MWSUG 2017 - Paper AA07

GREMOVE, Reassign, and let's GMAP! A SAS® Trick for Generating Contiguous Map Boundaries for Market-Level Research

Chad Cogan, Arbor Research Collaborative for Health, Ann Arbor, MI
Jeffrey Pearson, Arbor Research Collaborative for Health, Ann Arbor, MI
Purna Mukhopadhyay, Arbor Research Collaborative for Health, Ann Arbor, MI
Charles Gaber, Arbor Research Collaborative for Health, Ann Arbor, MI
Marc Turenne, Arbor Research Collaborative for Health, Ann Arbor, MI

ABSTRACT

In health services research, accurate health care market definitions are crucial for assessing the potential market-level consequences of policy changes. Political units of geography (e.g. counties) are generally not sufficient for capturing the service area of a provider. Alternatively, researchers may generate customized boundaries using data-driven approaches based on patient flow only to find that their newly defined areas are not contiguous. We use a novel approach to correct for the lack of contiguity using the information produced by the GREMOVE procedure. GREMOVE is often used along with the GMAP procedure when there is a need to generate customized boundaries on a map by removing the internal boundaries of smaller units of geography. However, SAS® users may not be aware of the logic used by PROC GREMOVE to assign segment values and the underlying data that goes into the maps. We first examine the logic used by PROC GREMOVE, and the map output data set it produces. We identify some potential limitations of GREMOVE along with some alternative uses, which we demonstrate using basic polygons. We then look at customized map boundaries produced using a data-driven approach to combine ZIP code tabulation areas (ZCTAs) based on patient flow and show how GREMOVE identifies non-contiguous segments in a newly defined area. We then use a SAS trick to modify the GREMOVE logic for segment assignment, and generate new contiguous boundaries.

INTRODUCTION

Map data sets include rows of x and y coordinates, which are often grouped by segments. Each segment defines the polygon that SAS uses to draw the boundaries that connect the coordinates for that area. If the value of the segment is different the program knows which polygons should be drawn separately. However, how many SAS users know what the value assigned to the segment represents and how it is assigned? Additionally, why would we care about the order of this value?

The GREMOVE procedure is often used for map data processing. The GREMOVE procedure does not generate any graphics itself. Rather, it produces an output data set that can later be used as an input map data for the GMAP procedure. The GREMOVE procedure can be used to generate larger unit areas by removing the internal boundaries that smaller unit areas share.

The PROC GREMOVE statement is used to identify the input and output map data sets.

PROC GREMOVE <DATE=*input-map-data-set*>
<OUT=*output-map-data-set*>;

The input data should include:

- The horizontal axis (longitude) coordinates on the map, which are named X variable.
- The vertical axis (latitude) coordinates on the map, which are named the Y variable.
- One or more identifiers that represent the current smaller unit areas (e.g. states). Include these identifiers in the ID statement.
    - **ID** *id-variable(s)*;

- One or more identifiers for a group of smaller areas that will form the new larger unit area when combined (e.g. census region). Include these identifiers in the BY statement. Be sure that any variables included in the BY statement are sorted properly.
    - **BY** <DESCENDING>*variable-1*
      <…<DESCENDING>*variable-n*>
      <NOTSORTED>;

All variables in the input map data set, except for the original X, Y, and ID variables, will be carried over to the output map data set.

The input map data set is not required to contain boundaries that neighbor each other (e.g. Alaska does not border the 48 contiguous states of the US, but the entire United States can be drawn as one region to include Alaska and Hawaii). However, all coordinates used to draw smaller areas should also be included with the larger area identifier. For example, ZIP code tabulation areas (ZCTAs) can and do cross county lines. Hence, be careful if trying to draw an accurate boundary for a county that is generated from a group of ZCTAs. In contrast, counties do not cross state lines. Therefore, a state is a good example of a boundary for a larger unit area that could be drawn using the boundaries of counties.

The input map data set may also include variable SEGMENT, which is often used to differentiate between non-contiguous segments of the smaller areas. Variable SEGMENT is not required in the input map data set for PROC GREMOVE. However, if the input map data set does include variable SEGMENT, and it is desirable to retain this variable in the output map data set, then it will need to be renamed in the input map data set, otherwise, variable SEGMENT will be overwritten. PROC GREMOVE will create a new SEGMENT variable in the output data to differentiate between any non-contiguous segments in the new larger area.

This new SEGMENT variable is key to determining whether the newly generated areas are contiguous or not. If an area only has one segment value assigned, then the area is completely contiguous with no segments separated by land or water. To better understand how segment assignment works, imagine you are drawing an area on a sheet of paper using a pencil. If the area can be drawn without lifting the pencil to a new spot then the area being drawn only has one segment. Each time the pencil must be lifted to begin drawing at another point a new segment is assigned.

PROC GREMOVE assigns values to one or more segments in the following manner:

- **1 Segment:** An area with only one segment should always have a value of 1 for variable SEGMENT.
- **2 or more Segments:** If there is more than one segment in an area separated by either land or water PROC GREMOVE assigns a value of 1 to the first ordered segment. It then assigns values of 2, 3, 4, and so on to the additional segments. The value assigned to each segment is ordered based on *the size of the boundary around the defined area*, where the largest surrounding boundary is assigned segment 1.

The size of the boundary should not be confused with the size of the area as measured by the number of square units.

PROC GREMOVE USING SMALL POLYGONS (EXAMPLE 1)

In example 1, seven small square polygons of equal size are defined. Each polygon has four X and Y coordinates, but are members of the same region. Polygon A is not contiguous with any other polygons. It has a boundary size of 4 units with an area of 1 square unit. Polygons B and C are contiguous. They share one border and two of the same coordinates. Together, these polygons have a total boundary size of 6 units with an area of 2 square units. Polygons D, E, F, and G are also contiguous. Polygon D shares one border and two coordinates with Polygon E, which shares one border and two coordinates with Polygon G, which shares one border and two coordinates with Polygon F. Together, these polygons

have a total boundary size of 8 units with an area of 4 square units. In the code below, coordinates and information related to each polygon are generated using the DATA step.

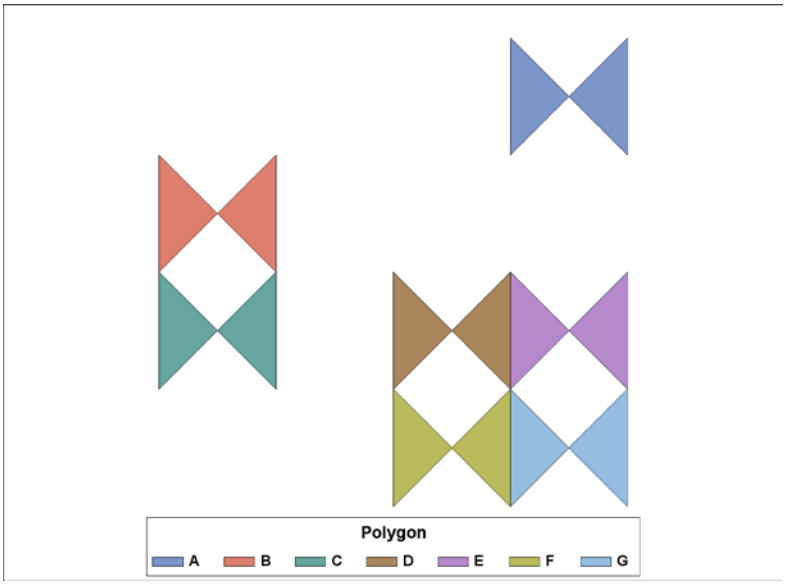
```
/******* EXAMPLE 1 ********/
data polygons;

Region=1;
 Polygon='A'; border_units=4; sq_units=1;
  x=4; y=4; output;
  x=4; y=5; output;
  x=5; y=5; output;
  x=5; y=4; output;

Polygon='B'; border_units=6; sq_units=2;
  x=1; y=3; output;
  x=1; y=4; output;
  x=2; y=4; output;
  x=2; y=3; output;

Polygon='C';
  x=1; y=2; output;
  x=1; y=3; output;
  x=2; y=3; output;
  x=2; y=2; output;

 Polygon='D'; border_units=8; sq_units=4;
  x=3; y=2; output;
  x=3; y=3; output;
  x=4; y=3; output;
  x=4; y=2; output;

 Polygon='E';
  x=4; y=2; output;
  x=4; y=3; output;
  x=5; y=3; output;
  x=5; y=2; output;

 Polygon='F';
  x=3; y=1; output;
  x=3; y=2; output;
  x=4; y=2; output;
  x=4; y=1; output;

 Polygon='G';
  x=4; y=1; output;
  x=4; y=2; output;
  x=5; y=2; output;
  x=5; y=1; output;
run;
```

The coordinates for these polygons follow a specific order so that the GMAP procedure can draw boundaries in a clockwise manner (e.g. {4, 4} ➔ {4, 5} ➔ {5, 5} ➔ {5, 4}). Alternatively, the coordinates could also be ordered counterclockwise (e.g. {4, 4} ➔ {5, 4} ➔ {5, 5} ➔ {4, 5}). However, be careful if sorting map input data sets. This might cause unintended results. Never sort the map input data set by x and y coordinates! Sorting can alter the way SAS connects coordinates on a map. SAS is reading and writing coordinates around the polygon boundaries.

```
/*Never sort map input data set by x and y coordinates!!!!!*/
proc sort data=polygons out=poly_sort; by polygon x y; run;

proc gmap data=polygons map=poly_sort;
   id polygon;
   choro polygon / discrete legend=legend1;
 run;
quit;
```

In this example, if the map input data set for square polygons are sorted by polygon and then by x and y coordinates the result will be a line drawn from one diagonal coordinate to the other changing the boundaries drawn on the map.

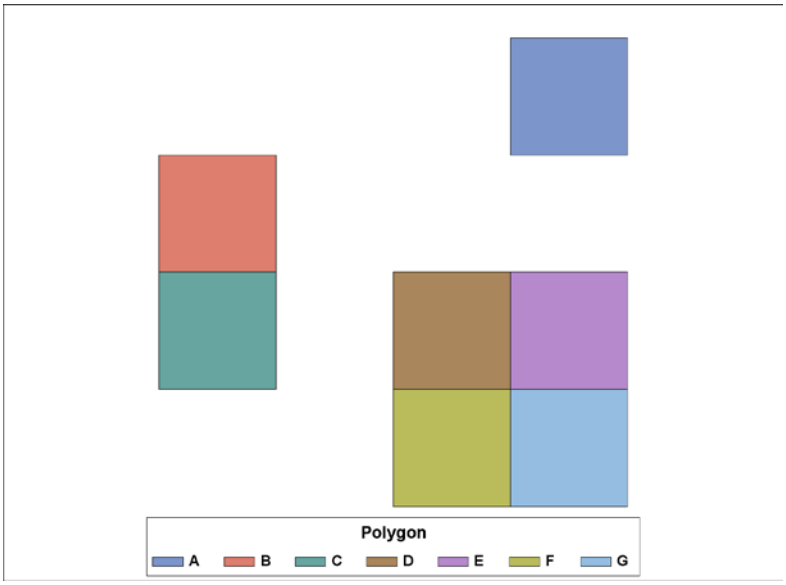
Figure 1. Map of Polygons When Sorted by Coordinates



Below is the map of the polygons using the original order assigned to the x and y coordinates. The map coordinates and data used for the map can be from the same data set if the x and y coordinates remain unsorted.

```
/*Draw polygons*/
goptions border;
   legend1 cborder=black label=(font="Arial/bold" height=1.5 position=top
   justify=center) position=(bottom center)
         value=(height=1.3 font="Arial/bold" justify=left);

proc gmap data=polygons map=polygons;
   id polygon;
   choro polygon / discrete legend=legend1;
 run;
quit;
```

Figure 2. Map of Polygons for Example 1



Now let's use the GREMOVE procedure to remove internal boundaries of the polygons that come together to form the overall region. Recall that the GREMOVE procedure will drop the id variables from the output map data set. If there is a need to retain this variable then create a copy of it. In this case, let's create a variable 'poly_copy'.

```
/*Create a copy of variable polygon*/
data polygons;
      set polygons;
      Poly_copy=Polygon;
run;
```

The GREMOVE procedure keeps all polygon coordinates that form the outer boundary of the new area. However, if multiple polygons share the same coordinate then the GREMOVE procedure will remove duplicate coordinates. For example, Polygons B and C share coordinates {1, 3} and {2, 3}. In this case, the GREMOVE procedure removes these coordinates from Polygon B in the output data set. However, we would not know this if we did not create a copy of the ID variable Polygon.

```
/*Input map data set using Polygons*/
proc print data=polygons noobs;
title "Input map data set using Polygons";
run;
```

Output 1. Print of Polygon Coordinates Before Using GREMOVE

Input map data set using Polygons

Region	Polygon	border_ units	sq_units	x	y	Poly_ copy
1	A	4	1	4	4	A
1	A	4	1	4	5	A
1	A	4	1	5	5	A
1	A	4	1	5	4	A
1	B	6	2	1	3	B

1	B	6	2	1	4	B
1	B	6	2	2	4	B
~~1~~	~~B~~	~~6~~	~~2~~	~~2~~	~~3~~	~~B~~
1	C	6	2	1	2	C
1	C	6	2	1	3	C
1	C	6	2	2	3	C
1	C	6	2	2	2	C
~~1~~	~~D~~	~~8~~	~~4~~	~~3~~	~~2~~	~~D~~
1	D	8	4	3	3	D
1	D	8	4	4	3	D
~~1~~	~~D~~	~~8~~	~~4~~	~~4~~	~~2~~	~~D~~
~~1~~	~~E~~	~~8~~	~~4~~	~~4~~	~~2~~	~~E~~
~~1~~	~~E~~	~~8~~	~~4~~	~~4~~	~~3~~	~~E~~
1	E	8	4	5	3	E
~~1~~	~~E~~	~~8~~	~~4~~	~~5~~	~~2~~	~~E~~
1	F	8	4	3	1	F
~~1~~	~~F~~	~~8~~	~~4~~	~~3~~	~~2~~	~~F~~
~~1~~	~~F~~	~~8~~	~~4~~	~~4~~	~~2~~	~~F~~
1	F	8	4	4	1	F
~~1~~	~~G~~	~~8~~	~~4~~	~~4~~	~~1~~	~~G~~
~~1~~	~~G~~	~~8~~	~~4~~	~~4~~	~~2~~	~~G~~
1	G	8	4	5	2	G
1	G	8	4	5	1	G

```
/*Remove inside borders of Region*/
proc gremove data=polygons out=area;
   by Region;
   id polygon;
run;

/*Data used to map region with polygon borders removed includes variable
segment*/
proc print data=area noobs;
title "Data used to map region with polygon borders removed";
run;
```

There are no duplicate coordinates in the output map data set created by the GREMOVE procedure. Additionally, the coordinates are ordered by the segment value that was assigned, and within each segment they are ordered counterclockwise beginning with the smallest ordered coordinate (e.g. {3, 1} for segment 1). Since the smaller segments within the new region that formed were not contiguous, multiple values were assigned. Value 1 was assigned to the segment with the largest border, which was created by combining Polygons D, E, and F. Values 2 and 3 were assigned to the segments with the second and third largest borders, respectively.

Output 2. Print of Polygon Coordinates with Rows Removed by GREMOVE

x	y	SEGMENT	border_ units	sq_units	Poly_ copy	Region
3	1	1	8	4	F	1
4	1	1	8	4	F	1
5	1	1	8	4	G	1
5	2	1	8	4	G	1
5	3	1	8	4	E	1
4	3	1	8	4	D	1

3	3	1	8	4	D	1
3	2	1	8	4	F	1
1	2	2	6	2	C	1
2	2	2	6	2	C	1
2	3	2	6	2	C	1
2	4	2	6	2	B	1
1	4	2	6	2	B	1
1	3	2	6	2	C	1
4	4	3	4	1	A	1
5	4	3	4	1	A	1
5	5	3	4	1	A	1
4	5	3	4	1	A	1

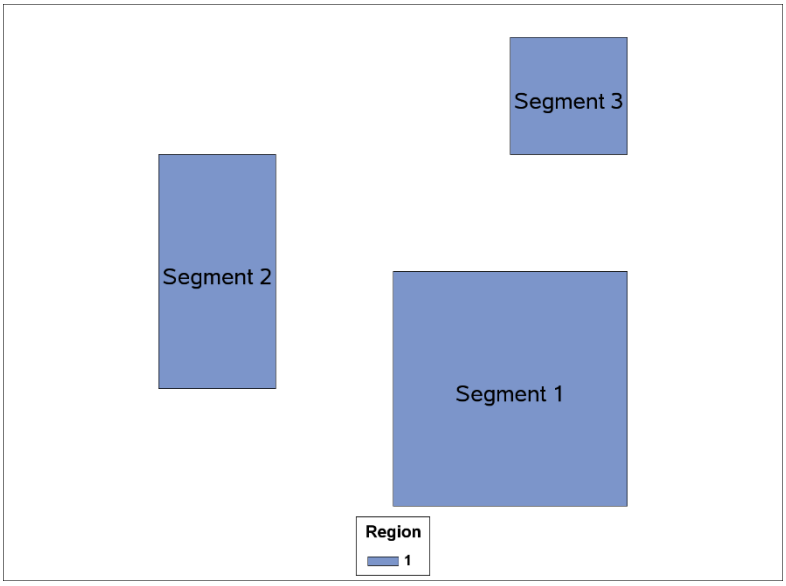
Next, we will look at the map of the new area. To show segment values we will need to calculate the segment centroid coordinates so that the labels are centered. The %centroid macro is one of many pre-constructed annotate macros, which should be available within your SAS software. You must first run %ANNOMAC macro to initiate these macros. The macro retrieves the centroids of specified polygons. The arguments required for the macro include an input data set, output data set, and ID variables for the area unit. There is also an optional argument to calculate a centroid for a specified segment. For example, if the user requests SEGONLY=1 then the macro returns a centroid coordinate only for segment 1.

```
/*Calculate centroid of the segment coordinates using macro*/
%annomac;
%centroid(area,seg_centroids,segment);

/*Create annotation data to label area segments*/
data anno_labels;
  length color $ 8 text $ 10 style $ 25;
  set seg_centroids;
  retain x y xsys ysys hsys when position function size color text;
     segname=compress("Segment"||segment);
     segname=tranwrd(segname,"t","t ");
     xsys='2'; ysys='2'; hsys='3'; when='a'; position='5';
     function='label';
     style="'Albany AMT'";
     text=segname;
     color='black';
     size=4;
  output;
run;

/*Draw Region*/
title1 h=2.5 "Example 1.2: Region with non-Contiguous Polygons";
proc gmap data=area map=area;
  id Region;
  choro Region / discrete legend=legend1 anno=anno_labels;
 run;
quit;
```

After removing the polygon borders the new area is called region. Region shares the same color and legend label on the map since it is considered one unique area. However, since the segments of the region are not contiguous more than one value is assigned. Here, we can clearly see that largest segment is assigned value 1. The value of 1 is assigned to the segment since it has the largest boundary size, but it just so happens that this segment is largest in terms of area. Usually, we expect larger areas

to also have large boundaries. Segment values of 2 and 3 were also assigned to the second and third largest boundaries, respectively.

Figure 3. Map of Region with Polygon Borders Removed for Example 2



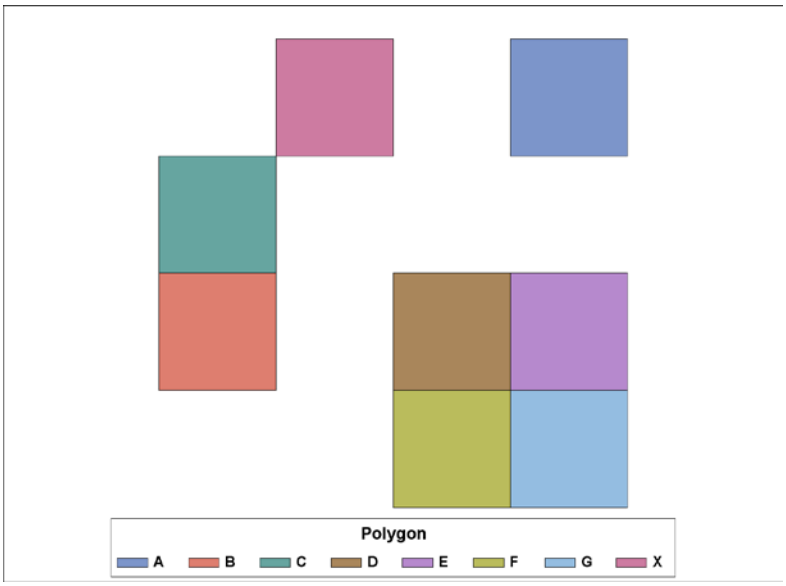
PROC GREMOVE USING SMALL POLYGONS (EXAMPLE 2)

Example 2 uses the same polygons in example 1. However, this time we will add one additional square polygon that borders diagonally with polygon B. Although areas that touch boundaries diagonally do not share a border they are still considered contiguous by the GREMOVE procedure. Together, polygons B, C, and X will have a total boundary size of 10 units with an area of 3 square units. In the code below, we again generate coordinates and information related to each polygon using the DATA step.

```
/******* EXAMPLE 2 ********/

/*Manually generate polygon*/
data polygons2;

Region=1;
 Polygon='A'; border_units=4; sq_units=1;
  x=4; y=4; output;
  x=4; y=5; output;
  x=5; y=5; output;
  x=5; y=4; output;

/*Add an extra square polygon diagonal to Polygon B*/
Polygon='X'; border_units=10; sq_units=3;
  x=1; y=4; output;
  x=1; y=5; output;
  x=2; y=5; output;
  x=2; y=4; output;

Polygon='B';
  x=1; y=2; output;
```

```
  x=1; y=3; output;
  x=2; y=3; output;
  x=2; y=2; output;

Polygon='C';
  x=1; y=3; output;
  x=1; y=4; output;
  x=2; y=4; output;
  x=2; y=3; output;

 Polygon='D'; border_units=8; sq_units=4;
  x=3; y=2; output;
  x=3; y=3; output;
  x=4; y=3; output;
  x=4; y=2; output;

 Polygon='E';
  x=4; y=2; output;
  x=4; y=3; output;
  x=5; y=3; output;
  x=5; y=2; output;

 Polygon='F';
  x=3; y=1; output;
  x=3; y=2; output;
  x=4; y=2; output;
  x=4; y=1; output;

 Polygon='G';
  x=4; y=1; output;
  x=4; y=2; output;
  x=5; y=2; output;
  x=5; y=1; output;
run;

/*Draw polygons*/
title1 h=2.5 "Example 2.1: Polygons";
proc gmap data=polygons2 map=polygons2;
   id polygon;
   choro polygon / discrete legend=legend1;
 run;
quit;
```

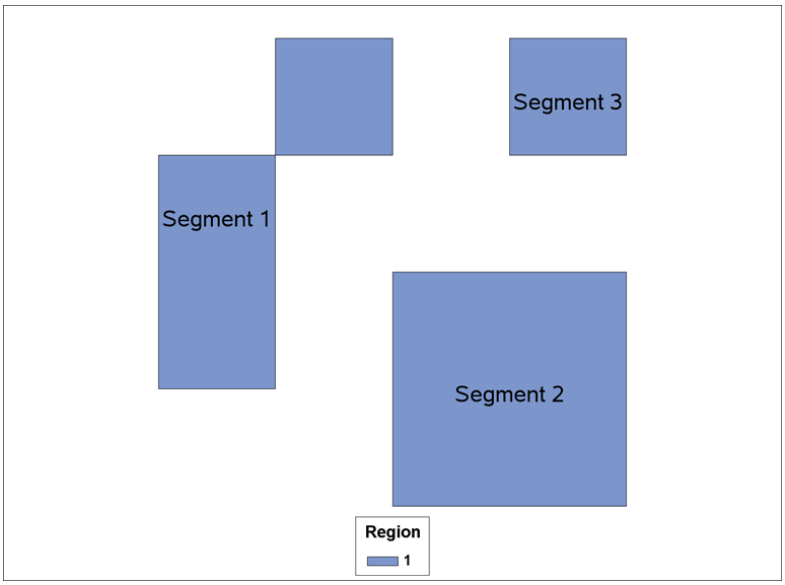
Figure 4. Map of Polygons for Example 2




```
/*Remove inside borders of Region*/
proc gremove data=polygons2 out=area2;
   by Region;
   id polygon;
run;

/*Calculate centroid of the segment coordinates using macro*/
%annomac;
%centroid(area2,seg2_centroids,segment);

/*Create annotation data to label area segments*/
data anno2_labels;
  length color $ 8 text $ 20 style $ 25;
  set seg2_centroids;
  retain x y xsys ysys hsys when position function size color text;
     segname=compress("Segment"||segment);
     segname=tranwrd(segname,"t","t ");
     xsys='2'; ysys='2'; hsys='3'; when='a'; position='5';
     function='label';
     style="'Albany AMT'";
     text=segname;
     color='black';
     size=4;
  output;
run;

title1 h=2.5 "Example 2.2: Region with non-Contiguous Polygons";
proc gmap data=area2 map=area2;
   id Region;
   choro Region / discrete legend=legend1 anno=anno2_labels;
 run;
quit;
```

After removing the polygon borders to create the new area we can see that the GREMOVE procedure assigns polygons A, B, and X to the same segment. With the addition of Polygon X, the boundary size of the segment increases from 6 to 10 units making it the largest boundary size among the three segments. Consequently, a value of 1 is assigned to this segment even though the the segment assigned value 2 has the largest area.

Figure 5. Map of Region with Polygon Borders Removed for Example 2



USING PROC GREMOVE OUTPUT DATA IN A HEALTHCARE SETTING

We now know that the GREMOVE procedure assigns segment values in order of descending boundary size regardless of the size of the area. In example 2, the assignment of value 1 to the segment with the second largest area might seem counterintuitive. After all, the segment with the largest area appears bigger on the map and covers more space. If nothing else, we do know that these segments are not contiguous because a different value is assigned to them. In this section, we extend this application of PROC GREMOVE to the healthcare policy research setting, and discuss whether the value of the segments can be used in a meaningful way.

IMPACT OF PROSPECTIVE PAYMENT SYSTEM (PPS) ON END STAGE RENAL DISEASE (ESRD)

ESRD is a chronic condition that effects more than 660,000 Americans of which 468,000 are dialysis patients. In the absence of a successful kidney transplant, ESRD patients almost always requires dialysis for survival. Since a diagnosis of ESRD is a basis for entitlement for Medicare benefits, most patients might have their dialysis services covered by Medicare. In 2011, Medicare implemented a prospective payment system that established fixed payment rates for dialysis treatments and related services provided by outpatient dialysis facilities. This new payment system replaced the previous fee-for-service approach where dialysis facilities billed Medicare separately for some of the individual services provided (e.g., based on the dose of a particular drug administered). While the new payment system was intended to eliminate the incentive to overuse separately billable services, there is a risk that prospective payment might incentivize under-treatment.

The response of dialysis facilities to this major change in payment might depend on their own attributes, such as their size or organizational affiliation, which might affect their ability to manage their increased risk under the new payment system that their costs for patient care might exceed their revenues. However, the response of facilities to the payment reform might also depend on the characteristics of the dialysis markets in which facilities operate, such as the local demand for dialysis services and the level

and type of competition from other providers. Consequently, evaluations of the impact of the dialysis payment reform might benefit from market-level research.

DEFINING HEALTHCARE MARKETS

Market level research will likely require some level of geography to define the unit of analysis. One crude method is to use units of geography that are already defined. Data for political units of geography are readily available and can be freely obtained from sources such as the US Census Bureau and the Health Resources and Service Administration (HRSA). However, these units of geography by themselves do not necessarily capture areas where the majority of patients live and receive care from their provider. Alternatively, researchers may turn to healthcare service areas already defined by institutions to use as their markets. For example, the Dartmouth Atlas of Healthcare developed health service areas (HSAs) and hospital referral regions (HRRs) based on the volume of hospitalizations in the Medicare population. HSAs and HRRs can be attractive for use as markets in healthcare research since the boundaries were created using data driven techniques and the size of both HSAs and HRRs are larger than counties but smaller than states. However, the boundaries defined based on hospitalizations will not necessarily apply well to the markets for dialysis services. Rather, data driven approaches that are based on dialysis patient flow might result in more accurate market definitions for dialysis services. Based on a treatment regimen that often requires patients to make three visits to a dialysis facility each week, the proximity between a patient's residence and their dialysis facility might be an especially important consideration in this particular healthcare context.

For our approach, we define the areas in which most dialysis patients are treated as dialysis service areas (DSAs). A DSA may contain one or more facilities, which contribute to the level of competition within that area.

METHODS FOR DEVELOPING DSAS

DSAs are generated using ZIP codes from both dialysis patient and dialysis provider (also referred to as facility) addresses. ZIP codes are first mapped to ZIP Code Tabulation Areas (ZCTAs) to obtain better estimates for an area. While some patients are treated at dialysis facilities located within the same ZCTA in which they lived, most others traveled outside to a nearby ZCTA for treatment. By tracking patient flow, we capture the surrounding areas in which patients are usually treated. This is useful in determining the size and location of DSAs in a variety of regions in the United States, which vary by population and geography. However, the drawback of this data driven method is that patient flow does not always result in the joining of ZCTAs that are geographically contiguous by land.

We demonstrate what non-contiguous ZCTAs resemble on a map; we obtain coordinates by downloading the 2010 ZCTA shape file for the United States, which is available with the TIGER/Line data through the US Census Bureau website. The MAPIMPORT procedure is then used to read all variables from the shape file into a SAS data set.

```
proc mapimport datafile=" C:\Users\Documents\tl_2010_us_zcta510.shp"
   out=zcta2010;
run;
```

The example below includes a hypothetical scenario where there are three DSAs formed by neighboring ZCTAs in Southeast Michigan and Northwest Ohio. We will assume that these ZCTAs have already been assigned to DSAs based on the flow of patients from their home address to a dialysis facility where they received their treatment, and one DSA is not contiguous.

```
data ZCTA;
   set zcta2010;
     ZCTA=input(GEOID10,8.);
         if ZCTA in (43560,43606,43613,43615,43617,43623,
```

```
                        48144,48177,48182,49267,49270,49276);

        if ZCTA in (48144,48177,48182,49270,49276) then DSA=1;
        if ZCTA in (49267,43560,43606) then DSA=2;
        if ZCTA in (43613,43615,43617,43623) then DSA=3;

     drop GEOID10;
run;
```

The code below contains the healthcare data associated with each ZCTA. The data are at the ZCTA level and have information on the count of ESRD patient living in the ZCTA and dialysis facilities. We further aggregate the patient and provider counts after grouping ZCTAs to DSAs. Although some ZCTAs are missing patients and/or providers each DSA contains patients and at least one provider.

```
/*Data set with ESRD patient and provider counts by ZCTA*/
data ZCTAsample;
 input ZCTA DSA Patients Providers;
 datalines;
 48144 1 25 1
 48177 1 10 0
 48182 1 15 0
 49270 1 5 0
 49276 1 5 0
 49267 2 0 0
 43560 2 10 0
 43606 2 120 3
 43613 3 15 0
 43615 3 30 1
 43617 3 40 1
 43623 3 15 0
 ;
run;

/*Remove inside borders of DSAs*/
proc gremove data=ZCTA out=DSA;
   by DSA;
   id ZCTA;
run;

/*Aggregate data from ZCTAs to the DSAs level*/
proc sql;
 create table DSAsample
  as select
    DSA, sum(patients) as patients,
      sum(providers) as providers,
      count(distinct ZCTA) as ZCTAs
      from ZCTAsample
      group by DSA
      order by DSA
      ;
quit;

/*Print of DSA aggregate data with patient, provider, and ZCTA counts*/
proc print data=DSAsample noobs;
title "DSA patient, provider, and ZCTA counts";
run;
```

Output 3. DSA Patient, Provider, and ZCTA Counts with a Non-Contiguous Assignment

DSA	patients	providers	ZCTAs
1	60	1	5
2	130	3	3
3	100	2	4

```
/*Calculate centroid of the DSA coordinates using macro*/
/*Calculate coordinates separately for ZCTA 43606*/
data temp z43606;
  set DSA;
    if ZCTA_copy ne 43606 then output temp;
    if ZCTA_copy eq 43606 then output z43606;
run;

/*Used later for labels. Remove inside borders of ZCTAs within a state*/
proc gremove data=ZCTA out=STATE;
   by STATE;
   id ZCTA;
run;

%annomac;
%centroid(temp,DSA_centroids,DSA);
%centroid(z43606,z43606_centroid,DSA);
%centroid(STATE,STATE_centroids,STATE);

/*Create annotation data to label STATE*/
data STATE_labels;
  length color $ 8 text $ 20 style $ 25;
  set state_centroids;
  retain x y xsys ysys hsys when position function size color text;
    xsys='2'; ysys='2'; hsys='3'; when='a'; position='5';
    function='label';
    style="'Albany AMT'";
    if STATE="MI" then do; statename="MICHIGAN"; y=y-0.04; x=x-0.02; end;
    if STATE="OH" then do; statename="OHIO    "; y=y+0.03; x=x-0.02; end;
    text=statename;
    color='gray';
    size=7;
run;

/*Create annotation data to label DSA names*/
data DSA_labels;
  length color $ 8 text $ 20 style $ 25;
  set DSA_centroids z43606_centroid;
  retain x y xsys ysys hsys when position function size color text;
    DSAname=compress("DSA"||DSA);
    DSAname=tranwrd(DSAname,"t","t ");
    xsys='2'; ysys='2'; hsys='3'; when='a'; position='5';
    function='label';
    style="'Albany AMT'";
    text=DSAname;
    color='black';
    size=4;
run;
```

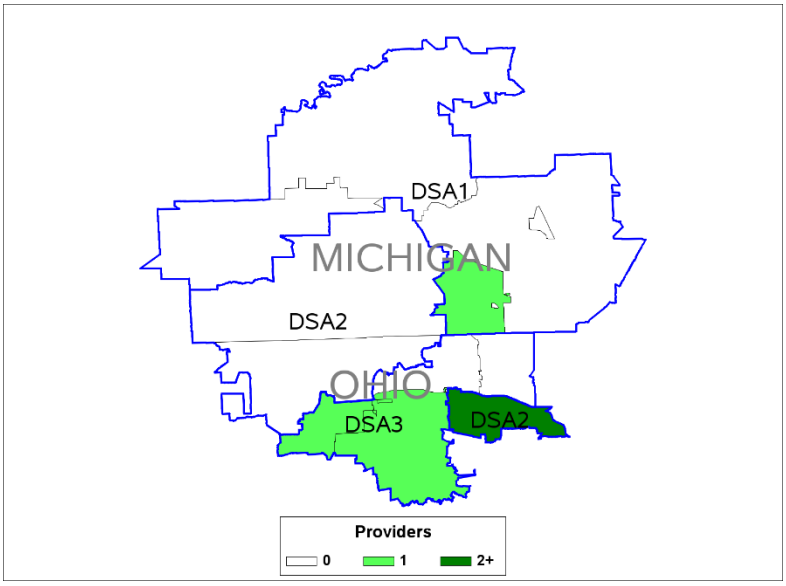
```
proc sort data=DSA; by DSA segment; run;

/*Create annotation data to overlay DSA boundaries*/
data DSA_border;
  length function color $8;
  retain xsys ysys '2' when 'a' color 'black' size 3.5;
  set DSA;
     by DSA segment;
      if first.segment then function='poly';
      else function='polycont';
  output;
run;

/*Combine label and border data into the same annotation data set*/
data DSA_anno;
      set DSA_border DSA_labels STATE_labels;
run;
```

In the previous examples, the GMAP procedure used the same data set for the input used in the map and coordinates to draw the map. In this example, we use the data set containing our provider counts and a separate data set with coordinates to draw the map. In addition, we can use the FORMAT procedure to both categorize and format the information for provider counts so that data can be labelled on the map.

```
proc format;
 value prov
   0='0 '
   1='1 '
   2-high='2+'
 ;

pattern1 value=solid color='white';
pattern2 value=solid color='light green';
pattern3 value=solid color='green';

proc gmap data=ZCTAsample map=ZCTA;
  id ZCTA;
  choro providers /discrete legend=legend1 anno=DSA_anno;
  format providers prov.;
 run;
quit;
```

After the GREMOVE procedure is applied, two of the three DSAs are contiguous, but one DSA contains two ZCTAs that are not adjacent to each other. The DSA that is not contiguous can be easily identified by detecting any segment values greater than 1. As we have seen in the previous examples, the segment that is assigned a value of 1 has the largest surrounding boundary.

Figure 6. Map of ZCTAs Grouped by DSAs for Example 3



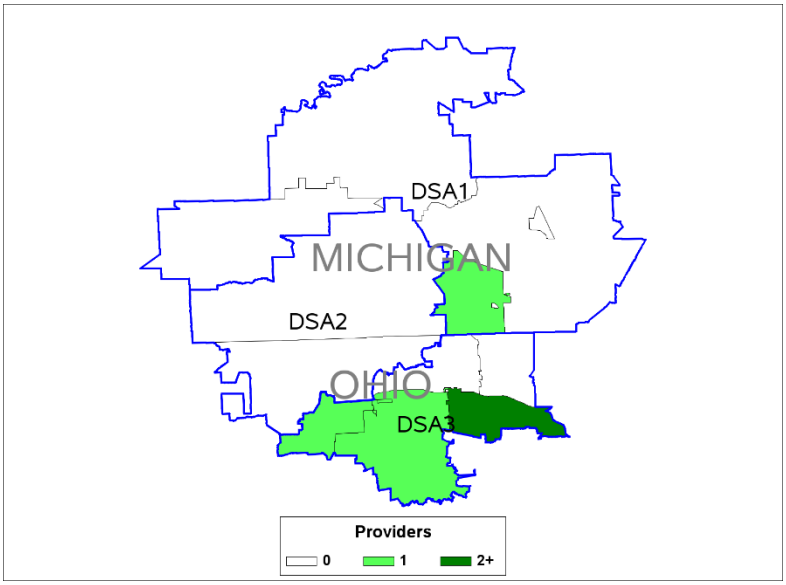
To correct this map for lack of contiguity we would like to retain most of the ZCTA assignments and the information within their respective DSAs. We can do this by reassigning low priority segments in non-contiguous DSAs to neighboring contiguous DSAs. We define low priority segments as those with a value greater than 1. In this case, segment 2 in DSA 2, which has a smaller surrounding boundary will be reassigned to DSA 3 since it is the closest neighboring DSA. We make this change in our original ZCTA data, and rerun the GREMOVE procedure.

After aggregating the data within the new DSAs we see that the patient and provider counts have been impacted. Specifically, we notice that the patient and provider counts within DSA 3 have increased, and DSA 2 has been reduced to having just ten patient residents and no providers. This is concerning, especially for DSA 2, since we would intend on having at least one provider and several patients living in an area for consideration of a market for dialysis services.

Output 4. DSA Patient, Provider, and ZCTA Counts if Segment with Small Boundary Size is Reassigned

DSA	patients	providers	ZCTAs
1	60	1	5
2	10	0	2
3	220	5	5

Figure 7. Map of ZCTAs Regrouped into DSAs if Segment with Small Boundary Size is Reassigned



Logically, if there are no underlying data for the areas we might think of prioritizing an area based on the size. However, since the areas we are trying to develop should also contain meaningful data, we modify our definition of low versus high priority segments. In our case, segments are prioritized based on the following data:

1. **Number of providers:** For a legitimate definition of a market, there should be at least one provider, if not more. DSA segments with the fewest number of providers should be reassigned first.

2. **Number of patient residents:** If there is an equal number of providers between different DSA segments then reassign the segment with the fewest patients living in that area.

3. **Number of ZCTAs:** Having a sufficient number of providers and patients is most important for a market definition. However, as a tiebreaker, we would prefer to reassign the DSA segment with fewer ZCTAs in order to conserve the size and shape of the original DSAs.

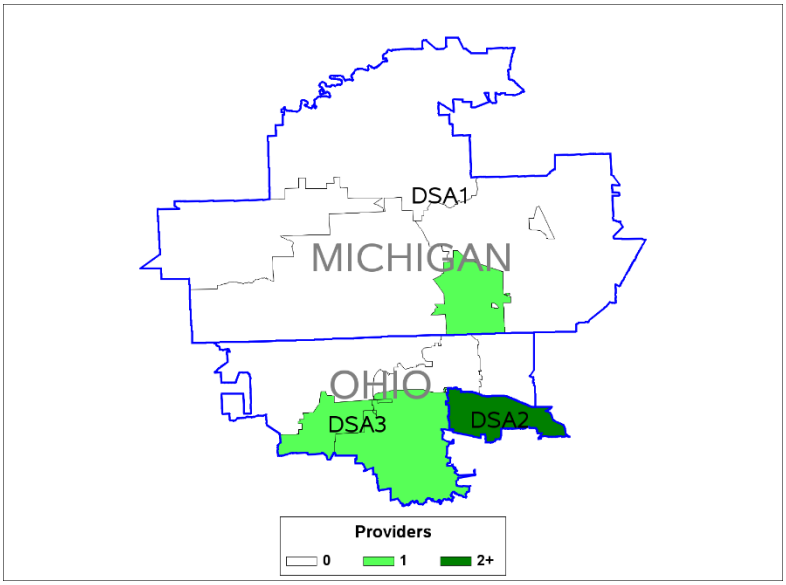
Using these three criteria we reassign the segment values in DSA 2. Consequently, the low priority segment based on our criteria happens to be the one with the largest surrounding boundary. Since this segment consists of two ZCTAs the reassignment is split between neighboring DSA 1 and neighboring DSA 3. With the new assignment we can see only slight changes to the patient resident counts and no changes to provider counts in any DSAs.

Output 5. DSA Patient, Provider, and ZCTA Counts if Segment with Low Provider Count is Reassigned

DSA	patients	providers	ZCTAs
1	60	1	6
2	120	3	1
3	110	2	5

The final map shows our new DSAs. The tradeoff for minor changes to the original shape and size of areas and minor changes to patient resident and provider counts is a nice clean representation of areas that are contiguous.

Figure 8. Map of ZCTAs Regrouped into DSAs if Segment with Low Provider Count is Reassigned



CONCLUSION

The GREMOVE procedure can be a handy tool for generating custom map boundaries to be used in the GMAP procedure by removing the internally shared boundaries of smaller unit areas. Understanding how the GREMOVE procedure assigns values to segments might be useful in practical settings. First, the presence of more than one segment indicates that an area is non-contiguous. Second, segment values are assigned based on boundary size, and can be used to distinguish smaller secondary non-contiguous segments from one larger primary segment. However, as shown in this paper these segment values may be reordered based on a different set of criteria that might be more appropriate based on the underlying data. Each of these steps and the additional process to reassign segments to neighboring areas is a potentially useful method for maintaining contiguity when developing market service areas for research.

REFERENCES

Eberhart, Michael. “Make the Map You Want with PROC GMAP and the Annotate Facility” Philadelphia Department of Public Health. Accessed July 5, 2017. http://www.lexjansen.com/nesug/nesug08/np/np05.pdf

SAS Institute Inc., SAS/GRAPH® *Software: Reference, Version 8*, Cary, NC: SAS Institute Inc., 1999. Accessed August 2, 2016. http://www.okstate.edu/sas/v8/saspdf/gref/c25.pdf

SAS Institute Inc., SAS/GRAPH® 9.4: Reference, Fifth Edition. %CENTROID Macro. Accessed July 5, 2017. http://support.sas.com/documentation/cdl/en/graphref/69717/HTML/default/viewer.htm#p10hx3f0vlu6q3n1k29xnhm1bflx.htm

The Dartmouth Atlas of Health Care 1999. Appendix on the Geography of Health Care in the United States. Accessed November 23, 2015. http://www.dartmouthatlas.org/downloads/methods/geogappdx.pdf

U.S. Renal Data System Annual Data Report (2015). Accessed July 5, 2017. https://www.usrds.org/2015/view/Default.aspx

Zdeb, Mike. “The Basics of Map Creation with SAS/GRAPH®” University of Albany School of Public Health. Accessed June 17, 2017. http://www2.sas.com/proceedings/sugi29/251-29.pdf

ACKNOWLEDGEMENTS

This research was supported by Award Number R01MD006247 from the Health National Institute on Minority Health and Health Disparities of the National Institutes of Health (NIH/NIMHD). The content is solely the responsibility of the authors and does not necessarily represent the official views of the NIH or NIMHD.

CONTACT INFORMATION

Author: Chad Cogan, MS
Organization: Arbor Research Collaborative for Health
Email: chad.cogan@arborresearch.org
Address: 340 E Huron Street Suite 300
Ann Arbor, MI 48104