

Who's Your Neighbor? A SAS® Algorithm for Finding Nearby Zip Codes

Andrew Clapson, MD Financial Management; Annmarie Smith, HomeServe USA

ABSTRACT

Even if you're not a GIS mapping pro, it pays to have some geographic problem-solving techniques in your back pocket. In this paper we'll illustrate a general approach to finding the closest location to any given US zip code, with a specific, user accessible example of how to do it using only Base SAS. We also suggest a method for implementing the solution in a production environment, as well as demonstrate how parallel processing can be used to cut down on computing time if there are hardware constraints.

INTRODUCTION

Let's say that you are in charge of marketing data services for your company. There is a new marketing campaign in the works, and part of that campaign involves some cool graphics that show a map that is colored and highlighted based on the geographic area of the recipient. This new campaign has given you the following request: given a list of address information for clients and prospects, supply the three closest zip codes for each mailing address. Because your company operates nationwide, all 50 of the United States must be included, and the list of clients will be updated on a regular basis for new 'waves' of the campaign.

The problem sounds fairly simple (and frankly, it is): given any zip code in the United States, return the three closest adjacent zip codes, ordered by distance. When looking at a map divided by zip code, it's easy enough to visually pick out the closest adjacent points, but what if your range of possible inputs is **any and all** US zip codes? As well, what if you also need to be able to do this repeatedly, programmatically, and you only have access to Base SAS?

One solution to the problem (called a 'linear search algorithm') would look like this:

1. Create a data set containing all zip codes in the United States
2. Cross-join this data set with every OTHER zip code in the United States (i.e. build a Cartesian product)
3. Calculate the distance between every possible pair of zip codes in the US
4. Sort the results in descending order
5. Keep the top three results for each zip code
6. Repeat as needed

This approach will logically 'work' and will find a solution, but let's see how this works in practice.

GETTING TO THE SOLUTION

TRACKING DOWN THE DATA

First, we need a list of every single zip code in the United States. This turns out to be easy - SAS has this data set built-in! The SASHELP library contains a data set named 'zipcode,' which contains the full list of the approximately 41,000 zip codes in the United States, including the District of Columbia, Puerto Rico and outlying territories. The following code creates a subset of the 'SASHELP.zipcode' data set (only selecting zip codes that begin with the digits '05') to use as our example working file:

```
proc SQL;
  CREATE TABLE ZipCode_1 AS
  SELECT left(put(A.Zip, Z5.)) AS Zip_A LENGTH = 5,
  left(put(B.Zip, Z5.)) AS Zip_B LENGTH = 5
  FROM SASHELP.Zipcode A, SASHELP.Zipcode B
  WHERE (substr(left(put(A.Zip, Z5.)), 1, 2) = '05')
  AND (substr(left(put(B.Zip, Z5.)), 1, 2) = '05')
  AND (A.Zip NE B.Zip);
QUIT;
```

Note that the final part of the WHERE clause in the above SQL procedure, '(A.Zip NE B.Zip)', excludes the 41,000 cases where zip codes are matched with themselves.

What we've done here is calculate a cross-join or Cartesian product: each row is mapped to every other row. In this case, the two source data sets are the same 'zipcode' data set in the SASHELP library. This query cannot be optimized and SAS will tell you so. Cross-joins tend to be very large because they are quadratic; the size of the resulting data set will be the square of the number of rows in the input data set(s).

THE ZIPCITYDISTANCE() FUNCTION

Now that we have created a data set containing every '05' zip code lined up with every other '05' zip code, how can we find the distance between them? SAS to the rescue again with the ZIPCITYDISTANCE() function, which takes two zip codes as inputs and gives us the distance (in miles) between them. Perfect! Let's run that, too:

```
data WORK.ZipCode_2;
set WORK.ZipCode_1;
  Distance = zipcitydistance(Zip_A, Zip_B);
run;
```

Figure 1 shows a sample of the resulting data set:

	Zip_A	Zip_B	Distance
1	05001	05047	1.1
2	05001	05009	1.9
3	05001	05088	2.2
4	05001	05059	3.4
5	05001	05084	4.1
6	05001	05055	4.5
7	05001	05052	4.8
8	05001	05048	5.2

Figure 1 - Sample of data set 'WORK.ZipCode_2'

All that's left now is to sort the data set in descending order by zip code and distance, take our top three closest zip codes for each, and then we're done! Right...?

```
proc sort data = WORK.ZipCode_2;
  by Zip_A Distance Zip_B;
run;

data WORK.ZipCode_3;
set WORK.ZipCode_2;
  by Zip_A Distance;
  if (first.Zip_A = 1) then do;
    numZipCodes = 1;
  end;
  else do;
    numZipCodes + 1;
  end;
  if numZipCodes <= 3 then output;
run;
```

Here are the first few rows of the resulting data set, shown in Figure 2:

	Zip_A	Zip_B	Distance	numZipCodes
1	05001	05047	1.1	1
2	05001	05009	1.9	2
3	05001	05088	2.2	3
4	05009	05047	0.9	1
5	05009	05088	1.6	2
6	05009	05001	1.9	3
7	05030	05089	2	1

Figure 2 - Sample of data set 'WORK.ZipCode_3'

Well...no, we're not quite done. In order for any of this work to be useful, we need to implement this solution in a way that allows us to use it in production, likely along with other SAS code we may run on a regular (daily/weekly/monthly) basis.

THE CANADIAN EXCEPTION: POSTAL CODES:

As one of the authors of this paper is Canadian, we'll make a brief comment on the Canadian flavour of this problem: because Canadian postal codes are not automatically included in SAS, we would first have to find a 'mapping' data set containing a list of all Canadian postal codes along with their latitude and longitude midpoints. From there we would use the same approach as shown above, instead using the GEODIST() function and the latitude and longitude coordinates of each postal code to compute the distances between each.

OPERATIONALIZING THE SOLUTION

Let's not lose sight of our goal here! Taking any valid US zip code as an input, we were tasked to find the closest neighboring three zip codes. Having run all the previous code, we've crunched the numbers and built our distance reference table, and we now need to perform the lookups.

Our first challenge is the data layout. Currently our data is normalized into a few long columns, and while this is optimal for SAS in terms of storage and performance, it can be inconvenient for creating reports or carrying out certain calculations. For this example, we simply want to identify the three closest zip codes for each mailing address in the marketing campaign, so let's use the TRANSPOSE procedure to create three new columns in our data set (and in the process turn 'Zip_A' into our primary key):

```
proc transpose data = WORK.ZipCode_3
              out = WORK.ZipCode_4 (drop = _NAME_)
              prefix = ClosestZip_;
  by Zip_A;
  ID numZipCodes;
  var Zip_B;
run;
```

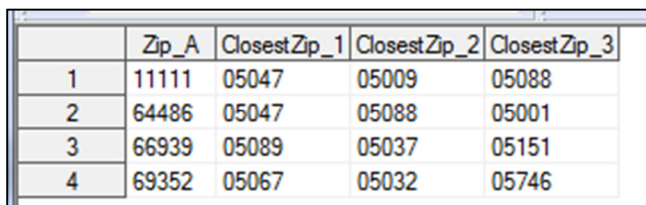
If this zip code-matching process is to become part of our regular production, we would here store the transposed reference table of zip codes and distances in a permanent library and use it as a lookup table for future problems. (It would require occasional updates as zip codes are modified, however.)

Now for the fun part! We have the zip code distances calculated and ranked, so where is our input? For the purposes of this paper, let's assume that the target campaign zip codes are submitted as a SAS data set – we'll create a handful of them here as an example:

```
data WORK.ZipsToMatch_1;
input Zip_A $5.;
datalines;
64486
66939
69352
11111
;
run;

data WORK.MatchedCodes;
merge WORK.ZipCode_4 (in = a) WORK.ZipsToMatch_1 (in = b);
  if a & b then output;
run;
```

As we can see in Figure 3, the final data set shows each zip code, along with the three closest neighboring zip codes (because we used only '05' zip codes for our example so far, these matches are far from optimal):



	Zip_A	ClosestZip_1	ClosestZip_2	ClosestZip_3
1	11111	05047	05009	05088
2	64486	05047	05088	05001
3	66939	05089	05037	05151
4	69352	05067	05032	05746

Figure 3 - Sample of data set 'WORK.MatchedCodes'

Now that we've defined a general solution to the problem, it same framework be applied to a variety of similar tasks. For instance, by referring to the original 'SASHELP.Zipcode' data set in a slightly different way, you can find the closest city, county, etc. The zip code is your key to finding this information!

SURE, IT WORKS...BUT DOES IT SCALE?

Total run time for the above code - including creating the example data itself - is less than a minute on a standard desktop SAS installation. The above example uses an artificially constrained data set, however; try taking all zip codes that start with '5' (as opposed to '05') and you'll find your run time increases to over 10 minutes. This isn't an unreasonable amount of time, but what if the possible inputs were the full set of over 41,000 zip codes in the United States? Recalling the quadratic relationship for a Cartesian product, $41,000^2$ yields over 1.68 billion observations! Running the above code on a data set of that size on a standard desktop machine can require over 24 hours of processing time.

It so happens that the bottleneck in our code is the ZIPCITYDISTANCE() function, which unfortunately we can't do without, so we'll need to work around this to find a way to cut down on processing time.

OPTIMIZING THE COMPUTATIONS

The main problem is simply the quantity of data that we have to deal with, as there are almost 1.7 billion (1,680,100,000) observations in the full data set. The first question we should ask is: can we reduce the inputs to the problem at all? If we only needed a subset of all zip codes (say, only those for the state of California), we could certainly leave out all the others, however as stated in the introduction, our general solution has to apply for all zip codes in the United States.

Depending on the precise nature of the bottleneck, we may be able to reduce our run time by using parallel processing or multi-threading. Using the concepts and code from Ian Ghent's excellent 2010 paper on this subject, let's submit this code for parallel processing and see what our time savings can be! For this particular example, I'm running SAS 9.4 on an HP Laptop with 8 Gb of RAM and a 4-core processor, so your mileage may vary! Table 1 shows the time taken for each run:

	Non-Threaded	Threaded	
Number of threads	1	2	4
Total Processing Time	> 24 hours	12 hours	9 hours

Table 1 - Runtimes by thread count

Note: The reader can find details on implementing SAS multi-threading either in the cited paper (Ghent 2010) or in the included sample code for this paper.

Quite the improvement! We have more than halved our processing time using multi-threading...although admittedly the total run time still exceeds most SAS programmer's workdays. Luckily, this distance table computation needs only to be run once, with perhaps occasional updates as zip codes are updated or changed. Again, if this zip code calculation was something we planned to use regularly, we would permanently store the resulting table of zip codes and distance and use it as a lookup table for future queries.

Although a server-based deployment of SAS would likely make short work of a problem like this, this same sort of code can be used to scale other tasks that may benefit from multi-threading.

USING HASH TABLES FOR LOOKUPS

For additional efficiency gains we can also replace the final DATA step MERGE with a hash table lookup. A hash table loads the entire set of keys into memory and does fast lookups against keys, versus sequentially reading a data set:

```
data WORK.MatchedCodes;
  if 0 then set WORK.ZipsToMatch_1 WORK.ZipCode_4;

  set WORK.ZipsToMatch_1;

  if _n_ = 1 then do;
    declare Hash findZipCode(data set:'WORK.ZipCode_4');
    findZipCode.defineKey('Zip_A');
    findZipCode.defineData('ClosestZip_1', 'ClosestZip_2',
      'ClosestZip_3');
    findZipCode.defineDone();
  end;

  if findZipCode.find(key:Zip_A) = 0 then output;
run;
```

In brief, the code in the above DATA step does the following:

- Uses the 'IF 0' conditional statement to copy the data set structure from the two data sets
- Declares (sets up) a hash table named 'findZipCode'
- Defines 'Zip_A' to be the key value and inserts the three closest zip code variables as the value associated with each key (i.e. the closest three zip code to each key value)
- Finally, uses the FIND() method to match the key value to the 'Zip_A' code in our 'ZipsToMatch_1' data set and output only those observations with matching key value pairs.

Whether the above hash table approach will yield material time-savings depends on both the amount of data being retrieved from the master lookup table (the size of the 'ZipsToMatch_1' dataset) and the frequency of the query.

CONCLUSION

Even without SAS/GIS software, Base SAS is a powerful tool and can help with virtually any task. The concepts and code in this paper demonstrate a basic but effective way to do geographic computations, and show some use of parallel processing. Like with most problems, it's worthwhile to not only think through a solution but also spend some time on implementation and optimization. In our case, the general process is well-defined, but the specific solution will vary on a case-by-case basis. Inefficient code is perfectly acceptable in many cases, but when data volume begins to grow, processing time becomes more and more troublesome. Being able not only to code efficiently, but also being able to think through the problem AND the application of potential solutions are some of the best tools any SAS programmer can have in their back pocket.

REFERENCES

Ghent, Ian J. April 2010. "Faster results by multi-threading data steps" *SAS Global Forum 2010 Conference*, Cary, NC: SAS Institute.

Available at: <http://support.sas.com/resources/papers/proceedings10/109-2010.pdf>

CONTACT INFORMATION

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

Andrew Clapson

MD Financial Management

andrew.clapson@cma.ca

Annmarie Smith

Director, Marketing Data Services

HomeServe USA

203.356.4217

annmarie.smith@homeserveusa.com

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.