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

UNIVERSITÉ DE NEUCHÂTEL

Institut de recherches économiques

A simple command to calculate travel distance and travel time*

Sylvain Weber† Martin Péclat‡

September 26, 2017

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.

*This research was financially supported by the Swiss National Science Foundation (SNSF), grant N° 100018-144310, and is part of the activities of SCCER CREST, which is financially supported by the Swiss Commission for Technology and Innovation (CTI).

†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 moving fast, most of these commands are now deprecated (see Huber and Rust, 2016, for a detailed account about which command is obsolete and why).

In this paper, we introduce the user-written command `georoute`, which 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 command is to provide relevant information for socio-economic research. Other existing commands (such as `geodist`, Picard, 2010) are dedicated to calculate straight line distance between two geographical coordinates, which might be relevant in different contexts.

The command `georoute` 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 commercial provider but offers free plans allowing large numbers of requests per month. As a consequence, `georoute` is a number of times

faster than `mqtime`. Also, while `georoute` cannot be guaranteed to remain operational over the long run because it depends on the stability of the HERE API, the risk of depreciation is minimized by forcing the user to create and use his own HERE account.

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 upfront time investment from the user.[1] 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 is sufficient 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

In order to use `georoute`, a HERE account (https://developer.here.com/) is necessary. HERE is a commercial provider, but it offers a 90-day free trial of its entire platform, which permits 100,000 requests per month, and a free Public Basic Plan, which is not limited in time and permits 15,000 requests per month.[2] Such accounts should be largely sufficient for most researchers and most empirical applications. It should nevertheless

[1] Honestly, it took a while before the authors of this paper were able to get something out of `osrmtime`. When trying to `osrmprepare` the maps for Europe as a whole, the process got stuck for a while "trying to extend the external memory space..." and then froze 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 travel distance and a travel duration that were clearly incorrect. The return code, supposed to signal any issue, was nevertheless set as "OK" for these observations.

[2] Moreover, in our experience, it is possible to reactivate a new 90-days free trial once it expires, using the same HERE account.

be mentioned that georoute will make three requests for computing a single routing distance when addresses are provided (one geocoding request per address plus one routing request for calculating the distance between the two points). The maximal number of travel distances that can be calculated in this case is thus one third of the above-mentioned limits. If geographical coordinates are directly specified instead of addresses, only one routing request will be necessary.

After having registered in HERE, the user should create a project and get an APP ID and an APP CODE (JavaScript/REST). These two elements are necessary to run georoute.[3] Also, note that a delay of around two hours may occur between the creation of the HERE project and its activation.

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*) diagnostic(*newvarname*) coordinates(str1 str2) replace herepaid timer pause]

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.

Note: the presence of special characters (e.g. French accents) in addresses might cause errors in the geocoding process. Such characters should be transformed before running georoute, e.g. using subinstr().

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

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

(latitude) and y (longitude) coordinates of the starting and ending points should be provided in `startxy` and `endxy`.
Note: x (latitude) must be between -90 and 90. y (longitude) must be between -180 and 180. Examples:

- United States Capitol: 38.8897, -77.0089
- Eiffel Tower: 48.8584, 2.2923
- Cape Horn: -55.9859, -67.2743
- Pearl Tower: 31.2378, 121.5225

`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*.

`diagnostic` creates a new variable containing a diagnostic code for the geocoding and georouting outcome of each observation in the database: 0 = OK, 1 = No route found, 2 = Start and/or end not geocoded, 3 = Start and/or end coordinates missing. If not specified, diagnostic codes will be stored in a variable named *georoute_diagnostic*.

`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.

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

`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 (see https://developer.here.com/rest-apis/documentation/geocoder/common/request-cit-environment-rest.html).

`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.

`pause` can be used to slow the geocoding process by asking Stata to sleep for 30 seconds every 100^{th} observation. This could be useful for large databases, which might overload the HERE API and result in missing values for batches of observations.

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 (latitude and longitude) must be specified as two numbers separated by a comma.

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:[4]

```
. *Starting points
. input str25 strt1 zip1 str15 city1 str11 cntry1
. "Rue de la Tambourine 17" 1227 "Carouge" "Switzerland"
. "" 1003 "Lausanne" "Switzerland"
. "" . "Paris" "France"
. "" 1003 "Lausanne" "Switzerland"
. end
. *Ending points
. input str25 strt2 zip2 str15 city2 str11 cntry2
. "Rue Abram-Louis Breguet 2" 2000 "Neuchatel" "Switzerland"
. "" 74500 "Evian" "France"
. "" . "New York" "USA"
. "" 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 %7.2f
. list city1 cntry1 city2 cntry2 dist time
```

	city1	cntry1	city2	cntry2	dist	time
1.	Carouge	Switzerland	Neuchatel	Switzerland	135.68	87.02
2.	Lausanne	Switzerland	Evian	France	73.22	77.73
3.	Paris	France	New York	USA	.	.
4.	Lausanne	Switzerland	Geneva	Switzerland	64.53	47.12

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

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

the other works, the outcome reveals more or less the travel distance both have to cover daily, back and forth.

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 `geodist`'s output:

```
. geodist p1_x p1_y p2_x p2_y, gen(distlin)
. format distlin %7.2f
. format p?_? %5.2f
. 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	135.68	109.76
2.	Lausanne	46.52	6.63	Evian	46.36	6.65	73.22	17.75
3.	Paris	48.86	2.34	New York	40.71	-74.01	.	5852.14
4.	Lausanne	46.52	6.63	Geneva	46.21	6.12	64.53	52.17

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. In less obvious cases, where doubts might remain about the reason why no distance was obtained with `georoute`, variable *georoute_diagnostic* could offer some guidance:

```
. l city1 city2 georoute_diagnostic
```

	city1	city2	georoute_dia~c
1.	Carouge	Neuchatel	OK
2.	Lausanne	Evian	OK
3.	Paris	New York	No route found
4.	Lausanne	Geneva	OK

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 regarding 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 routing distances are com-

parable for Lausanne-Evian and 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 us 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:

```
. georoutei, hereid(BfSfwSlKMCPHj5WbVJ1g) herecode(bFw1UDZM3Zgc4QM8lyknVg)
> startad(Rue de la Tambourine 17, 1227 Carouge, Switzerland) endad(Rue Abram-Lo
> uis Breguet 2, 2000 Neuchatel, Switzerland) km
------------------------------------------------------------------------------
From: Rue de la Tambourine 17, 1227 Carouge, Switzerland (46.17556,6.13906)
To:   Rue Abram-Louis Breguet 2, 2000 Neuchatel, Switzerland (46.99382,6.94049)
------------------------------------------------------------------------------
Travel distance:     135.68 kilometers
Travel time:          87.02 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`:[5]

```
. 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:     135.68 kilometers
Travel time:          87.02 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).

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

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 user-written command `georoute`, which computes travel distance and travel time between two points defined by their addresses or their geographical coordinates. As for its predecessors, it cannot be guaranteed that `georoute` will work in the long run, because it depends on whether the commercial provider HERE will maintain its API unchanged or alter its terms of use. Nevertheless, we have tried to minimize the risk of obsolescence by forcing 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, `georoute` possesses several advantages. Compared to `mqtime`, it 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.