSGI often executes the importing and editing procedures within one DATA step, it is a fast environment for creating customized graphical output. See detailed information in the SAS/GRAPH reference.

To enable `%SASGoogleMaps` to overlay spatial patterns with static maps (or existing images), one specifies the desired type of plot through the `plot_type` argument. The types of plots currently supported are: `points` (scattered

points defined by the longitude and the latitude), `lines` (a set of lines each connecting two points), `polylines` (a sequence of lines defining a polygon), `polygon` (a filled polygon) and `text` (textual feature on plot). If one of these options is specified, the spatial information determined by the two variables supplied to the arguments `lon` and `lat` in the input data will be processed, converted to pixel coordinates if `convert_coord=1`, and plotted onto a newly downloaded map or an imported image.

The argument `add` affects the behavior of `%SASGoogleMaps` by determining whether the spatial pattern is to be added to an existing image (as opposed to downloading a new map) in the overlaying process. If `add=0` (the default), `%SASGoogleMaps` will download the necessary map first using either the information supplied in `center` or by extracting such information from the input data, and then overlay this map with the requested pattern. In this case, the processes of retrieving static map image and plotting users' spatial data on map are automatically combined. On the contrary, when `add=1`, no new map will be downloaded. Instead, `%SASGoogleMaps` reads in an existing image specified in the `map_in` argument, and uses that image as a background when plotting the required spatial patterns. For further simplification, we allow `map_in` to be missing, in which case, the output image from the most recent call of `%SASGoogleMaps` will be imported. As a result of this design, a sophisticated graph can be built up gradually, layer by layer, where different layers contribute to the graph with different sets of graphical features that can possibly come from different data sources. Such an approach dramatically extends the scope of the spatial features that can be included and improves the flexibility of the macro, as we will illustrate in the following applications.

APPLICATIONS

SURVEY FIELD LISTING MAPS

In area-probability surveys, one important step is to select a sample of housing units to be interviewed from a target area. This is often accomplished through the practice called field listing. The goal of field listing is to construct a frame of housing units, from which a representative sample of cases will be selected for in-person interviews. However, field listing of the housing units is often quite expensive and time-consuming, so it is important to provide high-quality listing instruments, e.g., paper listing maps, to field listers in order to enhance the efficiency of the listing and data collection process. However, traditional listing maps created by professional mapping software are often limited by the capacity of the available geographic data (e.g., digitized layers of roads, rivers, water bodies etc.). Compared to traditional listing maps, Google Maps images are more informative and provide a more familiar geographic scale for field listers to interpret spatial information. This is particularly beneficial when the listing area is located in a rural place with limited identifiable geographic features, or when invisible boundaries (e.g., political boundaries as opposed to visible boundaries such as roads, rivers and railways) are present.

An illustrative example is shown below using a census block in the west suburb of Chicago. Figure 2a shows the regular listing map provided to field listers created by professional mapping software (We have changed the street names for confidentiality concerns).

Figure 2. (a) Traditional block map (b) Google Image based block map

This map shows an invisible boundary along the north side - the north boundary is not defined by visible boundaries such as roads, rivers or railways. When a lister is in the field, he/she will start from the northwest corner of the block and continue clockwise to record the address/information for every housing unit in this census block until the whole block is covered. The question arises as to which housing unit is the right spot to start with. Apparently, Figure 2a produced from traditional listing map does not provide enough information to interpret the most northwest housing unit within the census block. Alternatively, we obtain a satellite image from the Google Static Maps server and overlay it with block boundaries and produce Figure 2b. We see that this new map clearly defines the first house unit in the northwest corner of block, and field listers are more likely to be able to identify which housing unit is within the block boundary.

To produce Figure 2b, we call the `%SASGoogleMaps` macro as follows, where we specify the input data set (`input=block`), map type (`maptype=statellite`) and plot type (`plot_type=polylines`) as well as the line characteristics such as color (`lcolor`), type (`ltype`), and width (`lwidth`). By specifying `plot_type=polylines` and `add=0` (the default), this call of `%SASGoogleMaps` performs both map downloading and pattern overlaying. We also have implicitly used the default `lat=Y` and `lon=X` to tell the macro where to find the latitude and the longitude information in the input data.

```
%SASGoogleMaps(input=block,                          /* data including boundary */
               plot_type=polylines,                  /* connected lines         */
               lcolor = 2,                           /* red line color          */
               ltype = 20,                           /* dashed line type        */
               lwidth = 2,                           /* line width              */
               out_dir= H:\SAS\SAS-Google\Example,   /* output directory        */
               out_file=tmp3,                        /* output image name       */
               maptype=satellite);                   /* type of map downloaded  */
```

CRIME DISTRIBUTION IN CHICAGO

The City of Chicago has published a data base with reported incidents of crimes that occurred in the city of Chicago from 2001 to present. Included in the data are the location of the crime, the type of the crime, the date of the crime, indicator of whether the criminals have been arrested, and other related information. The full dataset can be accessed at: http://data.cityofchicago.org/Public-Safety/Crimes-2001-to-present/ijzp-q8t2. Here, we include only one year of data (Aug.2010-Aug.2011), and only violent crimes such as assault, robbery, criminal sexual assault and homicide.

We first focus on the community of Hyde Park, where the University of Chicago is located. It would be interesting to see the geographical distributions of the different types of crimes in this neighborhood. Such a crime map could provide some helpful information about the safety of specific areas within this neighborhood.

The following code would retrieve the map of Hyde Park and plot the incidents of crimes in the input data file as colored points onto the Google static map, the color indicating the type of the crime. The coloring of different types of crimes is achieved through the `pcolor` argument, which is supplied with a variable name from the input in this example. The variable color in the input data set `hydepark` defines the color index for different types of crime. The map and the colored points are shown in Figure 3.

```
%SASGoogleMaps(input = hydepark,                     /* data containing crime   */
                 maptype = terrain,                  /* terrain may type        */
                 plot_type = points,                 /* plot of locations       */
                 pcolor = color,                     /* dot color (column name) */
                 psize = 1.5,                        /* dot size  (constant)    */
                 size = 500x500,                     /* size of image           */
                 out_dir=H:\SAS\SAS-Google\Example,/* output directory       */
                 out_file=tmp4);                     /* output image name       */
```

The above code only plots the locations of the crimes, represented by colored points on the map. It may be helpful to add the boundary that delineates the Hyde Park community, which will come from a different shape file `hydepark_bd`. This can be achieved readily by calling another `%SASGoogleMaps` with the image from the above as background.

```
%SASGoogleMaps(input = hydepark_bd,                  /* data including boundary */
                 plot_type = polylines,              /* connected polylines     */
                 add = 1,                            /* add onto last plot      */
                 lcolor = 1 ,                        /* set color to be black   */
                 ltype = 20 ,                        /* dashed line type        */
                 lwidth = 2);                        /* line width              */
```

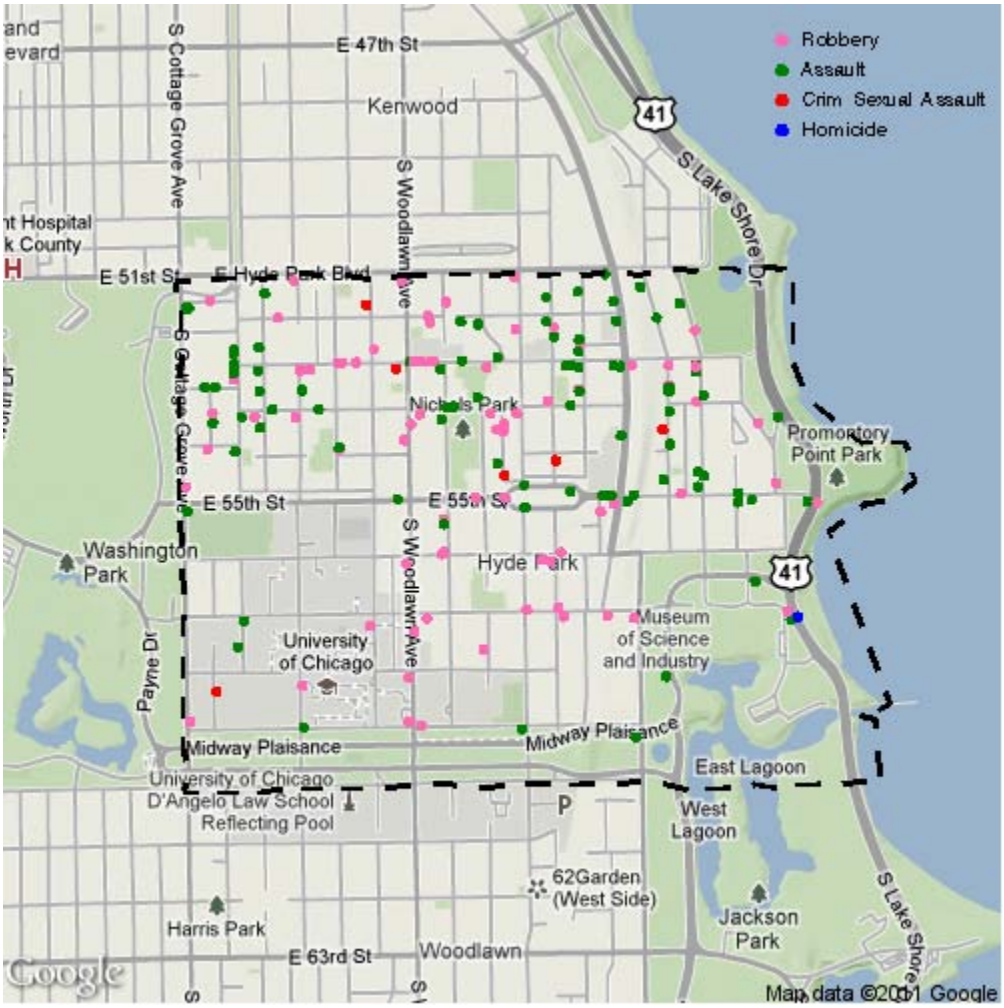

Figure 3. Locations of incidents of crime in Hyde Park, Chicago

In the above, we have specified `add=1` to indicate that the background image will be read in from an existing file rather than downloaded from the Google Static Maps server. We have also used the shortcut of specifying a missing value of `map_in` (the default) so that the most recently outputted image from a call of `%SASGoogleMaps`, that is, the map in `H:\SAS\SAS-Google\Example\tmp4.png`, will be imported automatically. And since the second call did not specify `out_dir` or `out_file`, the imported image will be automatically overwritten with the new image, where community boundary is added. The black dashed line in Figure 3 is a result of the second call.

In a similar fashion, one can add legends onto the image that will provide information to interpret the different coloring of the points. Another two calls of `%SASGoogleMaps` have to be performed, with the first call overlaying texts and the second call plotting colored points. A data set containing the coordinates (in this example, DSGI coordinates) of the textual labels and points are created beforehand to serve as the input file for `%SASGoogleMaps` calls.

```
data legend ;
    infile datalines delimiter=',';
    input x1   y1  x2 y2 pcolor crime $20. ;       /* x1,y1: coord for text     */
    datalines;                                     /* x2,y2: coord for dot      */
    80, 96, 78, 96.5, 25, Robbery                  /* pcolor: coloring of crime */
    80, 93, 78, 93.5, 3, Assault
    80, 90, 78, 90.5, 2, Crim Sexual Assault
    80, 87, 78, 87.5, 4, Homicide
    ;
run ;
```

The following two calls of `%SASGoogleMaps` add onto the previous graph with textual labels and colored points using the information from the above manually constructed data. Notice that we have used `convert_coord=0` to keep the macro from converting the coordinates, and `add=1` to indicate the requested patterns are to be added onto previous images.

```
%SASGoogleMaps(input = legend,                    /* data including text     */
                lat = y1,                         /* y coordinate            */
                lon = x1,                         /* x coordinate            */
                plot_type = text,                 /* plot text               */
                add = 1,                          /* add to last plot        */
                convert_coord = 0,                /* do not convert coord    */
                text = crime,                     /* column having text      */
                tcolor = 1 ,                      /* black text color        */
                theight = 2 ,                     /* text height             */
                tfont = swiss);                   /* text font               */

%SASGoogleMaps(input = legend,                    /* data including dots     */
                lat = y2,
                lon = x2,
                plot_type = points,               /* plot dots               */
                add = 1,
                convert_coord = 0,
                pcolor = pcolor ,                 /* coloring dots in pcolor */
                psize = 2 ,
                ptype = 12);
```

Figure 3 is the final result of the four calls of `%SASGoogleMaps`. One sees that the main campus of the University of Chicago (between 55th St and 60th St, and between Cottage Grove Ave and Woodlawn Ave) is the safest place in this neighborhood, possibly due to the extra police protection from the University police department. Assault, both simple and aggravated, is the most frequent crime type in Hyde Park. The distribution has a clear pattern: the vast majority of the assault cases happened in the northern part (between 51st St and 55th St), while few occurred south of 55th St. Robbery, another frequent crime type, is ubiquitous in this neighborhood except the university campus. Concerning other two violent crimes, few criminal sexual assaults and homicides are found in Hyde Park in the past year.

Next, we consider all the communities in Chicago, and plot the ten communities with the highest number of crimes and the ten communities with the lowest number of crimes. This can be achieved in a similar fashion as above: we call the `%SASGoogleMaps` macro twice: in the first call, we specify `plot_type=polygon` to get filled polygons, where the color (`fcolor`) and the filling style (`fstyle`) are supplied by the two columns `color` and `style` in the input data set `chicomm2`, respectively; and in the second call, we overlay all 77 Chicago community area boundaries from a different data set `chicomm` using `plot_type=polylines`. In both calls, we specify `group=community` so that the supplied coordinate information is plotted polygon by polygon (community by community). The code we used is in the following, and the resulting graph is displayed in Figure 4. The code for adding legends is similar to the previous example and is left out here.

```
%SASGoogleMaps(input = chicomm2,                  /* data with crime rank    */
                maptype = roadmap,
                plot_type = polygon,              /* plot filled polygon     */
                fcolor = color,                   /* coloring as in color    */
                ftype = HATCH,                    /* fill type as HATCH      */
                fstyle = style,                   /* fill style as in style  */
                group = community,                /* polygon identifier      */
                out_dir=H:\SAS\SAS-Google\Example,
                out_file=tmp5,
                size = 500x500);

%SASGoogleMaps(input = chicomm,                   /* data with boundary      */
                plot_type = polylines,            /* plot connected lines    */
                add = 1,                          /* add to last plot        */
                lcolor = 18 ,
                ltype = 1 ,                       /* solid line              */
                lwidth = 1.5,
                group = community);               /* polygon identifier      */
```

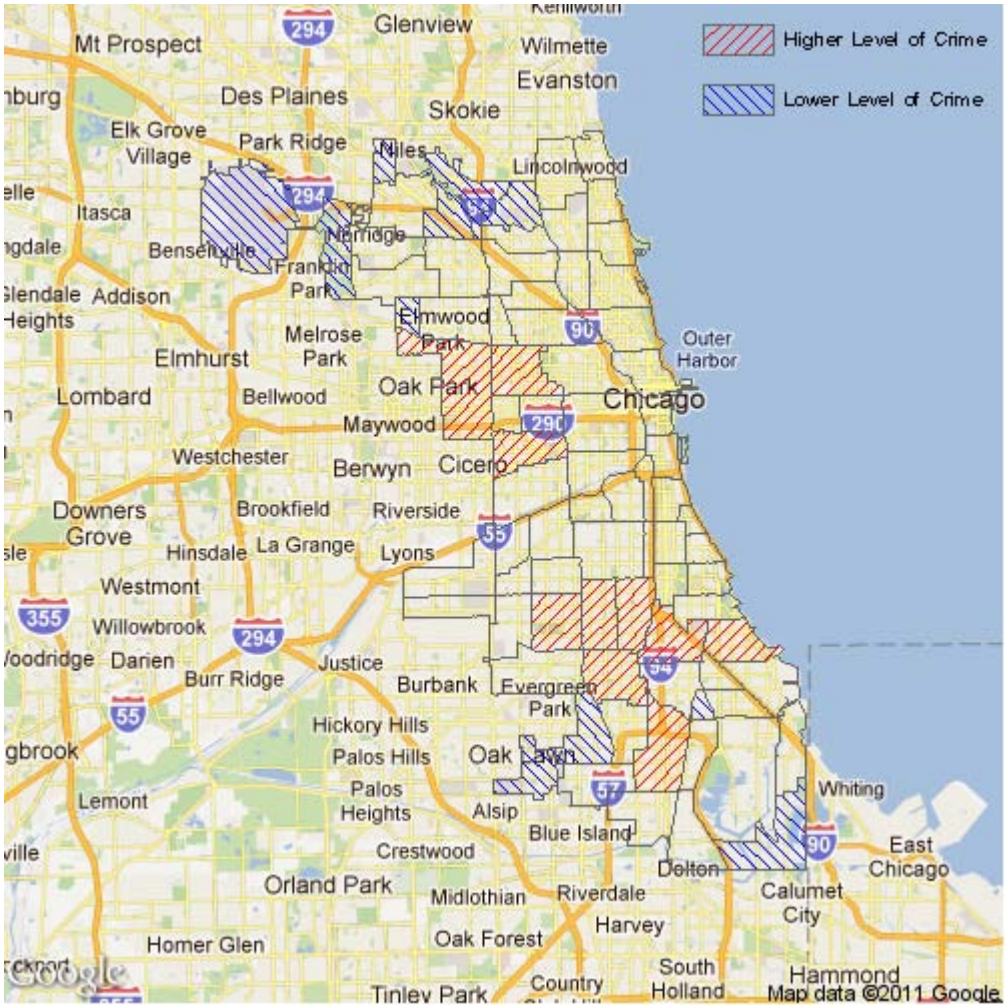

Figure 4. Communities of higher and lower levels of crime in Chicago

CONCLUSION

In this paper, we have presented a tool that allows various SAS spatial data to be displayed on Google static maps and shown that there are many advantages of integrating SAS and Google Static Maps through two separate applications. The tool we present is straightforward and simple to use and does not require knowledge of other programming languages, which is an advantage to other similar imaging tools in SAS. The tool is also very flexible in many ways. First, it allows different forms of spatial inputs (latitude/longitude values, addresses, or existing SAS datasets with geospatial information) to be specified to retrieve the Google static maps, performing necessary geocoding of the address or calculation of the spatial bounding box internally. Second, the macro makes available various types of plots such as points, lines, polygons and texts, and provides the users an easy method to access the desired type of pattern to be displayed on the static maps. Third, the tool is designed to allow the final image to be constructed layer by layer, where each layer contains one set of plotting characteristics. As a result, different graphical patterns available in the tool (points, lines, polygons or texts) or geographical information from different data sources can be combined and plotted on the same graph. This design greatly reduces the difficulty of constructing complex and sophisticated graphs and dramatically extends the scope of the spatial features that can be included in the graph.

It should be noted that, when using Google Maps APIs to include maps and other contents in applications, individuals must follow Google Terms of Service. Also, the use of Google Geocoding API is subject to a query limit of 2,500 location requests per day and geocoding results must be used in conjunction with a Google Map. See details in http://code.google.com/apis/maps/terms.html .

The source code of the tool, a companion help documentation and the data sets used in the paper can be downloaded at: http://www.actuaryzhang.com/software/software.html

REFERENCES

1. Azimaee M. (2010). KML Macro: Integrating SAS® and Google API and Its Application in Mapping Manitoba's Health Data on Google Earth and Google Map. SUGI 2010.

2. Drunkenbrod J. and Mintz D. (2008). Using SAS® and Google Earth to Access and Display Air Pollution Data. SUGI 2008.

3. Loecher M. (2010). Plotting on Google Static Maps in R.

4. Massengill D. (2010). Google Maps and SAS/GRAPH®. SUGI 2010.

5. SAS Institute Inc. 2010. SAS/GRAPH 9.2 documentation in the Knowledge Base. http://support.sas.com/documentation/onlinedoc/graph/index.html .

CONTACT INFORMATION

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

Name: Jizhou Fu
E-mail: fu-jizhou@norc.org

Name: Yanwei (Wayne) Zhang
E-mail: actuary_zhang@hotmail.com
Web: http://www.actuaryzhang.com