Paper PO-18

Winning the War on Terror with Waffles: Maximizing GINSIDE Efficiency for Blue Force Tracking Big Data

Troy Martin Hughes, in support of the Joint IED Defeat Organization (JIEDDO)

ABSTRACT

The GINSIDE procedure represents the SAS solution for point-in-polygon determination. The procedure requires three parameters—a map data set representing the polygon, an input data set containing geographic points, and a list of ID fields (i.e., attributes) that are conferred to an observation when point-in-polygon status is determined. The most significant factor predicting longer GINSIDE runtime or procedure failure is file size of the input data set. When big data—having either a large number of fields or observations, or both—are encountered, the GINSIDE procedure operates inefficiently or fails entirely. The traditional GINSIDE procedure, thus, was unacceptable to analyze millions of records of the Department of Defense (DoD) Blue Force Tracking (BFT) data. To improve point-in-polygon determination efficiency and prevent GINSIDE errors, a method is demonstrated that substantially reduces the number of observations required to be calculated through the GINSIDE procedure. The map data set is overlaid by a matrix (i.e., waffle schema) of rectangles, each of which is wholly inside the polygon. By determining whether a point falls between the vertices of a rectangle, polygon inclusion is determined through a much simpler and efficient conditional logic algorithm than required by GINSIDE. Only the few remaining points that fall very close to polygon borders must be interpreted through the GINISIDE procedure, thus big data now can be processed with efficiency and without error. In addition to debuting waffle technology, this paper describes two critical errors in the SAS v9.3 implementation of the GINSIDE procedure, neither of which are corrected or documented by SAS as of this publication.

INTRODUCTION

The GINSIDE[1] procedure answers the classic point-in-polygon question: given a simple polygon—one for which its perimeter does not intersect itself—does a point fall inside, outside, or on the boundary? For points falling inside the polygon, attributes—i.e., one or more fields within the map data set—are appended to the observation in the output data set to indicate polygon inclusion. While SAS does not disclose its point-in-polygon algorithm, common computational methods such as ray casting are an iterative and timely process and, when applied to big data, can present an efficiency challenge to the user or cause process failure.

The Department of Defense (DoD) Blue Force Tracking (BFT) system provides real-time situational awareness and historic data analysis to troops and analysts through an integrated command and control geographic information system (GIS) interface[2]. Global positioning system (GPS) units mounted on vehicles or aircraft or carried by troops demonstrate location and movement and have become a staple of military operations. A single unit that beacons once per second will produce over 31 million observations per year. When the number of units in Afghanistan and the length of the US campaign are considered, the billions of BFT observations present a voluminous analytic challenge.

All GPS data minimally include latitude, longitude, and transmission time of the device; other attributes must be inherited from external sources such as map or feature data sets. Because BFT data do not inherently contain country or province information, the Joint Improvised Explosive Device Defeat Organization (JIEDDO) utilizes GINSIDE to attribute country, province, and other geospatial information to BFT data. Given the enormity of BFT data, a more efficient point-in-polygon determination methodology was required and was developed to reduce runtime and to extend the number of observations that could be processed through a single GINSIDE procedure. The improved methodology analyzes and stores information about a map data set in a SAS data set called a waffle schema. For map data sets that are frequently utilized, “waffles” improve GINSIDE performance much like website cookies improve the navigation and performance of web pages that are visited repeatedly.

GINSIDE PERFORMANCE FACTORS

GINSIDE has three inputs: the map data set of the bounded polygon, the ID fields of the polygon to be attributed to observations, and the data set containing observations to be analyzed for inclusion or exclusion. In seeking to build a more efficient GINSIDE process, the first step was to analyze runtime to determine which factors are most influential. Variables analyzed included the number of observations in the data set, the number and type of fields, and file size. The macro %GINSIDE_PERFORMANCE (see Appendix A) first creates a test data set then iterates through thousands of GINSIDE scenarios varying the above independent variables. The macro invocation below demonstrates one scenario that will produce (and measure performance of) 270 independent GINSIDE procedures:

```
%macro GINSIDE_PERFORMANCE (map=test
   DataRowIter=10000,
   DataRowMax=100000,
   DigFieldsIter=10,
   DigFieldsMax=20,
   Char10Iter=5,
   Char10Max=10,
   Char100Iter=5,
   Char100Max=10,
   Char1000Iter=0,
   Char1000Max=0,
   Char32000Iter=0,
   Char32000Max=0);
```

This invocation will utilize the SAS map data set TEST and create fictional data that occur roughly inside the defined polygon of the map. The 270 GINSIDE procedures ([(100,000 / 10,000)] x [(20 / 10) + 1] x [(10 / 5) + 1] x [(10 / 5) + 1]) are iteratively looped through until macro completion. Because of the factorial nature of the total number of iterations produced, care should be utilized before execution to ensure that a realistic number of procedures is being created.

The field types were chosen to represent data typically encountered at JIEDDO, including numeric data and character data of varying lengths. Because some data sets do include verbose text fields, two long fields containing 1,000 and 32,000 characters, respectively, were included. This code was utilized to run thousands of GINSIDE procedure iterations and, when the number of observations, number and type of field, and file size were regressed against GINSIDE procedure runtimes, file size was most predictive of runtime. This result was expected given that the GINSIDE procedure requires both a complete reordering of the input data set as well as a complete rewrite of the output data set, both of which are I/O intensive processes. **Thus, any reduction in file size—due either to a reduction in the number of observations or to the removal of gratuitous fields—improves the performance of the GINSIDE procedure.**

DATA SET REDUCTION: FEWER OBSERVATIONS

The first method to improve GINSIDE efficiency is to reduce the number of observations that require the GINSIDE procedure, by instead applying a simpler, more efficient algorithm to determine point-in-polygon inclusion. The waffling technique that was developed produces a waffle schema—a matrix of rectangles of decreasing size in which each rectangle is wholly inside the original polygon. Waffles improve efficiency by reducing substantially the number of observations that must pass through the traditional GINSIDE procedure. Instead, because the polygon inclusion status of each rectangle is known *prima facie*, the waffle acts as a giant lookup table that can confer point-in-polygon status much more rapidly than the complex GINSIDE algorithms.

While the waffle technology was developed to solve a BFT big data processing conundrum, it can be implemented successfully with any SAS map data set having one or more ID fields. For demonstration purposes and for ease of recognition, the map of the Continental United States is used hereafter as opposed to the map of Afghanistan. The processes that create, interpret, and utilize waffles are automated and internal to the macro WAFFLE_EATIN (see Appendix C), but for conceptual demonstration, these processes are delineated hereafter.

MAKIN' WAFFLES

The first step of making a waffle is to determine the boundaries of the map data set, by identifying the minimum and maximum coordinate values. Any points falling outside of these boundaries are immediately determined to be outside the polygon, thus eliminating the need for these values to be processed with the GINSIDE procedure. In other words, you don't eat the batter that pours over the edge of the waffle iron—it's crusty and best fed to ducks. This step is illustrated in Figure 1, in which a black outline has been drawn around the minimum and maximum values of the Continental United States; geographic coordinates that fall outside of this box should not be considered and will not receive any attributes in the output data set:

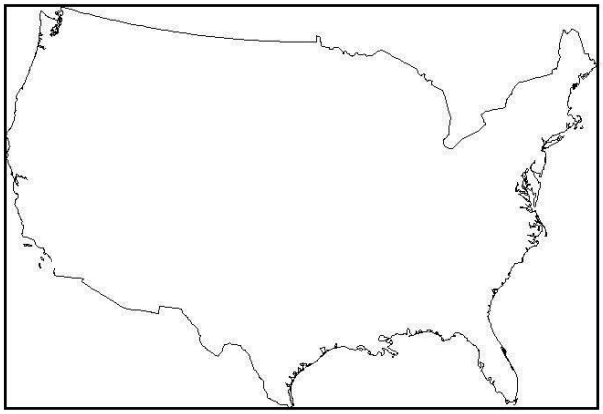

Figure 1. Waffling: Rectangular region bounding Continental United States

The first iteration begins by dividing the initial bounded rectangle into four equally sized rectangles. If any of these subsequent rectangles is wholly inside the polygon, its coordinates are added to the waffle schema; conversely, if a rectangle is wholly outside the polygon, it is removed from further analysis. The first iteration that divides the Continental United States is demonstrated in Figure 2 and, because none of the resultant rectangles are wholly inside or outside the polygon, each rectangle will continue into the next iteration:

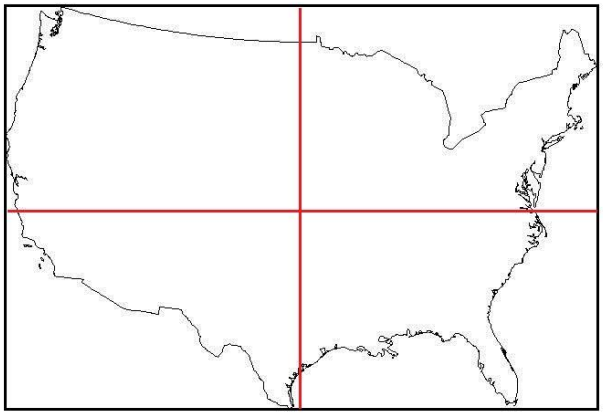

Figure 2. Waffling: First iteration (i.e., division) of the Continental United States

The second iteration divides each of the four rectangles into four more equally sized rectangles, which produces 16 rectangles, two of which are wholly inside the polygon. The minimum and maximum X and Y values of each of these rectangles is output to the waffle schema, as well as any associated ID values. In this example, the only variable COUNTRY identifies whether a coordinate falls inside the Continental United States, so COUNTRY equals US for the two rectangles in the waffle schema. The remaining 14 rectangles all partially fall within the polygon, so they pass to the next iteration. Figure 3 depicts iteration two and the two highlighted rectangles:

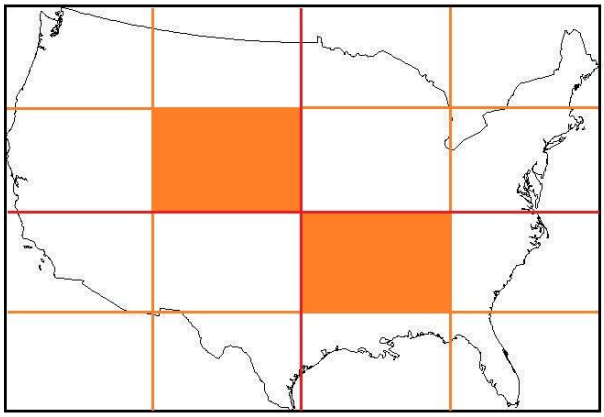

Figure 3. Waffling: Second iteration (i.e., division) of the Continental United States

The third iteration ignores the two rectangles that are wholly inside the polygon, and divides the remaining 14 rectangles each into four smaller rectangles, producing a total of 56 smaller rectangles. Of these 56, 13 rectangles (highlighted in Figure 4) are wholly inside the polygon, each of which is added to the waffle schema. Seven additional rectangles are also determined to be wholly outside the polygon and, while these regions are not added to the waffle schema, they are eliminated from further processing and iterations. Figure 4 highlights the first two interior rectangles identified through iteration two, and the 13 additional interior rectangles identified through iteration three:

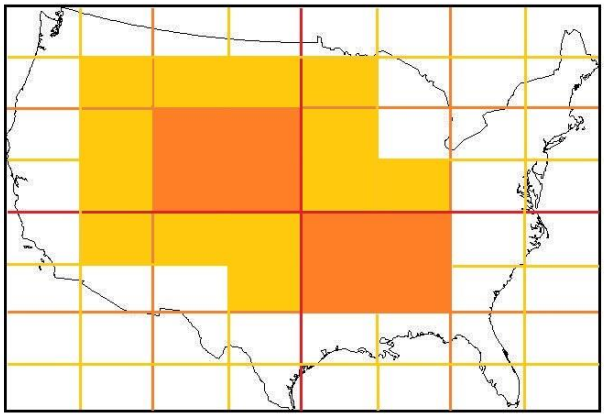

Figure 4. Waffling: Third iteration (i.e., division) of the Continental United States

In iteration four (not depicted), the remaining 36 rectangles (56 total minus 13 interior rectangles minus 7 exterior rectangles) will be further divided into four smaller rectangles, and reassessed. These iterations continue until the parameterized value MAXITER (in the WAFFLE_MAKIN macro) is reached. With each iteration, the highlighted region formed from interior rectangles increases in size and specificity, more closely representing the original polygon and more closely approximating the total area of the polygon.

The WAFFLE_MAKIN macro (see Appendix B) never is required to be called directly, because it is called implicitly through the WAFFLE_EATIN macro if a requested schema cannot be located. Nevertheless, in order to understand the process, the WAFFLE_MAKIN parameters follow:

```
%macro WAFFLE_MAKIN(lib= /* library in which the SAS map data set is located */,
   map= /* SAS map data set for which waffles are being created */,
   id= /* space-delimited list of variables used to group map data set, which
          MUST BE IN IDENTICAL ORDER AS MAP DATA SET IS SORTED! */,
   maxiter= /* the maximum number of iterations (rectangle divisions) */);
```

For example, the following statement would generate the schema depicted in Figure 4:

```
%macro WAFFLE_MAKIN(lib=maps,
   map= america,
   id= country,
   maxiter= 3);
```

This invocation describes that the SAS library MAPS includes the map data set AMERICA, representing the outline of the Continental United States. *If the map data set also included state divisions demarcated by the field STATE, the ID utilized in the macro call would be "country state" to include both levels of boundaries.* The MAXITER parameter describes the number of iterations (i.e., division into smaller rectangles) that will occur to produce the waffle schema. The schema represented in Figure 4 is produced as an HTML report (maps_america_3.htm) in the directory of the WAFFLES library, and the SAS data set (waffles.maps_america_3) referred to as the waffle schema is produced in the same library. The HTML report is useful to demonstrate the approximate area of coverage that a specific level of a schema affords, and the SAS data set is the actual schema that is utilized by the macro WAFFLE_EATIN to perform efficient GINSIDE processing.

In practice, waffle schemas containing fewer than five iterations are useless because they do not cover sufficient area of a polygon to be more efficient than the traditional GINSIDE procedure. That said, if the schema depicted in Figure 4 were being utilized and the geographic points in the input data set fell predominantly in the center of the United States, the waffle process would outperform the GINSIDE procedure. This is to say that only by examining both the waffle schema and the input data set together can the optimal iteration level be established.

Also in practice, waffle schemas greater than 8 iterations tend to show diminishing returns in efficiency gains, or actually perform less efficiently than the traditional GINSIDE procedure. This occurs because at higher iterations, the addition of each incremental iteration can produce tens of thousands of more tiny rectangles. While the area coverage will more closely approximate the polygon, each of those tens of thousands of rectangles adds an additional several lines of conditional logic that SAS must process. Once a specific iteration level has been identified as being the most efficient method to determine point-in-polygon inclusion, as defined by faster runtimes than the next higher and lower iteration values, that iteration level should be utilized exclusively for that combination of the map data set and input data set for future analyses. **Note that the SAS library "Waffles" must be established prior to invocation of the WAFFLE_MAKIN and WAFFLE_EATIN macros.**

EATIN' WAFFLES

The only thing more rewarding than making waffles…is eating waffles! Despite the one-time overhead involved in creating a waffle schema, if a map data set is accessed repeatedly and the input data set is large enough, waffles will always provide a more efficient point-in-polygon calculation. Moreover, because the GINSIDE procedure inherently fails when big data are encountered (see GINSIDE ERROR below), utilizing waffle technology will expand the data sets and scenarios to which point-in-polygon determination can be established. The declaration for the WAFFLE_EATIN macro follows:

```
%macro WAFFLE_EATIN (lib= /* library in which the SAS map data set is located */,
   map= /* SAS map data set for which waffles are being created */,
   tab= /* SAS data set that contains coordinate values X and Y */,
   outtab= /* SAS data set that is output, containing appended ID fields */,
   id= /* the ID variable (or space-delimited list) used to group map data set */,
   maxiter= /* the maximum number of iterations (rectangle divisions) */);
```

For the following examples, consider the data set DOTS that contains 1,000,000 observations representing geographic coordinates in North America. An analyst wishing to determine which specific observations have coordinates occurring within the Continental United States (see Figure 1) could perform the following traditional GINSIDE procedure:

```
proc ginside data=dots map=maps.america out=results;
   id country;
```

This would produce an output data set RESULTS in which all coordinates falling within the bounds of the polygon MAPS.AMERICA received the value of the attribute COUNTRY (i.e., "US") from the map data set. The following macro invocation would generate the same data set RESULTS, albeit with improved efficiency because only those points falling very close to the polygon borders would need to be assessed through the traditional GINSIDE procedure:

```
%macro WAFFLE_EATIN (lib=maps,
   map= america,
   tab= dots,
   outtab= results,
   id= country,
   maxiter= 2);
```

In an actual scenario, the MAXITER parameter should be set to a whole value near 8; however, in this example, 2 is utilized because it can be displayed visually and calculated mathematically more easily. The macro first determines whether the waffle schema waffles.maps_america_2 exists as a SAS data set; if it does not exist, the schema is created by invoking the WAFFLE_MAKIN macro. To improve readability in this example, the following simplified X-Y coordinates (representing geographic coordinates of rectangle vertices) have been added to Figure 5 below:

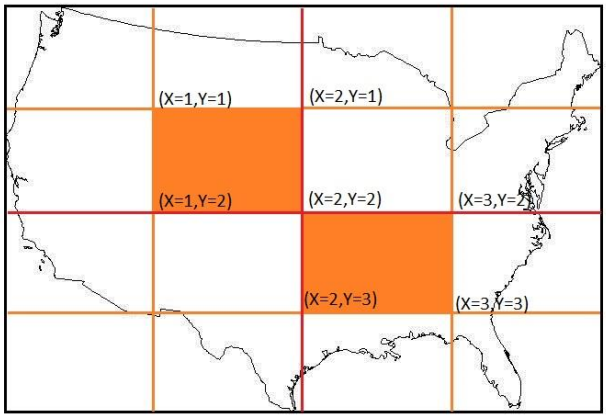

Figure 5. Visualization of sample coordinates for waffles.maps_america_2

The schema in Figure 5 would be represented as two observations in the data set waffles.maps_america_2, as depicted in Table 1:

X1	X2	Y1	Y2	Country
1	2	1	2	US
2	3	2	3	US

Table 1. Observations in waffle schema waffles.maps_america_2

Because the map data set MAPS.AMERICA has only one ID field (COUNTRY), only this field appears in Table 1, and every value will be identical. *If a second ID field STATE were present, this field also would be appended to the data, and its respective values would contain the name of the state in which the coordinate point for the observation fell.* As the schema is ingested, it is translated into conditional logical code stored in macro variables &code1 through &code[*N*] that will be applied later to the input data set. For example, the data in Table 1 would be translated into the following code, which would be stored as macro variable &code1:

```
if 1 <= X <= 2 and 1 <= Y <= 2 then do;
   country='US';
   output;
   end;
else if 2 <= X <= 3 and 2 <= Y <= 3 then do;
   country='US';
   output;
   end;
```

Because of the length of the conditional logic code in schemas that have tens of thousands of defined rectangles, the &code macro variable increments to create a new macro variable after each 150 observations (i.e., rectangles) in the schema. Once the conditional logic code has been created, it is executed from within a data step that ultimately creates the output data set:

```
data results;
   &code1;
   &code[N];
   else output unknown;
run;
```

The schema in essence acts as a giant sieve, filtering observations having coordinates easily determined through conditional logic statements directly into the output data set. While the schema waffles.maps_america_2 represents only two iterations—which would never in actuality be utilized because it covers such a sparse area of the polygon—higher level schemas commonly can cover 95 percent or more of the area of a map. Thus, for a point data set that is geographically equally distributed, at least 95 percent of the points will receive polygon determination without the necessity of the GINSIDE procedure. In this example, only the remaining five percent of observations—those that lie near the polygon borders—would require the GINSIDE procedure, and would be appended to the original observations in the RESULTS data set.

DATA SET REDUCTION: FEWER VARIABLES

As mentioned, traditional GINSIDE runtime is most closely correlated with file size, which is influenced by both number of observations and number and type of fields. While the waffle technique presented above dramatically can reduce the number of observations required to be processed through the traditional GINSIDE procedure, this technique will still be inefficient if the data set contains numerous or large gratuitous fields—any fields that neither represent a primary key nor a geographic coordinate. While a few meager gratuitous fields will not hamper point-in-polygon processing, the overhead processing of these fields should be considered in every calculation. Consider that a 10,000 observation data set with the minimal X and Y coordinates and a numeric primary key will create an approximately 300KB SAS file. The addition of only five 32,000 character text fields to that original data set will cause the file size to increase roughly 5,000 times! Because the GINSIDE procedure requires both a sort and rewrite of the input data set, significant time can be wasted when numerous or large gratuitous fields are not removed before the GINSIDE procedure.

Removal of fields, however, typically implies a later reconstitution of the data set; gratuitous as these fields may be to geospatial processing, they no doubt do serve some other purpose. And, especially where big data are involved, reconstitution of a data set that has had a significant number of fields sheared off into a temporary data set can be a resource intensive operation, at times rivaling or exceeding process time required to GINSIDE the original, intact data set. Because the GINSIDE procedure reorders observations in the output data set, a unique key also must exist in both the data set that is being processed by GINSIDE as well as the temporary data set that contains the remaining fields. Reconstitution can be expedited through a hash join or by an index, but both of these methods also require some overhead at the outset to produce later efficiency gains. For some data sets, the sheer file size will dictate that gratuitous fields must be removed before GINSIDE and later reconstituted because of errors inherent in the GINSIDE procedure when processing big data (see GINSIDE ERROR below); in other cases, however, the user will need to determine whether it is more efficient to process the original data set, or whether to remove fields, perform the GINSIDE, and reconstitute thereafter.

MAP DATA SET REDUCTION: FEWER POINTS

To a far lesser extent than data set file size, the runtime of the GINSIDE procedure will vary based on the complexity (i.e., number of observations) of the map data set, with smaller map data sets processing more rapidly. The GREDUCE[3] procedure reduces the number of points that define the polygon, which will cause the GINSIDE procedure to execute quicker. The cost of this speed, however, is reduced accuracy and for many applications GREDUCE is not a viable option. If utilized with BFT data, for example, the introduction of GREDUCE might attribute a geographic point as being in the wrong district or province in Afghanistan, or even as belonging to a neighboring country. If some degree of accuracy can be sacrificed for improved execution, however, the GREDUCE procedure may be an alternative to consider.

GINSIDE ERROR: OUTPUT DATA SET HAS FEWER OR ZERO OBSERVATIONS

At least two critical errors exist in the SAS v9.3 implementation of the GINSIDE procedure, neither of which has been corrected or documented by SAS as of this publication. The first error occurs when the map data set is “too large” to process, although the SAS log will erroneously depict a successful execution and an appropriate runtime. Upon inspection, however, the output data set specified in the OUT parameter of the procedure will contain either zero observations or far fewer observations than the original data set. Most vexing is that neither the SAS log nor the automatic macro variable &SYSERR report any execution errors. No specific threshold appears to exist after which GINSIDE ceases to work; however, as file size increases past two and three gigabytes, these inexplicable execution failures become commonplace.

A best practice when using the GINSIDE procedure, thus, is always to perform an observation count of the output data set to ensure it is consistent with the input data set. If the error persists, the file size must be reduced, either by reducing the number of fields or the number of observations (e.g., through the waffle technique.)

GINSIDE ERROR: FAILURE WHEN X-Y COORDINATES ARE MISSING

Also not mentioned in the GINSIDE reference material is the fact that all X and Y coordinate values must be present in the input data set to avoid confounding, erroneous results. When the GINSIDE procedure encounters an observation for which either the X or Y value is missing, it performs the following incorrect operations:

- The ID value of the observation that is missing a coordinate is retained from the previous observation; it should instead be given a null or missing value.
- The next immediate observation is deleted from the output data set, regardless of coordinate validity.

This error can be demonstrated with the following code, which generates a map data set MAP (i.e., a square) and performs the GINSIDE procedure against this polygon using the generated data set TEST having 10 observations:

```
data map; /* creates a simple polygon */
   infile datalines delimiter=',';
   input x y inside $;
datalines;
0,0,inside
0,1,inside
1,1,inside
1,0,inside
0,0,inside
;

data test; /* generates 10 points that all fall inside the above polygon */
   length x y i 8;
   format x y 12.10;
   do i= 1 to 10;
          call streaminit(123);
          x=rand('uniform');
          y=rand('uniform');
          if i=8 then x=.; /* deletes the X coordinate in the 8th observation */
          output;
          end;

proc ginside data=test map=map out=testout;
   id inside;
run;
```

The above code generates an output data set TESTOUT that has only nine observations—not 10—and which erroneously ascribes the attribute 'inside' to observation eight (which should be given no attribute, because X is blank.) The actual ninth observation is deleted, which should have been given the attribute 'inside'. *Even more alarming, if the last observation of the input data set is missing either the X or Y coordinate, the GINSIDE procedure appends the entire map data set onto the test data set!*

The second best practice when utilizing the GINSIDE procedure, thus, is to delete all observations for which either the X or Y coordinate is missing. This eliminates the errors described above, although a thorough quality control process will not only delete observations having missing values but, moreover, will delete coordinate values that fall outside of the range of acceptable X and Y coordinates.

CONCLUSION

With a little finesse, the GINSIDE procedure can be a powerful tool in solving point-in-polygon determination. Its major limitation is its inability innately to handle big data, in addition to at least two critical internal errors that can cause disastrous, unexpected results. The introduction of the waffle methodology described herein not only ensures that the GINSIDE procedure produces more efficient results, it moreover allows significantly larger data sets to be processed, by reducing the number of observations that must be processed using the traditional GINSIDE procedure. The robustness and portability of the WAFFLE_MAKIN and WAFFLE_EATIN macros ensure that with relative ease and no additional coding, any SAS map data set can be transformed into delicious, ready-to-eat waffles!

APPENDIX A. GINSIDE_PERFORMANCE MACRO

```
* GINSIDE_PERFORMANCE macro tests the efficiency (as measured in runtime) for data;
* sets of varying dimensions, including observations and type and number of variables;

%macro GINSIDE_PERFORMANCE (map= /* map data to be used in GINSIDE procedure */,
       id= /* ID field to be used in GINSIDE procedure */,
       DataRowIter= /* obs by which each successive data set will be incremented */,
       DataRowMax= /* max obs in SAS data set to which GINSIDE is applied */,
       DigFieldsIter= /* iteration value for number of numeric variables */,
       DigFieldsMax= /* max number of numeric variables */,
       Char10Iter= /* iteration value for number of 10-character variables */,
       Char10Max= /* max number of 10-character variables */,
       Char100Iter= /* iteration value for number of 100-character variables */,
       Char100Max= /* max number of 100-character variables */,
       Char1000Iter= /* iteration value for number of 1000-character variables */,
       Char1000Max= /* max number of 1000-character variables */,
       Char32000Iter= /* iteration value for number of 32000-character variables */,
       Char32000Max= /* max number of 32000-character variables */);
%local i;
%local j;
%local k;
%local l;
%local m;
%local n;
%local o;
%local err;
%local tot;
%local filesize;
%local Gerr;
%local Gobs;
* create values for long and short integers;
%let char10=0123456789;
%let char100=0123456789012345678901234567890123456789012345678901234567890123456789012
       3456789012345678901234567890;
%let char1000=;
%do i=1 %to 10;
       %let char1000=&char1000.0123456789012345678901234567890123456789012345678901234
              5678901234567890123456789012345678901234567890;
       %end;
%let char32000=;
%do i=1 %to 320;
       %let char32000=&char32000.0123456789012345678901234567890123456789012345678901234
              3456789012345678901234567890123456789012345678901234567890;
       %end;
* find min and max X Y values from the MAP data set for random number generator;
data _null_;
       set gis_af.af_districts end=eof; /* map data set used in this example */
       retain minx maxx miny maxy;
       if missing (minx) or x<minx then minx=x;
       if missing (maxx) or x>maxx then maxx=x;
       if missing(miny) or y<miny then miny=y;
       if missing (maxy) or y>maxy then maxy=y;
       if eof then do;
              call symput ('minx' ,minx);
              call symput ('maxx' ,maxx);
              call symput ('miny' ,miny);
              call symput ('maxy' ,maxy);
              end;
       run;
* initialize the table used to collect statistics and compute number of iterations;
data statistics (compress=yes); /* output data set that records metrics */
```

```
      length Data_Observations filesize Data_NumericFields Data_Char10Fields
             Data_Char100Fields Data_Char1000Fields Data_Char32000Fields
             beforeFile afterFile secdiffFile beforeGINSIDE AFTERGINSIDE
             secdiffGINSIDE GINSIDEobs 8 Map_Table GINSIDEerr $50;
      format beforeFile afterFile beforeGINSIDE afterGINSIDE datetime17.
             filesize 15.0;
run;
%if &dataRowIter=0 or &DataRowMax=0 %then %do;
      %let err=Neither DATAROWITER nor DATAROWMAX parameters can equal zero.;
      %goto err;
      %end;
%else %let tot1=%sysevalf(&DataRowMax/&DataRowIter);
%if &digFieldsIter=0 or &DigFieldsMax=0 %then %do;
      %let tot2=1;
      %let DigFieldsIter=1;
      %let DigFieldsMax=0;
      %end;
%else %let tot2=%sysevalf((&DigFieldsMax+&DigFieldsIter)/&DigFieldsIter);
%if &Char10Iter=0 or &Char10Max=0 %then %do;
      %let tot3=1;
      %let Char10Iter=1;
      %let Char10Max=0;
      %end;
%else %let tot3=%sysevalf((&Char10Max+&Char10Iter)/&Char10Iter);
      %if &Char100Iter=0 or &Char100Max=0 %then %do;
      %let tot4=1;
      %let Char100Iter=1;
      %let Char100Max=0;
      %end;
%else %let tot4=%sysevalf((&Char100Max+&Char100Iter)/&Char100Iter);
%if &Char1000Iter=0 or &Char1000Max=0 %then %do;
      %let tot5=1;
      %let Char1000Iter=1;
      %let Char1000Max=0;
      %end;
%else %let tot5=%sysevalf((&Char1000Max+&Char1000Iter)/&Char1000Iter);
%if &Char32000Iter=0 or &Char32000Max=0 %then %do;
      %let tot6=1;
      %let Char32000Iter=1;
      %let Char32000Max=0;
      %end;
%else %let tot6=%sysevalf((&Char32000Max+&Char32000Iter)/&Char32000Iter);
%let tot=%sysevalf(&tot1*&tot2*&tot3*&tot4*&tot5*&tot6);
%let counter=1;
* create the data set to be analyzed, which varies by obs, field number, and type;
%do j=0 %to &DigFieldsMax %by &DigFieldsIter; * DO LOOP number of numeric fields;
      %do k=0 %to &Char10Max %by &Char10Iter; * DO LOOP number of 10 char fields;
            %do l=0 %to &Char100Max %by &Char100Iter; * number of 100 char fields;
                  %do m=0 %to &Char1000Max %by &Char1000Iter; *1000 char fields;
                        %do n=0 %to &Char32000Max %by &Char32000Iter; *32k char;
                              %do o=&DataRowIter %to &DataRowMax %by &DataRowIter;
%let beforeFile=%sysfunc(datetime(),datetime17.);
%if &o=&DataRowIter %then %do; *create a base table on the first iteration;
      data test (drop=j k l m n o p) testcum (drop=j k l m n o p); *initialize
                  testcum for later;
            length x y j k l m n o p 8;
            format x y 8.6;
            %if &j^=0 %then %do;
                  array num{&j} 8 num1-num&j;
                  %end;
            %if &k^=0 %then %do;
                  array char10_{&k} $10 char10_1-char10_&k;
                  %end;
```

```
		%if &l^=0 %then %do;
			array char100_{&l} $100 char100_1=char100_&l;
			%end;
		%if &m^=0 %then %do;
			array char1000_{&m} $1000 char1000_1-char1000_&m;
			%end;
		%if &n^=0 %then %do;
			array char32000_{&n} $32000 char32000_1-char32000_&n;
			%end;
		do o=1 to &o;
			call streaminit(123); * randomize Y-Y coordinates;
			x=(rand('uniform')*(&maxx-&minx))+&minx;
			y=(rand('uniform')*(&maxy-&miny))+&miny;
			%if &j^=0 %then %do;
				do j=1 to &j;
					num{j}=rand('uniform')*10;
					end;
				%end;
			%if &k^=0 %then %do;
				do k=1 to &k;
					char10_{k}="&char10";
					end;
				%end;
			%if &l^=0 %then %do;
				do l=1 to &l;
					char100_{l}="&char100";
					end;
				%end;
			%if &m^=0 %then %do;
				do m=1 to &m;
					char1000_{m}="&char1000";
					end;
				%end;
			%if &n^=0 %then %do;
				do n=1  to &n;
					char32000_{n}="&char32000";
					end;
				%end;
			output test;
			output testcum;
			end;
		run;
	%end;
%else %do; * incrementally build table if base table already exists;
	proc append base=testcum data=test;
	run;
	%end;
%let afterFile=%sysfunc(datetime(),datetime17.);
* compute file size;
proc sql;
	select filesize
	into :filesize
	from dictionary.tables
	where libname='WORK' and memname='TEST';
	quit;
run;
* GINSIDE the table that has been created;
%let beforeGINSIDE=%sysfunc(datetime(),datetime17.);
proc ginside data=test map=&map out=Gout;
	id &id;
run;
%let Gerr=&syserr;
%let afterGINSIDE=%sysfunc(datetime(),datetime17.);
```

```
* determine if GINSIDE produced the correct number of observations;
proc sql;
       select count(*)
       into :Gobs
       from work.Gout;
       quit;
run;
* compute runtime statistics;
data z (compress=yes);
       length filesize beforeFile afterFile beforeGINSIDE afterGINSIDE GINSIDEobs 8
              Map_Table GINSIDEerr $50;
       format beforeFile afterFile beforeGINSIDE afterGINSIDE datetime17.
              filesize 15.0;
       Data_Observations=&o;
       filesize=&filesize;
       Data_NumericFields=&j;
       Data_Char10Fields=&k;
       Data_Char100Fields=&l;
       Data_Char1000Fields=&m;
       Data_Char32000Fields=&n;
       beforeFile=”&beforeFile”dt;
       afterFile=”&afterFile”dt;
       secdiffFile=afterFile-beforeFile;
       beforeGINSIDE=”&beforeGINSIDE”dt;
       afterGINSIDE=”&afterGINSIDE”dt;
       secdiffGINSIDE=afterGINSIDE-beforeGINSIDE;
       Map_Table="&map";
       GINSIDEerr="&Gerr";
       GINSIDEobs="&Gobs";
run;
proc append base=statistics data=z force;
run;
                                          %put ITERATION: &counter of &tot;
                                          %let counter=%eval(&counter+1);
                                          %end;
                                   %end;
                            %end;
                     %end;
              %end;
       %end;
data statistics;
       set statistics;
       where ^missing(data_observations);
run;
%err: %put &err;
%mend;
```

APPENDIX B. WAFFLE_MAKIN MACRO (INCLUDES SPACETOCOMMA MACRO)

```
* SPACETOCOMMA accepts a space-delimited list and converts to a comma-delimited-
double-quotes-enclosed list;
* a character other than space can be utilized as defined by the SEP parameter;

options minoperator;
%macro spacetocomma (list= /* space-delimited or other-delimited list */,
       sep=SP /* the delimiter, defaulted to SPACE but can be any other character */);
%global spacetocommaout;
%local i;
%local list;
%local mod;
%let i=1;
%let mod=;
%let spacetocommaout=;
%if %length(&sep)=0 %then %let sep=SP;
%if "&sep" in "SP" "sp" "SPACE" "space" %then %do;
       %let sep=;
       %let mod=S;
       %end;
%do %while(%length(%scan(&list,&i,&sep,&mod))>1) and %eval(&i<10));
       %if %eval(&i=1) %then %let spacetocommaout="%scan(&list,&i,&sep,&mod)";
       %else %let spacetocommaout=&spacetocommaout,"%scan(&list,&i,&sep,&mod)";
       %let i=%eval(&i+1);
       %end;
&spacetocommaout
%mend;



* WAFFLE_MAKIN builds the matrix of rectangles that cover the region of the map data;
* it is called by WAFFLE_EATIN when a specific waffle matrix cannot be found;
* the map limits are first established, which is divided into four rectangles;
* any rectangle wholly inside the polygon of the map data set is attributed as such;
* other rectangles are subdivided into 4 more rectangles in successive iterations;

libname waffles '\\sas\libraries\gis\waffles\'; *set to applicable location;
options minoperator;

%macro WAFFLE_MAKIN(lib= /* library in which the SAS map data set is located */,
       map= /* SAS map data set for which waffles are being created */,
       id= /* space-delimited list of variables used to group map data set, which
              MUST BE IN IDENTICAL ORDER AS MAP DATA SET IS SORTED! */,
       maxiter= /* the maximum number of iterations (rectangle divisions) */);
%local lenlist;
%local dist;
%local i;
%local j;
* determine the field lengths that should be utilized and place in LENLIST variable;
proc sql;
       select (strip(name) || ' $' || strip (put(length,8.))) as lenlist
       into :lenlist separated by ' '
       from dictionary.columns
       where libname="%upcase(&lib)" and memname="%upcase(&map)" and upcase(name)
              in(%upcase(%spacetocomma(list=&id)));
       quit;
run;
* determine the min and max coordinates of the shapefile/polygon;
%let dist=.001; *distance in radians between each point that is generated;
data _null_;
       set &lib..&map (keep=x y) end=eof;
```

```
       length minx miny maxx maxy 8;
       if missing (minx) or x<minx then minx=x;
       if missing (maxx) or x>maxx then maxx=x;
       if missing (miny) or y<miny then miny=y;
       if missing (maxy) or y>maxy then maxy=y;
       retain minx maxx miny maxy;
       if eof then do;
              call symput ('minx',minx);
              call symput ('maxx',maxx);
              call symput ('miny',miny);
              call symput ('maxy',maxy);
              end;
run;
data rectangles;
       length x1 y1 x2 y2 8 &lenlist;
       format x1 y1 x2 y2 16.14;
run;
data miss;
       length x1 y1 x2 y2 8;
       format x1 y1 x2 y2 16.14;
       x1=&minx;
       y1=&miny;
       x2=&maxx;
       y2=&maxy;
run;
* divides each rectangle into four equally sized rectangles;
%let i=1;
%do %while(%sysevalf(&i<=&maxiter));
data test (keep=x1_ y1_ x2_ y2_);
       set miss;
       x1_=x1;
       y1_=y1;
       x2_=((x2-x1)/2)+x1;
       y2_=((y2-y1)/2)+y1;
       output;
       x1_=((x2-x1)/2)+x1;
       y1_=y1;
       x2_=x2;
       y2_=((y2-y1)/2)+y1;
       output;
       x1_=x1;
       y1_=((y2-y1)/2)+y1;
       x2_=((x2-x1)/2)+x1;
       y2_=y2;
       output;
       x1_=((x2-x1)/2)+x1;
       y1_=((y2-y1)/2)+y1;
       x2_=x2;
       y2_=y2;
       output;
run;
* form the dotted outline of each interior square;
data test2;
       set test;
       length x y n 8;
       do x=x1_ to x2_ by &dist; * make the horizontal sides of box;
              n=_n_;
              if x<x2_ then do;
                     y=y1_;
                     output;
                     y=y2_;
                     output;
                     end;
```

```
            end;
      x=x2_;
      y=y1_;
      output;
      y=y2_;
      output;
      do y=y1_ to y2_ by &dist; *make the vertical sides of box;
            if y<y2_ then do;
                  x=x1_;
                  output;
                  x=x2_;
                  output;
                  end;
            end;
      y=y2_;
      x=x1_;
      output;
      x=x2_;
      output;
run;
proc ginside data=test2 map=&lib..&map out=test_gin; *determine if rectangle in shape;
      id &id;
run;
proc sort data=test_gin out=sort_gin;
      by n &id;
run;
data sort_gin2 rectangle_plus (keep=x1_ y1_ x2_ y2_ &id rename=(x1_=x1 y1_=y1 x2_=x2
            y2_=y2)) miss (rename=(x1_=x1 y1_=y1 x2_=x2 y2_=y2)
            keep=x1_ y1_ x2_ y2_);
      set sort_gin end=last;
      by n &id;
      if first.n then bad=.;
      if ^first.n and (
            %let j=1;
            %do %while(%length(%scan(&id,&j))>1);
                  %let thisid=%scan(&id,&j);
                  %if &j=1 %then %do;
                        first.&thisid
                        %end;
                  %else %do;
                        or first.&thisid
                        %end;
                  %let j=%eval(&j+1);
                  %end;
                  ) then bad=1;
      if last.n then do; * if all points in polygon then add to rectangle_plus;
            if bad=1 then output miss;
            else if ^missing(%scan(&id,1)) then output rectangle_plus;
            end;
      output sort_gin2;
      retain bad;
run;
proc append base=rectangles data=rectangle_plus force;
run;
      %let i=%eval(&i+1);
      %end;
proc sql;
      select count(*)
      into :cnt
      from rectangles;
      quit;
run;
* produce waffle schema and print a map if rectangles were produced;
```

```
%if %eval(&cnt>1) %then %do;
	data waffles.&lib._&map._&maxiter;
		set rectangles;
		where ^missing(x1);
	run;
	data map (keep=x y segment &id);
		set rectangles;
		if ^missing(x1);
		segment=_n_;
		x=x1;
		y=y1;
		output;
		x=x2;
		y=y1;
		output;
		x=x2;
		y=y2;
		output;
		x=x1;
		y=y2;
		output;
		x=x1;
		y=y1;
		output;
	run;
	%if %eval(&cnt<10000) %then %do; * no HTML if too many rectangles exist;
		ods listing close;
		ods noproctitle;
		ods html path="%sysfunc(pathname(waffles))"
			file="&lib._&map._&maxiter..htm" style=sasweb;
		proc gmap map=map data=map all;
			id segment;
			choro &id;
			titlel "&map";
		run;
		quit;
		ods html close;
		ods listing;
		%end;
	%end;
%mend;
```

APPENDIX C. WAFFLE_EATIN MACRO

```
* WAFFLE_EATIN performs a more efficient GINSIDE, by first comparing known rectangles;
* if a waffle schema does not exist, one is created using the WAFFLE_MAKIN macro;
* macro variables CODE1-CODE[n] are created and parse each point mathematically;
* points falling inside known rectangles are attributed, others are GINSIDEd;
* after 9+ iterations, diminishing returns are experienced and efficiency gains lost;

%macro WAFFLE_EATIN (lib= /* library in which the SAS map data set is located */,
      map= /* SAS map data set for which waffles are being created */,
      tab= /* SAS data set that contains coordinate values X and Y */,
      outtab= /* SAS data set that is output, containing appended ID fields */,
      id= /* the ID variable (or space-delimited list) used to group map data set */,
      maxiter= /* the maximum number of iterations (rectangle divisions) */);
%global beforeWaffleReadFile;
%global afterWaffleReadFile;
%global beforeWaffleSubsetFile;
%global afterWaffleSubsetFile;
%global beforeWaffleGINSIDE;
%global afterWaffleGINSIDE;
%global beforeWaffleAppend;
%global afterWaffleAppend;
%global cntUnk;
%local lenlist;
%local i;
%local err;
* determine the field lengths that should be utilized and place in LENLIST macro;
proc sql;
      select (strip(name) || ' $' || strip(put(length,8.))) as lenlist
      into :lenlist separated by ' '
      from dictionary.columns
      where libname="%upcase(&lib)" and memname="%upcase(&map)" and upcase(name)
            in(%upcase(%spacetocomma(list=&id)));
      quit;
run;
* event handling;
%if %sysfunc(exist(waffles.&lib._&map._&maxiter)=0 %then %do; *try to create waffle
            schema if it does not exist;
      %waffle_makin (lib=&lib, map=&map, id=&id, maxiter=&maxiter);
      %end;
%if %sysfunc(exist(waffles.&lib._&map._&maxiter)=0 %then %do; *if the schema still
            does not exist, it could not be created because MAXITER value is too low;
      %let err=NO WAFFLE SCHEMA EXISTS OR COULD BE CREATED--TRY INCREASING MAXIMUM
            NUMBER OF ITERATIONS;
      %goto err;
      %end;
%let beforeWaffleReadFile=%sysfunc(datetime(),datetime17.);
data _null_;
      set waffles.&lib._&map._&maxiter end=eof;
      length code $32000;
      format x1 y1 x2 y2 16.14;
      if _n_=1 then do;
            iter=0;
            i=1;
            end;
      if i=1 then code='';
      code=strip(code) || ifc(i=1 and iter=0,"if ","else if ") || put(x1,16.14)
            || "<=x<=" || put(x2,16.14) || " and " || put(y1,16.14) || "<=y<="
            || put(y2,16.14) || " then do; " ||
      %let i=1;
      %do %while(%length(%scan(&id,&i))>1);
            %let var=%scan(&id,&i);
```

```
              "&var=""" || strip(&var) || """;" ||
              %let i=%eval(&i+1);
              %end;
                     "output &outtab;end;";
       if mod (_n_,150)=0 or eof then do;
              i=0;
              iter+1;
              call symput("code" || strip(put(iter,$8.)),strip(code));
              end;
       i+1;
       if eof then do;
              call symput("max", strip(put(iter,$8.))); * last code segment created;
              end;
       retain i iter code;
run;
%let afterWaffleReadFile=%sysfunc(datetime(),datetime17.);
%let beforeWaffleSubsetFile=%sysfunc(datetime(),datetime17.);
data &outtab; * data set must be reset;
       length &lenlist;
run;
data &outtab (drop=cntUnk) Unknown (drop=&id cntUnk);
       length &lenlist cntUnk 8;
       if _n_=1 then cntUnk=0;
       set &tab end=eof;
       %do i=1 %to &max;
              &&&code&i
              %end;
       else do;
              output Unknown;
              cntUnk+1;
              end;
       if eof then call symput('cntUnk',cntUnk);
       retain cntUnk;
run;
%let afterWaffleSubsetFile=%sysfunc(datetime(),datetime17.);
* now GINSIDE the remaining points that did not fall inside any waffles;
%let beforeWaffleGINSIDE=%sysfunc(datetime(),datetime17.);
proc ginside data=Unknown map=&lib..&map out=Unknown2(drop=density lat lon);
       id &id;
run;
%let afterWaffleGINSIDE=%sysfunc(datetime(),datetime17.);
%let beforeWaffleAppend=%sysfunc(datetime(),datetime17.);
* join the results;
proc append base=&outtab data=Unknown2;
run;
%let afterWaffleAppend=%sysfunc(datetime(),datetime17.);
%err: %put &err;
%mend;
```

REFERENCES

[1] SAS/Graph® 9.3: Reference, Third Edition: GINSIDE Procedure. Retrieved from http://support.sas.com/documentation/cdl/en/graphref/65389/HTML/default/viewer.htm#p0ian43grdtjosn1midf5xidk1ho.htm

[2] Army upgrades blue force tracking in Afghanistan to prepare for new network, February 13, 2013. Retrieved from http://www.army.mil/article/96438/

[3] SAS/GRAPH® 9.3: Reference, Third Edition: GREDUCE Procedure. Retrieved from http://support.sas.com/documentation/cdl/en/graphref/65389/HTML/default/viewer.htm#n1hhxy4sve4o36n1g1tcatc5nnk5.htm

ACKNOWLEDGMENTS

Special thanks to the employees of CACI-Wexford and JIEDDO for their assistance, encouragement, and perseverance toward a noble mission.

CONTACT INFORMATION

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

Name: Troy Martin Hughes
E-mail: troymartinhughes@gmail.com