Paper 009-2009

An Algorithm for Verifying Contiguity in a Set of Linked Entities

David A. Vandenbroucke[1]

ABSTRACT

Many kinds of data can be represented as a field of nodes connected by links. It is sometimes useful to know whether a set of nodes is contiguous, in the sense that all nodes can be traversed without leaving the set. This paper presents a SAS® macro program that can be used to verify whether this is so. It begins with a data set containing the full field of nodes and links. This is then reduced to a data set containing only the set of interest and the links internal to the set. It then attempts to trace links from an arbitrarily chosen reference node to every other node in the set. If this can be done, the set is contiguous. This macro uses only Base SAS® DATA step and macro programming. The specific example is verifying that a group of census tracts is contiguous. The paper and the code use terminology appropriate to that example.

I. INTRODUCTION

Data are often interconnected. In many cases, these interconnections can be represented by key fields or combinations of fields, in the standard relational database model. However, sometimes something more is needed. Entities of the same type, such as servers, persons, firms, or cities, can have an arbitrary number of connections to others of the same type. These connections might be called channels, friends, partners, or borders. Indirect connections exist when one entity can trace a series of links from itself to another via more than one intervening entity. Given a set of entities embedded in a larger field, the set can be said to be contiguous if every entity has a direct or indirect link via members of the set to every other entity. Contiguity can often be verified by inspection: Any small child can look at a map and tell whether all the parts are connected. However, an automated procedure may need to run without the intervention of small children. This paper describes an algorithm, in the form of a SAS macro procedure, which can verify that a given set of entities is contiguous.

The need for the procedure arose in the course of defining metropolitan area zones for the American Housing Survey.[2] The American Housing Survey (AHS) is a periodic survey of the housing stock in the United States, funded by the U.S. Department of Housing and Urban Development (HUD) and implemented by the U.S. Census Bureau.[3] The metropolitan portion of the AHS program, called AHS-MS, conducts surveys of the housing stock in selected metropolitan areas on a rotating basis (there is also a national component, which does not concern us for the purposes of this paper). To protect the confidentiality of the survey respondents, the Census Bureau limits the geographic detail that may be revealed in the survey's public use data sets. In response to these limitations, HUD developed a set of custom geographic areas, called zones, which are groups of census tracts that conform to the Census Bureau's restrictions. Thus, on the public use data set, a housing unit is identified as being in particular zone. In general, the zone is the smallest geographic entity which the AHS data associates with a sampled housing unit.

Zones have a few required, and many desired, properties. Not all of the latter are consistent with one another, and most are not important for the purposes of this paper. However, one desirable property is contiguity. It is handy if the census tracts comprising the zones form recognizable, mappable, subregions of the metropolitan areas. Thus, the need arose to verify contiguity while refining zones in an automated program.

II. THEORY OF OPERATION

This algorithm relies on a data set developed by HUD GIS experts which indicates the census tracts adjacent to every census tract in the country. Each record of this data set represents one census tract, identified by its FIPS code[4]. The record's fields include this code in a variable, called FIPS, plus variables containing the FIPS codes of all the census tracts to which it is adjacent. These variables are named GEOID1-GEOID*n*, where *n* is the maximum number of tracts to which any one tract is adjacent. For the national data set, *n* is 30, but for the working subsets discussed below, the maximum is usually much smaller, in the high single digits. Cases that have fewer than the maximum number of links have the remaining GEOID*x* variables set to missing values.

---

[1] Senior Economist, Office of Policy Devlopment & Research, U.S. Department of Housing & Urban Development. The views expressed in this document are the author's and do not necessarily reflect those of the United States government.

[2] See Vandenbroucke, David A. 2008.

[3] For more information about the AHS, see U.S. Census Bureau. 2004.

[4] FIPS is an acronym for Federal Information Processing Standards. It is a unique code associated with a geographic entity. See http://quickfacts.census.gov/qfd/meta/long_fips.htm for details.

The calling program works with a small portion of a metropolitan area (typically, a county). It begins with a preliminary set of zones that are contiguous and distinct from one another in terms of demographic and housing characteristics. However, these preliminary zones do not meet the Census Bureau's requirement that each have a population of at least 100,000 persons. The program proceeds by selecting census tracts on the periphery of larger zones and reassigning them to neighboring, smaller zones. At each step, it is possible that the transfer will render the donor zone noncontiguous. This macro is designed to detect such situations, so that an alternative tract can be selected. The main steps in the algorithm are:

1. Trim the input data set to discard all records that are not members of the donor zone and all links to tracts outside the donor zone. Thus, what remains are just the tracts in the zone and internal links.

2. Choose one tract as the reference point to begin tracing links to all other tracts. Since there are no special requirements for this tract, the first record in the data set is used. Make a list of all the links from the reference tract to the other zone members.

3. Make a pass through the data set looking for the linked-to tracts. Make a new list of links, based on the links from those tracts. Delete those tracts from the data set, since they have now been traced to the reference tract.

4. Check to see if all the tracts in the zone have been accounted for. If so, then report success. If not, check if at least one new tract was linked in step 3. If so, then loop back to step 3. If not, then report failure.

III. DETAILS

For reference, here are lists of the data sets, macro variables, and macro procedures used:

```
%MACRO Contiguity;
     /*

          Data sets
          ---------
          Contiguity          Table of tracts in OldZone [FIPS] and the links [GeoID]
                              from them to other tracts in OldZone (1 record per
                              FIPS).
          Region.Tracts       Tract data (1 record per tract)
          Links               Working list of links from recently connected tracts.
          TractList           List of tracts and links from them (1 record per link)

          Macro Variables
          ---------------
          EndObs         Number of tracts yet to be connected at the end of the main
                         loop (set by this macro).
          LinkFound      Flag indicating that an internal link to a given tract was
                         found (set by this macro).
          MaxAdj         Maximum number of possible adjacencies (set exogenously).
          OldZone        ID Number of the zone losing the tract (set outside this
                         macro).
          Result         Code indicating success, failure, or continue (global
                         variable,set by this macro).
          StartObs       Number of tracts yet to connected at start of main loop (set
                         by this macro).
          TargetTract    ID of the tract being reallocated (set outside this macro).

          Macro Routines
          --------------
          NObs           Function-type macro that returns the number of records in the
                         data set.
     */
```

The first step is to extract the tracts in the zone of interest, and then trim their lists of links to include only internal ones. The input data set Region.Tracts contains data on the entire area whose zones are being defined. Data set TractList is created from Region.Tracts. It includes all of the tracts in the zone (&OldZone), excluding the one that is the candidate for transfer (&TargetTract). Each record of TractList is one FIPS (tract) and one GEOID (link), representing a link to an adjacent tract. Each combination of FIPS+GEOID is unique.

```
/* Output list of GEOIDs [links] for each FIPS [tract] */
DATA TractList (KEEP = FIPS GeoID);
     SET Region.Tracts (
          WHERE = (Zone = &OldZone AND FIPS ^= "&TargetTract" )
          )
     ;
     ARRAY link[&MaxAdj] GeoID1-GeoID&MaxAdj;

     /* output list of links to current tract, skipping the target tract.
          Note that each record in Region.Tracts generates up to MaxAdj records
          in TractList.
     */
     DO _I_ = 1 TO &MaxAdj;
          IF NOT MISSING(link[_I_]) AND
               link[_I_] ^= "&TargetTract"
               THEN DO;
               GEOID = link[_I_];
               OUTPUT;
          END;
     END;
RUN;
```

We sort TractList in GEOID order. Now the data set is ordered by links, rather than tracts.

```
/* put tract list in order of geoid */
PROC SORT DATA=TractList;
     BY GeoID;
RUN;
```

At this point, we have a list of the links from the zone to anywhere in the region. We want to restrict this to links within the zone. We merge TractList with Region.Tracts, matching GEOID in TractList with FIPS in Region.Tracts. This allows us to identify the Zone of the links. The new version of TractList includes this information as the variable LinkedZone.

```
/*Check GeoIDs against original tract database and record the zone of the link*/
DATA TractList (KEEP = FIPS GeoID LinkedZone);
     MERGE
          TractList (in = inTract)
          Region.Tracts (
               KEEP = FIPS Zone
               RENAME = (FIPS = GEOID Zone = LinkedZone)
               )
     ;    /* end of merge */;
     BY GEOID;
     IF inTract;
RUN;
```

Now we sort TractList back into FIPS order, so that it is sorted by tracts rather than links.

```
/* sort back into FIPS order */
PROC SORT DATA=TractList;
     BY FIPS;
RUN;
```

This is a convenient point at which to initialize the Result macro variable, which we will use to communicate with the calling program.

```
%LET Result = 0;    /* initialze to mean that the work isn't finished yet
                         -1 = contiguity check fails
                          0 = no result yet; continue checking
                          1 = contiguity check succeeds
                    */
```

The next DATA step has two functions. First, it filters from TractList all external links, keeping only those within the target zone. Second, it checks to see if any tract (FIPS) has *no* links to other members of the zone. If that is the case, then the tract is isolated, and the zone cannot be contiguous. When this situation is identified, the Result macro variable is set to -1, indicating a failure.

In the application for which this macro was written, a single tract can never have a large enough population to be an acceptable zone on its own. Thus, discovering an isolated tract necessarily means that the zone is not contiguous. In other applications, it may be that a single entity *is* an acceptable zone. If that is so, then this step would need additional logic to distinguish between a single-tract zone (acceptable) and a multi-tract zone that has an isolated piece (not acceptable).

```
DATA TractList (KEEP = FIPS GEOID);
     SET TractList;
     BY FIPS;  /* We need this to get the first and last attributes */
     RETAIN LinkFound 0; /* flag used to check for isolated tracts */
     IF LinkedZone = &OldZone THEN DO;  /* we found an internal link */
          OUTPUT;   /* save the record */
          LinkFound = 1; /* Note that we found a link */
     END;

      /* check for isolated tract, reset flag */
      /* Notice that this section changes Result only if it finds an isolated
         tract.  Thus, Result stays equal to 0 unless at least one tract is
         isolated.
      */
     IF last.FIPS THEN DO;
          IF NOT LinkFound THEN
               CALL SYMPUT('Result','-1');   /* if no link was found, set the
                                                result to failure */
          LinkFound = 0;                     /* In any case, initialize the flag
                                                for the next FIPS group */
     END;
RUN;
```

If Result = -1 at this point, the rest of the macro is skipped, since we have already established that the zone is not contiguous. Otherwise, we reassemble the data as a list of tracts with all of its links. Data set Contiguity has one record per tract (FIPS), each containing only internal links. In addition, this DATA step creates data set Links, which contains the links from the reference tract to the other zone members. The reference tract is simply a starting point from which to trace connections to all other tracts. For convenience, the first record in TractList is chosen as the reference tract. Note that this tract is *not* included in Contiguity, since we don't need to trace a connection *to* it.

```
%IF &RESULT = 0 %THEN %DO;
     DATA Contiguity (KEEP = FIPS GeoID1-GeoID&MaxAdj)
          Links (KEEP = GeoID)
          ;
          SET TractList;
          BY FIPS;
          LENGTH RefTract GeoID1-GeoID&MaxAdj $ 11;
          RETAIN RefTract GeoID1-GeoID11;
          RETAIN _I_ 0; /*note that _I_ is initialized to zero */

          /* The first tract in the database is defined as the reference tract.
               We trace links from it to all other tracts in the zone.
               For the reference tract, output the links to other tracts to the
               Links data set.  Note that the reference tract is not included in
               Contiguity, because it has already been processed.
          */
          IF _N_ = 1 THEN DO;
               RefTract = FIPS;
          END;

          IF FIPS = RefTract THEN DO;
               OUTPUT Links;
          END;
```

```
                /* For tracts other than the reference tract, add one record to the
                     Contiguity data set, consisting of the tract code and all of the
                     links from that tract.
                */
                ELSE DO;
                     _I_ + 1;  /* increment the geoid counter*/

                     /* add one GeoID link to the list for this tract */
                     ARRAY link[&MaxAdj] GEOID1-GEOID&MaxAdj;
                     link[_I_] = GEOID;

                     /* when all of the links for this tract have been processed,
                        output One record
                     */
                     IF last.FIPS THEN DO;
                          OUTPUT Contiguity;

                          DO _J_ = 1 TO _I_;  /* reset GeoIDs that have been used */
                               link[_J_] = ' ';
                          END;

                          _I_ = 0;  /* reset counter */

                     END;
                END;
           RUN; /* data contiguity and links*/
      %END;     /* of if result = 0 ... */
```

At this point, the tracts and internal links have been isolated from the rest of the region. In addition, we have a list of links leading from the reference tract to its adjacent tracts. The task now is to see if we can trace a path of links from the reference tract to all other tracts in the zone. This is an iterative process that takes place in a DO WHILE loop, which keeps running as long as Result = 0. At the top of the loop, we record the number of records in contiguity, using a function type macro.[5] This will be used at the bottom of the loop to determine if we have made any progress. Then we sort the Links data set, using NODUPKEY, to keep only one instance of each link (this doesn't matter for the first pass of the loop, but it may in subsequent passes).

```
     /* main loop */
     %DO %WHILE ("&Result" = "0");
          %LET StartObs = %NOBS(Contiguity);     /* record record count at top of loop
*/

          PROC SORT DATA = Links NODUPKEY;
               BY GeoID;
          RUN;
```

The next DATA step does the main work of tracing the links. It merges the Links and Contiguity data sets, matching the link (GEOID) with the tract (FIPS). At the same time, it creates a new Links data set, which will be used to trace the connections further in the next loop.

```
          DATA
               Contiguity (KEEP = FIPS GeoID1-GeoID&MaxAdj)
               LINKS (KEEP = GeoID)
          ;    /* end of data statement*/

               MERGE
                    Links ( IN = inLinks
                              RENAME = (GeoID = FIPS)
                          )
                    Contiguity (IN = inContiguity)
               ;    /* end of merge*/
               BY FIPS;
```

[5] This macro was taken from Cody, Ron. 2004, p. 346.

If a match is found, this is a tract that can be connected to the reference tract. We write all of the links *from* this tract to the Links data set. The record of the tract itself is *not* written to the new Contiguity data set, because we no longer need to trace a connection to it.

```
            IF inLinks AND inContiguity THEN DO; /* linked tract has not been
                                                    processed*/
                 ARRAY Link[&MaxAdj] GeoID1-GeoID&MaxAdj;
                 DO _i_ = 1 TO &MaxAdj;
                      GeoID = Link[_i_];
                      IF GeoID ^= ' ' THEN OUTPUT Links;
                 END;

            END;
```

If no link is matched to a tract, then this tract has not yet been connected to the reference tract. We simply write it to the Contiguity data set, so that it can be checked again in the next loop.

If no tract is matched to a link, then it must be that the tract the link points to was already connected in a previous loop. That's fine. We do nothing. The link is *not* written to the new Links data set.

```
            IF inContiguity AND NOT inLinks THEN
                 OUTPUT Contiguity; /* tract has not yet been linked to. We keep
                                       it. */
       RUN; /* data step*/
```

At this point we record the number of tracts in the new Contiguity data set (EndObs) and compare it to the number recorded at the beginning of the the loop (StartObs). The new count tells us how many tracts are still not connected to the reference tract. There are three possible outcomes:

1. There are no unconnected tracts (EndObs = 0). This means that all of the tracts have been connected, and so we have confirmed that the zone in contiguous. Set Result = 1 (success).

2. There are still some unconnected tracts, but fewer than at the beginning of the loop (0 < EndObs < StartObs). This means that the loop continues to make progress connecting the tracts, but the work is not over. Result =0, and the DO WHILE loop will run again.

3. There are still unconnected tracts, and the count at the end of the loop is the same as at the beginning (0 < EndObs = StartObs). This means that the loop failed to find any additional links from the reference tracts to unconnected tracts. Thus, there are some tracts that cannot be connected. Set Result = -1, meaning failure.

```
          %LET EndObs = %NOBS(Contiguity); /* record record count at bottom of loop */

          /* check for completion */
          %IF &EndOBS = 0 %THEN
               %LET Result = 1;    /* all tracts linked--success */
          %ELSE %IF %EVAL(&EndObs < &StartObs) %THEN
               %LET Result = 0;    /* progress, but not finished--continue */
          %ELSE
               %LET Result = -1;   /* no progress, not finished--failure */

     %END;     /* of do while result... */

%MEND;    /* Macro contiguity*/
```

IV. USING THE MACRO IN A CALLING PROGRAM

Before calling the macro, the NObs macro must already have been compiled. The MaxAdj macro variable must have been set to the maximum number of links allowed from any one tract. OldZone must hold the identifier of the zone being verified. TargetTract must contain the identifier for tract which is the candidate for reassignment. The macro is then invoked without parameters.

When the macro has completed its operation, the result is returned in the Result global macro variable. The calling program must evaluate that variable and then decide what to do, depending on whether the result is success or failure. In the original application, when Result = 1, the program proceeds to transfer the target tract to its new zone. When Result = -1, the program marks the tract as unsuitable for transfer and attempts to find another one to transfer instead.

V. CONCLUSION

Although the code of this macro is written in terms of tracts, zones, and GeoIDs, it can accommodate any data in which entities are related by links to other entities in the data set. One could, of course, do a search-and-replace to change variable names in order to match those in the calling program. The advantage of the approach used in this macro is that it uses only DATA step and macro logic available in Base SAS. All data are organized as data sets and processed in the record-by-record fashion familiar to SAS programmers.

REFERENCES

Cody, Ron. 2004. *SAS Functions by Example.* Cary, NC: SAS Institute, Inc.

U.S. Census Bureau. 2004. *Housing Data Between the Censuses: The American Housing Survey*. Census Special Reports, AHS/R/04−1, Washington, DC. http://www.census.gov/prod/2004pubs/ahsr04-1.pdf

Vandenbroucke, David A. 2008. "The Improved AHS Sausage Machine, or American Housing Survey Metropolitan Zones." Paper delivered to the Southern Regional Science Association, March 28. [Electronic copy available from author.]

CONTACT INFORMATION

David A. Vandenbroucke
U.S. Department of Housing and Urban Development
451 7th Street SW, Room 8218
Washington, DC 20410
Phone: 202-402-5890
Fax: 202-708-3316
E-Mail: david.a.vandenbroucke@hud.gov