Paper 091-2012

Batch Production of Driving Distances and Times Using SAS® and Web Map APIs

Ash Roy and Yingbo Na, Canadian Institute for Health Information, Toronto, Canada

ABSTRACT

This is a new methodology of using SAS® URL access method and Web APIs to run queries on an interactive Web site. This method will capture driving distances and times from a Web map based on points marked by postal codes.

INTRODUCTION

Distance analysis has become a growing need in health and consumer businesses in order to determine how far patients or customers are from a hospital or service centre. SAS® 9.2 has added some new tools for distance analysis and map functionality which make distance analyses and map visualizations easier. The new tools are based on the spatial relationships between the coordinates of latitudes and longitudes. These calculations give us straight line distances (i.e. “as the crow flies” distances). With complex road systems, we want to know the fastest driving time or the shortest driving route to reach a nearby hospital or service centre.

In a healthcare delivery system, especially in cases of serious injuries, the time taken to arrive at an emergency department has a significant impact on the treatment plan and its outcome. Therefore in planning health facilities, it is advantageous to take into account driving distances and travel durations for patients to reach hospitals or health services.

In this paper, we will describe the SAS® URL access method to parse an XML (Extensible Markup Language) file returned from an interactive website. This method is applicable to APIs for MapQuest, Google Maps, Yahoo Maps or Bing Maps. For the purpose of this paper, we will explore the possibilities of using SAS and MapQuest APIs. Please refer to the MapQuest website for complete API documentations.

Nowadays, before setting off for an unfamiliar place, we usually consult web applications e.g. Google Map or MapQuest for travel information. In cars, we use satellite navigation (Global Positioning System) to direct us to our destinations. Either in web map or in GPS, postal codes or full street addresses are widely used to determine distances between two or more points. Some tools offer the option of using latitude and longitude coordinates instead of postal codes or address as there are many places with no postal codes e.g. non-residential areas, highways etc.

SPATIAL DISTANCE

Let us take a quick look at the present method of straight line distance analysis. So far, we are using coordinates of latitudes and longitudes to find out the distances between two or more points. There are ways to convert postal code to latitude-longitude coordinate or vice versa. For example, we can use Statistics Canada’s Postal Code Conversion File (PCCF) to perform such conversion. We use the following macro to calculate distances either in miles (MI) or in kilometers (KM). This is known as the Great Circle Distance Formula.

```
%MACRO geodist (lat1,long1,lat2,long2, unit) ;
        %local ct ;
        %let ct = constant('pi')/180 ;
        %if %upcase(&unit) = KM %then %let radius = 6371 ;
        %else %if %upcase(&unit) = MI %then %let radius = 3959 ;
        &radius * ( 2 * arsin(min(1,sqrt( sin( ((&lat2 - &lat1)*&ct)/2 )**2 +
                cos(&lat1*&ct) * cos(&lat2*&ct) * sin( ((&long2 - &long1)*&ct)/2 )**2)
)));
%MEND;
```

For example, the following dataset (Figure 1) name PCS has a pair of postal codes with their coordinates.

	pc1	lat1	long1	pc2	lat2	long2
1	M4C5L8	43.695328	-79.291638	M2P2B7	43.747882	-79.405258

Figure 1. A Pair of Postal Codes with Coordinates of Latitudes and Longitudes

The above macro calculates the distance between these two postal codes using their geographical coordinates.

The codes are as follows:

```
DATA pcs_dist;
      set pcs;
      distance = %geodist (lat1,long1,lat2,long2, KM);
RUN;
```

And the output dataset contains the distance. The distance between M4C5L8 and M2P2B7 was found to be 10.840573649 km.

	pc1	lat1	long1	pc2	lat2	long2	distance
1	M4C5L8	43.695328	-79.291638	M2P2B7	43.747882	-79.405258	10.840573649

Figure 2. Output Dataset Containing Distance

However, to travel from M4C5L8 to M2P2B7, we cannot fly like a crow. To reach the destination from the starting point we have to take a feasible route e.g. by driving, walking, biking etc. This macro cannot give us the actual driving distance which is obviously different from the above straight-line distance.

WEB MAP

Let us look at the MapQuest web map to see the driving distance and time. One of the options from a suggested routes found the distance to be 17.76 km and 21 minutes driving time (this time varies and it is an estimate based on certain given factors but not all road conditions are considered). Please note that the web query takes a few seconds to figure out the route options and draws lines on the map. The SAS URL method can parse the distance and time

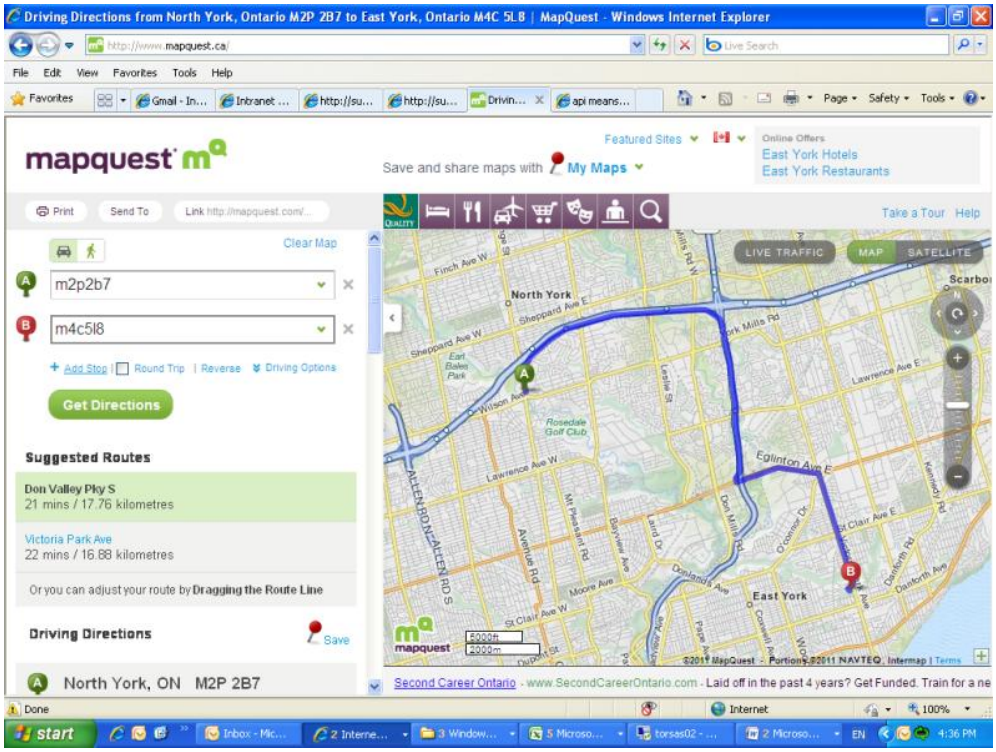

Figure 3. Travel Distance on a Traditional Web Map

from web maps as described by Mike Zdeb in SAS Global Forum 2010. Our test found that it took about a minute to parse one pair of distance and time in this method using SAS. In fact, this reading depends on the distance between two points. The longer the distance, the longer the time it takes to calculate. Moreover, same SAS® code may not work over a period of time due to changes in the underlying HTML codes.

APPLICATION PROGRAMMING INTERFACE (API)

An application programming interface (API) is a particular set of rules ('code') and specifications that software programs can follow to communicate with each other. It serves as an interface between different software programs and facilitates their interaction, similar to the way the user interface facilitates interaction between humans and computers.

MapQuest, Google Map, Yahoo Map and Bing Map have provided many powerful and functional APIs for programmers to develop various applications. For driving distance calculation purposes, we found all of them have similar kind of API functionalities. We have used MapQuest APIs to demonstrate how it works using its Community Key which is free of charge. They have other available options available.

Please refer to the MapQuest API documentation website for detailed descriptions of key words and codes.

UNDERSTANDING MAPQUEST APIS

With a valid API key (without the bracket), if we submit the following address in a browser: http://www.mapquestapi.com/directions/v1/route?key=(key)&outFormat=xml&unit=k&routeType=shortest&narrativeType=none&from=m4c5l8&to=m2p2b7 , then the output would be as follows in XML:

```
<?xml version="1.0" encoding="UTF-8" ?>
<response>
  <info>
    <statusCode>0</statusCode>
    <messages />
    <copyright>
      <imageUrl>http://tile21.mqcdn.com/res/mqlogo.gif</imageUrl>
      <imageAltText>© 2011 MapQuest, Inc.</imageAltText>
      <text>© 2011 MapQuest, Inc.</text>
    </copyright>
  </info>
  <route>
    <sessionId>4e7b9a9e-00db-0000-02b7-3efb-002655806132</sessionId>
    <options>
      <shapeFormat>raw</shapeFormat>
      <generalize>-1.0</generalize>
      <maxLinkId>0</maxLinkId>
      <narrativeType>none</narrativeType>
      <stateBoundaryDisplay>true</stateBoundaryDisplay>
      <countryBoundaryDisplay>true</countryBoundaryDisplay>
      <sideOfStreetDisplay>true</sideOfStreetDisplay>
      <destinationManeuverDisplay>true</destinationManeuverDisplay>
      <avoidTimedConditions>false</avoidTimedConditions>
      <enhancedNarrative>false</enhancedNarrative>
      <timeType>0</timeType>
      <routeType>SHORTEST</routeType>
      <locale>en_US</locale>
      <unit>K</unit>
      <tryAvoidLinkIds />
      <mustAvoidLinkIds />
      <manmaps>true</manmaps>
      <drivingStyle>2</drivingStyle>
      <highwayEfficiency>22.0</highwayEfficiency>
      <useTraffic>false</useTraffic>
    </options>
    <boundingBox>
      <ul>
        <lat>43.745819</lat>
        <lng>-79.407104</lng>
      </ul>
      <lr>
        <lat>43.694721</lat>
        <lng>-79.290466</lng>
      </lr>
    </boundingBox>
    <distance>15.026440620422363</distance>
    <time>1648</time>
    <formattedTime>00:27:28</formattedTime>
    <legs>
      <leg>
        <distance>15.026441</distance>
        <time>1648</time>
        <formattedTime>00:27:28</formattedTime>
        <index>0</index>
        <maneuvers>
          <maneuver>
            <startPoint>
              <lat>43.694721</lat>
              <lng>-79.291778</lng>
            </startPoint>
            <maneuverNotes />
            <distance>0.082076</distance>
            <time>14</time>
            <formattedTime>00:00:14</formattedTime>
```


Output 1. XML Output of Web Map Query

Let's take a closer look at the following options in the query string:

- outFormat=xml (the other option is JSON)
- unit=k (k for kilometer and m for mile)
- routeType=shortest (it could be fastest, if preferred)
- narrativeType=none (to get minimum data in the XML file)
- from=m4c5l8 (beginning postal code)
- to=m2p2b7 (end postal code)

In the XML output file, distance and time are identified in <distance> </distance> and <time> </time> tags. The time is in seconds and also in the formatted value. Reading in the distance and time values by positioning the pointer with a character variable technique is much faster, which takes less than a second.

WRITING THE MACRO

Step 1: Checking Dataset (optional)

Although our dataset contains valid postal codes but it is always prudent to check postal codes' formats. Any wrong or incomplete values in the dataset would give rise to unwanted results. The first step in this macro is to check that the postal codes have valid formats e.g. 6-character long with a format CNCNCN, where C is a character and N is a number for Canadian postal codes. If the postal codes are checked by other means, this step may be omitted. Invalid postal codes will be excluded to save time when querying MapQuest.

Step 2: Counting the Number of Postal Codes

This step counts the number (by creating a macro variable using call symputx function) of valid postal codes in the input dataset created by previous step. This number defines how many loops are needed.

Step 3: Looping Through Postal Code Pairs

The vital step starts here. This step will go through each pair of postal codes, construct an API query string for submission and processing. *Step 4: URL Access to API*

This step of SAS® URL access method connects to the MapQuest API, submits the query string and reads the output XML file. It is a good idea to check the returned XML file from the API on a browser to see the layout and structure. Before actually reading in the XML file, we first "peek" into the URL file reference using the sequential input mode to make sure that the Internet connection is available. This will avoid connection failure "hard error" and catch a "soft error" so that the program can exit gracefully

Please note the use of %nrstr to mask "&"in the query string.

Step 5: Parsing the XML

The program reads in the XML file one character at a time to calculate the file size, then it uses the "input" statement by positioning the pointer at '<statusCode>", "<distance>" and "<time>" to fetch their values. If the return status is not "OK", an error code (-2) will be written to the final data set. There could be several reasons when the status is not "OK" even after valid format of postal codes e.g. invalid or retired postal codes. When both the postal codes are same then the distance and time values would '0' (zero).

In this step, XML file is read in infile statement and the values of distance and time are fetched in input statement using the XML tags as position pointers. As the values of drive distance and time are appearing at the beginning so it stops as soon as it captures the first values after given XML tag.

Finally, it creates a clean dataset that contains the postal codes, distance and time values.

This parsing cannot be done so easily without the power of the API. The complete macro would appear as follows:

```
%MACRO distance_time(ds=, pc1=, pc2=,  out=);
     %local j  npc filesize p1 p2;
     PROC DATASETS lib=WORK memtype=data nolist;
       delete &out _pc_;
     QUIT;

/* Step 1: validate postal codes format */
     PROC SQL;
        create table _pc_ as
        select &pc1, &pc2
        from &ds
        where prxmatch('/[a-zA-Z]\d[a-zA-Z]\s?\d[a-zA-Z]\d/', &pc1)
          and prxmatch('/[a-zA-Z]\d[a-zA-Z]\s?\d[a-zA-Z]\d/', &pc2);
     QUIT;

/* Step 2: Count number of valid postal code pairs */
    DATA _null_;
       if 0 then set _pc_ nobs=obs;
       call symputx('npc',obs);
    RUN;

/* Step 3: Loop through each pair */
    %do j=1 %to &npc;
       DATA _null_;
          nrec = &j;
          set _pc_ point=nrec;
          call symputx('p1',&pc1);
           call symputx('p2',&pc2);
          stop;
       RUN;

/* Step 4: URL access to API */
       filename x url
"http://www.mapquestapi.com/directions/v1/route?key=(key)%nrstr(&outFormat=xml&narrati
veType=none&unit=k)%nrstr(&from)=&p1.%nrstr(&to)=&p2";
       filename z temp;
        %let url_flag = 0;
       DATA _null_;
          fid = fopen('x','S'); /* Check if Internet is available */
          if fid <= 0 then do; /* Internet is not available */
            time = datetime();
           stop = datetime() + 30; /* Max time to try connect */
           do while (fid <= 0 and time <=stop ); /* Loop while waiting... */
            fid = fopen('x','S'); /* Check if Internet is available */
            time = datetime(); /* Reset current time */
           end;
            if fid>0 then do;
            call symputx('url_flag',1);
            rc = fclose(fid);
            end;
          end; /* End if Internet is not available */
          else do;
           call symputx('url_flag',1);
             rc = fclose(fid);
```

```
          end;
       RUN;
          %if &url_flag=1 %then %do;
/* Read in XML file 1 character at a time, calculate the file size */
       DATA _NULL_;
          infile x recfm=f lrecl=1 end=eof;
          file z recfm=f lrecl=1;
          input @1 char $char1.;
          put @1 char $char1.;
          if eof;
          call symputx('filesize',_n_);
       RUN;
 /* Step 5: Parsing the XML */
       DATA tmp;
           keep p1 p2 distance_val time_val;
           p1="&p1";
           p2="&p2";
          infile z recfm=f lrecl=&filesize. eof=done dlm='<' scanover;
          input @'<statusCode>' status_code
                @'<distance>' distance_val
                @'<time>' time_val;
             if status_code ^= 0 then do;
            distance_val=-2;
             time_val=-2;
             end;
          output;
          stop;
          done:
          output;
       RUN;
       filename x clear;
       filename z clear;
       PROC APPEND base=&out data=tmp;
       RUN;
       PROC DATASETS lib=WORK memtype=data nolist;
           delete tmp;
       RUN;
       quit;
    %end;
       %else %do;
        filename x clear;
     filename z clear;
     %put ERROR: Cannot connect to Internet;
        %goto exit;
       %end;
    %end;
       %exit:
    PROC DATASETS lib=WORK memtype=data nolist;
          delete _pc_;
    RUN;
    QUIT;
%MEND;
```

RUNNING THE MACRO

The following SAS® dataset has 10 pairs of postal codes. Our goal is to determine the driving distances and times of these pairs of postal codes by using the above macro. Please note that these postal codes are randomly taken from the web for this paper only. The name of the dataset is “pc” and variable names are “p1” as origin and “p2” as destination postal codes.

Note that if origin and destination postal codes are reversed, the results may be different.

	p1	p2
1	L3M1P3	L0R1B6
2	P1H1H7	L0S1E3
3	L2R7C6	L0S1E7
4	L2A1Z2	L2A1L6
5	L5B1B8	L2A1R6
6	L2A1Z2	L2A1Z1
7	L2A1Z2	L2A1Z5
8	L2A1Z2	L2A2E4
9	L2A1Z2	L2A2E6
10	L2A1Z2	L2A2J3

Figure 4. Input Dataset

Let's run the above macro on this dataset of 10 pairs of postal codes and look at the output file. We found the desired values of distances and times.

```
%distance_time (ds=pc, pc1=p1, pc2=p2 out=dist_time);
```

EXAMINING THE OUTPUT

The final output is a clean dataset containing all the desired values. Please note that the unit of distance can be controlled in the API in either kilometers or in miles. The value of time is in seconds, which could be converted to hours and minutes.

p1	p2	distance_val	time_val
M2P2B7	K2A4H6	434.31851196	16762
M2P2B7	V8W2B7	4360.8217773	158799
M2P2B7	H3A2R7	532.08776855	20779
M2P2B7	A1C6H6	3070.2053223	137836
K2A4H6	v8W2B7	4385.9213867	173520
K2A4H6	H3A2R7	203.89744568	8512
K2A4H6	A1C6H6	2736.2663574	125510
V8W2B7	H3A2R7	4890.4912109	179122
V8W2B7	A1C6H6	7441.4438477	296517
H3A2R7	A1C6H6	2536.7512207	117730

Figure 5. Output Dataset

DISCUSSIONS

Injury data analysis which requires calculating driving distances and times has motivated us to explore the use of SAS to solve this problem. We found that this is an underutilized SAS analytical tool in our settings. When used with an appropriate API, this tool can assist us in obtaining answers to major research questions e.g. the strategic location of specialized medical facilities and its implications.

The output can be used for further statistical analysis e.g. mean driving distance and time between patients' homes to the nearest treatment centers. This can also be linked to treatment outcomes by distance groups.

We looked into various web APIs for this project – MapQuest, Google, Yahoo and Bing maps. Each has its own advantages and drawbacks. Our primary objective was to look for XML output option in one of these APIs that contained driving distance and time between a pair of points marked by postal codes. We used minimum options in this API query so that we could better understand the processes involved.

Parsing XML in SAS is not new, but fetching data in XML from an interactive website is. This method uses the combined power of SAS® and an API with reasonable control over the web data. It is both fast and reliable.

One potential limitation is related to web services. This method depends on reliable access to the Internet, so that Internet availability and its speed do matter. Some SAS® server deployments do not allow internet connection for privacy and security reasons. Another limitation is imposed by the API provider. For example, the MapQuest API Community Key limits 5,000 queries per day.

CONCLUSION

Nowadays, web APIs are used for many purposes ranging from complex visualizations to difficult calculations. Thanks to these APIs, many tasks are becoming easier in our data analyses. We have shown that by combining the power of an API and SAS®, it is possible to create a tool for the batch calculation of driving distances and times.

ACKNOWLEDGEMENT

The paper acknowledges the contribution of Claire Marie Fortin, Patricia Sidhom and Robert Williams of Clinical Registries Department of Canadian Institute for Health Information (CIHI).

REFERENCES

SAS Institute Inc. 2008. "*What's New in SAS® 9.2*." Cary, NC: SAS Institute Inc. Available at http://www.sas.com/offices/europe/denmark/pdf/whatsnew92.pdf

Okerson BB (2011). "Distance Mapping in Health and Health Care: SAS® as a Tool for Health Geomatics." Proceedings of SAS Global Forum Paper 299-2011. Cary, NC: SAS Institute Inc.

Zdeb M (2010). "Driving Distances and Times Using SAS® and Google Maps." Proceedings of SAS Global Forum Paper 050-2010. Cary, NC: SAS Institute Inc.

Shahid R, Bertazzon S, Knudtson ML, Ghali WA (2009). "Comparison of distance measures in spatial analytical modeling for health service planning." *BMC Health Services Research*. 9: 200. London, UK: BioMed Central. Available at http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2781002/.

Wikipedia "Application programming interface" Wikipedia: The Free Online Encyclopedia. October 2011 Available at http://en.wikipedia.org/wiki/Application_programming_interface.

CONTACT INFORMATION

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

Ash Roy/Yingbo Na
Canadian Institute for Health Information (CIHI)
4110 Yonge Street, Suite 300
Toronto, ON M2P 2B7
Phone: 416-481-2002
emails: ashoke.k.roy@gmail.com, yna@cihi.ca