University of Neuchatel

Institute of Economic Research

IRENE, Working paper 16-10

A Simple Command to Calculate Travel Distance and Travel Time

*Sylvain Weber** *Martin Péclat***

** University of Neuchâtel, Institute of Economic Research, Rue Abram-Louis Breguet 2, 2000 Neuchâtel, Switzerland. sylvain.weber@unine.ch.*

*** University of Neuchâtel, Institute of Economic Research, Rue Abram-Louis Breguet 2, 2000 Neuchâtel, Switzerland and University of Applied Sciences Western Switzerland (HES-SO Geneva), Rue de la Tambourine 17, 1227 Carouge, Switzerland. martin.peclat@unine.ch.*

A simple command to calculate travel distance and travel time[*]

Sylvain Weber[†] Martin Péclat[‡]

November 11, 2016

Abstract

Obtaining the routing distance between two addresses should not be a hassle in the current state of technology. This is unfortunately more complicated than it first seems. Recently, several Stata commands have been implemented for this purpose (`traveltime`, `traveltime3`, `mqtime`, `osrmtime`), but most of them went out of order only a few months after their introduction or appear as complicated to use. In this paper, we introduce the new command `georoute` to retrieve travel distance and travel time between two points, defined either by their addresses or by their geographical coordinates. Compared to other existing commands, we argue it is simple to use, efficient in terms of computational speed, and versatile regarding the information that can be provided as input.

JEL Classification: C87, R41.

Keywords: Stata, geocoding, travel distance, travel time.

[*]Financial support from the Swiss National Science Foundation (SNSF) grant N° 100018-144310 is gratefully acknowledged.

[†]University of Neuchâtel, Institute of Economic Research, Rue Abram-Louis Breguet 2, 2000 Neuchâtel, Switzerland. sylvain.weber@unine.ch.

[‡]University of Neuchâtel, Institute of Economic Research, Rue Abram-Louis Breguet 2, 2000 Neuchâtel, Switzerland and University of Applied Sciences Western Switzerland (HES-SO Geneva), Rue de la Tambourine 17, 1227 Carouge, Switzerland. martin.peclat@unine.ch.

1 Introduction

The demand for calculating routing distance between two geographical points is growing. In particular, researchers in energy economics (such as the authors of this paper) might be interested in knowing the travel distance between two places. Numerous applications in spatial econometrics also rely on such data. The development of large surveys containing addresses (e.g., of home and work places) has contributed to increasing the need for a systematic way of retrieving distances based on such information.

This paper follows a series of publications on the topic of geocoding in the Stata Journal (Ozimek and Miles, 2011; Voorheis, 2015; Huber and Rust, 2016) and several user-written commands available via SSC (Anderson, 2013; Ansari, 2015; Hess, 2015; Picard, 2010; Zeigermann, 2016). However, the domain of geocoding is moving fast, and most of these commands are now deprecated (see Huber and Rust, 2016, for a detailed account about which command is obsolete and why).

The command `georoute`[1] we introduce in this paper allows to retrieve travel distance and travel time between two points defined by their addresses or their geographical coordinates. Travel distance is the number of miles (or kilometers) one should drive by car to join the first point to the second. Travel time is how long it takes to drive the latter distance under normal traffic conditions. As these definitions make clear, the purpose of the new command is to provide information relevant for socio-economic research. Other existing commands (such as `geodist`, Picard, 2010) are dedicated to calculate straight line distance between two geographical coordinates.

The new command is close to `mqtime`, which is *in principle* also capable to retrieve travel distances and geographical coordinates from addresses. However, the behavior of `mqtime` appears inconstant, functioning sometimes but not always. Huber and Rust (2016) apparently tested `mqtime` in a bad day and concluded that "`mqtime` no longer works." The inconstancy of `mqtime` is probably due to MapQuest's Open API, which used to allow for an unlimited number of requests but has now altered its policy. For requests facing problems with MapQuest's Open API, `mqtime` uses the HERE API as an alternative to try to retrieve a distance. In `georoute`, we rely directly and only on the HERE API, which is managed by a com-

[1]Available from SSC. In Stata, type `ssc install georoute`.

mercial provider but offers a 90-day free evaluation plan allowing 100,000 requests per month. As a consequence, `georoute` is a number of times faster than `mqtime`.

Our command is also closely related to `osrmtime`, implemented by Huber and Rust (2016), but there are two major differences. First, `osrmtime` only accepts geographical coordinates (latitude, longitude) as input while `georoute` also accepts addresses. Second, we argue that `osrmtime` is quite complicated to use. Indeed, before running `osrmtime`, the user has to follow a series of prerequisites (in particular downloading and preparing the map files), some of which are quite involved and imply a substantial time investment from the user.[2] Contrarily, we argue that `georoute` is very user-friendly and the results obtained are very reliable. Starting from a database of addresses, a one line command will suffice to obtain what is wanted. The only prerequisite is to register for a HERE account and obtain an APP ID and an APP CODE.

2 The new command

2.1 Prerequisite: get an HERE account

Before running `georoute`, one has to visit `https://developer.here.com/` and register for a HERE account. HERE is a commercial provider, but it offers a 90-day free trial of its entire platform, which permits 100,000 requests per month. Such an account should be largely sufficient for most researchers and most empirical applications.

After having registered, the user should create an APP in order to get an APP ID and an APP CODE. These two elements are necessary to run `georoute`.[3]

[2]Honestly, it took a while before the authors of this paper were able to make `osrmtime` work. When trying to `osrmprepare` the maps for Europe as a whole, the process of installing the maps got stuck for a while "trying to extend the external memory space..." and then definitely while "building node id map ...". Preparing the maps for a single country (in our case a small one: Switzerland) was less problematic. Nevertheless, `osrmtime` then yielded some strange (not to say dangerous) outcomes when coordinates outside the country were specified. Instead of excluding such observations, `osrmtime` calculated a distance and a duration that were clearly incorrect. The return code, supposed to signal any issue, was nevertheless set as "OK" for these observations.

[3]The APP ID should be a 20-character series such as "BfSfwSlKMCPHj5WbVJ1g",

2.2 The georoute command

The syntax of `georoute` is as follows:

`georoute` [*if*] [*in*], `hereid(`*string*`)` `herecode(`*string*`)` { `startaddress(`*varlist*`)` | `startxy(`*varlist*`)` } { `endaddress(`*varlist*`)` | `endxy(`*varlist*`)` } [`km` `distance(`*newvarname*`)` `time(`*newvarname*`)` `coordinates(str1 str2)` `timer` `herepaid` `replace`]

`hereid` and `herecode` are compulsory. They indicate the APP ID and APP CODE of the user.

`startaddress` and `endaddress` specify the addresses of the starting and ending points. Addresses can be inserted as a single variable or as a list of variables. Alternatively, `startxy` and `endxy` can be used. Either `startaddress` or `startxy` is required. Either `endaddress` or `endxy` is required.

`startxy` and `endxy` specify the coordinates in decimal degrees of the starting and ending points. They can be used as an alternative to `startaddress` and `endaddress`. Two numeric variables containing x and y coordinates of the starting (ending) point should be provided in `startxy` (`endxy`).

`km` specifies that distances should be returned in kilometers. The default is to return distances in miles.

`distance(`*newvarname*`)` creates a new variable containing the travel distance between the two addresses. If not specified, distance will be stored in a variable named *travel_distance*.

`time(`*newvarname*`)` creates a new variable containing the travel time (by car and under normal traffic conditions) between the two addresses. If not specified, time will be stored in a variable named *travel_time*.

`coordinates(`*str1 str2*`)` creates the new variables *str1_x*, *str1_y*, *str1_match*, *str2_x*, *str2_y*, and *str2_match*, that contain the coordinates and the match code of the starting (*str1_x*, *str1_y*, *str1_match*) and ending (*str2_x*, *str2_y*, *str2_match*) addresses. If `coordinates` is not specified, coordinates and match code are not saved. The match code indicates how well the result matches the request in a 4-point scale: 1=exact, 2=ambiguous, 3=upHierarchy, 4=ambiguousUpHierarchy.

and the APP CODE a 22-character series such as "bFw1UDZM3Zgc4QM8lyknVg". The authors of this paper find it useless to provide their own APP ID and APP CODE given that the maximal number of requests would be exceeded very fast if these were made available to all Stata users.

`herepaid` allows the user who owns a paid HERE plan to specify it. This option will simply alter the url used for the API requests so as to comply with HERE policy.[4]

`timer` requests that a timer is printed while geocoding. If specified, a dot is printed for every centile of the dataset that has been geocoded and the number corresponding to every decile is printed. If distances are calculated based on addresses (and not geographical coordinates), two different timers will appear successively: one while geocoding addresses and one while geocoding routes. When geocoding large numbers of observations, this option will let the user when to expect the end.

`replace` specifies that the variables in `distance`, `time`, and `coordinates` be replaced if they already exist in the database. It should be used cautiously because it might definitively drop some data.

2.3 The georoutei command

In order to facilitate quick requests for a single pair of addresses or coordinates, we also implemented an immediate command where all arguments must be specified interactively. The syntax of `georoutei` is as follows:

`georoutei , hereid(`*string*`) herecode(`*string*`)` { `startaddress(`*string*`)` | `startxy(`#x`,`#y`)` } { `endaddress(`*string*`)` | `endxy(`#x`,`#y`)` } [`km herepaid`]

`hereid`, `herecode`, `km`, and `herepaid` are exactly as described above in Section 2.2.

`startaddress(`*string*`)` and `endaddress(`*string*`)` specify the addresses of the starting and ending points. Addresses must simply be typed within the parentheses. Alternatively, `startxy` and `endxy` can be used. Either `startaddress` or `startxy` is required. Either `endaddress` or `endxy` is required.

`startxy(`#x`,`#y`)` and `endxy(`#x`,`#y`)` specify the coordinates in decimal degrees of the starting and ending points. They can be used as an alternative to `startaddress` and `endaddress`. Coordinates must be specified as two numbers separated by a comma.

[4]See `https://developer.here.com/rest-apis/documentation/geocoder/common/request-cit-environment-rest.html`.

2.3.1 Saved results

`georoutei` saves the following results in `r()`:

Scalars			
`r(dist)`	Travel distance	`r(time)`	Travel time
Macros			
`r(start)`	Coordinates of starting point	`r(end)`	Coordinates of ending point

3 Examples

In order to illustrate the functioning of `georoute`, let's build up a small dataset:[5]

```
. *Starting points
. input str25 strt1 zip1 str15 city1 str11 cntry1
. "Rue de la Tambourine 17" 1227 "Carouge" "Switzerland"
. "Place de la Gare" 1003 "Lausanne" "Switzerland"
. "" . "Paris" "France"
. "Place de la Gare" 1003 "Lausanne" "Switzerland"
. end
. *Ending points
. input str25 strt2 zip2 str15 city2 str11 cntry2
. "Rue Abram-Louis Breguet 2" 2000 "Neuchatel" "Switzerland"
. "Avenue Jean Leger" 74500 "Evian" "France"
. "" . "New York" "USA"
. "Place de Cornavin" 1203 "Geneva" "Switzerland"
. georoute, hereid(BfSfwSlKMCPHj5WbVJ1g) herecode(bFw1UDZM3Zgc4QM8lyknVg)
> startad(strt1 zip1 city1 cntry1) endad(strt2 zip2 city2 cntry2) km di(dist)
> ti(time) co(p1 p2)
. format dist time %5.2f
. list city1 cntry1 city2 cntry2 dist time

     +--------------------------------------------------------------------+
     |    city1        cntry1       city2        cntry2     dist     time |
     |--------------------------------------------------------------------|
  1. |  Carouge   Switzerland   Neuchatel   Switzerland   135.67    86.88 |
  2. | Lausanne   Switzerland       Evian        France    67.19    65.68 |
  3. |    Paris        France    New York           USA        .        . |
  4. | Lausanne   Switzerland      Geneva   Switzerland    61.85    43.98 |
     +--------------------------------------------------------------------+
```

For the record, the first observation contains the office addresses of the two authors of this paper. Each of them living close to the city where the other works, the outcome reveals more or less the travel distance both have to cover daily, back and forth.

[5]The reader should be warned that simply introducing the following lines in Stata will not yield any result, because the APP ID and APP CODE displayed here are fake. In order to replicate the results, include your own APP ID and APP CODE (see Section 2.1).

The second observation was chosen so as to demonstrate an essential feature of `georoute`. Both the cities of Lausanne and Evian are in fact located on the shores of Geneva's Lake: Lausanne at the North, Evian at the South. By car, one would have to go all around the lake and the distance would be 67 kilometers. However, connecting these two cities by a straight line would give 13 kilometers, as shown by the output of `geodist`:

```
. geodist p1_x p1_y p2_x p2_y, gen(distlin)
. format distlin %5.2f
. format p?_? %3.2f
. l city1 p1_x p1_y city2 p2_x p2_y dist distlin

     +------------------------------------------------------------------------+
     |     city1    p1_x   p1_y       city2    p2_x     p2_y     dist   distlin |
     |------------------------------------------------------------------------|
  1. |   Carouge   46.18   6.14   Neuchatel   46.99     6.94   135.67    109.69 |
  2. |  Lausanne   46.52   6.63       Evian   46.40     6.59    67.19     13.25 |
  3. |     Paris   48.86   2.34    New York   40.71   -74.01        .   5852.14 |
  4. |  Lausanne   46.52   6.63      Geneva   46.21     6.14    61.85     50.58 |
     +------------------------------------------------------------------------+
```

On the other hand, one may notice with the third observation that no distance is computed by `georoute` between Paris and New York (for obvious reasons) but `geodist` indicates the geodetic distance as being almost 6,000 km. The purpose of these two commands is different and which distance (travel distance from `georoute` or geodetic distance from `geodist`) to use depends on the goal of the user.

It should also be noted that `geodist` could be used thanks to the latitudes and longitudes (variables *p1_x*, *p1_y*, *p2_x*, and *p2_y*) previously produced by `georoute` with option `coordinates`. In that sense, these two commands are complementary. Furthermore, note that `georoute` is quite versatile in terms of how addresses can be specified. If several variables should be combined to produce the entire address, these different variables, be they string or numeric, can be simply introduced in the `startaddress` and `endaddress` options as a variable list.

By comparing the second and fourth observations, another interesting feature of `georoute` can be highlighted. While the routing distance is almost identical for Lausanne-Evian and for Lausanne-Geneva, one may notice that travel time is much lower for the latter. This is due to the fact that most of the travel between Lausanne and Geneva can be done on a highway, while a large share of the travel between Lausanne and Evian takes place on regional roads with much lower speed limits. Distance and time are thus two different dimensions of a travel and both might be useful in empirical applications.

Finally, let's assume we want to check one of the results obtained above. In such case, the immediate command `georoutei` would be very convenient. For instance, one could obtain the results for the first observation as follows:[6]

```
. georoutei, hereid(BfSfwSlKMCPHj5WbVJ1g) herecode(bFw1UDZM3Zgc4QM8lyknVg)
> startad(Rue de la Tambourine 17, 1227 Carouge, Switzerland)
> endad(Rue Abram-Louis Breguet 2, 2000 Neuchatel, Switzerland) km
-------------------------------------------------------------------------------
From: Rue de la Tambourine 17, 1227 Carouge, Switzerland (46.1761413,6.1393099)
To:   Rue Abram-Louis Breguet 2, 2000 Neuchatel, Switzerland (46.99382,6.94049)
-------------------------------------------------------------------------------
Travel distance:     135.67 kilometers
Travel time:          86.88 minutes
```

Given that we also know latitudes and longitudes corresponding to the addresses from the previous call of `georoute`, we could as well provide this information to `georoutei`:

```
. georoutei, hereid(BfSfwSlKMCPHj5WbVJ1g) herecode(bFw1UDZM3Zgc4QM8lyknVg)
> startxy(46.1761413,6.1393099) endxy(46.99382,6.94049) km
----------------------------
From: (46.1761413,6.1393099)
To:   (46.99382,6.94049)
----------------------------
Travel distance:     135.67 kilometers
Travel time:          86.88 minutes
```

We emphasize that not only travel distance is the same as before, but so is travel time. In our opinion, this is an important feature of the HERE API: it provides travel time under normal traffic conditions. Said otherwise, the results will not be influenced by current traffic conditions, which is essential in terms of reproducibility. Whenever `georoute` and `georoutei` are run, results will be identical (unless of course roads have been built or closed in the meantime).

4 Conclusion

The techniques for geocoding evolve at a rapid pace. As a consequence, new commands appear but depreciate rapidly as well. This article introduces the new command `georoute`, which computes travel distance and travel time between two points defined by addresses or their geographical coordinates. This command is expected to remain in use for a while.

[6]Note that in order to get strictly identical results when using addresses and coordinates, one must be careful to include all available digits in the latitudes and longitudes.

The only uncertainty lies in potential changes in the terms of use of the HERE API, which is managed by a commercial provider. In order to minimize the obsolescence risk, the command forces the user to use his own HERE account, which can be created free of charge while benefiting from a substantial number of requests.

Compared to existing commands that have a similar purpose and are still in operation, the new command possesses several advantages. Compared to `mqtime`, `georoute` is computationally efficient, versatile regarding how addresses can be specified, and it encompasses a number of additional options. The behavior of `mqtime` moreover seems erratic, in the sense that is does not always work, while many checks have not revealed any inconstancy in the functioning of `georoute`. Compared to `osrmtime`, `georoute` is objectively simpler to use and has the advantage that it can be used on addresses while the former command can only obtain distance between coordinates and might thus require a first step to geocode addresses if this is the only information initially available. Hopefully, `georoute` should facilitate the life of a number of researchers for a long time.

References

Anderson, M. L. (2013). GEOCODEOPEN: Stata module to geocode addresses using MapQuest Open Geocoding Services and Open Street Maps. Statistical Software Components, Boston College Department of Economics.

Ansari, M. R. (2015). GCODE: Stata module to download Google geocode data. Statistical Software Components, Boston College Department of Economics.

Hess, S. (2015). GEOCODEHERE: Stata module to provide geocoding relying on Nokia's Here Maps API. Statistical Software Components, Boston College Department of Economics.

Huber, S. and Rust, C. (2016). Calculate travel time and distance with OpenStreetMap data using the Open Source Routing Machine (OSRM). *Stata Journal*, 16(2):416–423.

Ozimek, A. and Miles, D. (2011). Stata utilities for geocoding and generating travel time and travel distance information. *Stata Journal*, 11(1):106–119.

Picard, R. (2010). GEODIST: Stata module to compute geodetic distances. Statistical Software Components, Boston College Department of Economics.

Voorheis, J. (2015). mqtime: A stata tool for calculating travel time and distance using MapQuest web services. *Stata Journal*, 15(3):845–853.

Zeigermann, L. (2016). OPENCAGEGEO: Stata module for forward and reverse geocoding using the OpenCage Geocoder API. Statistical Software Components, Boston College Department of Economics.