AgEcon SEARCH
RESEARCH IN AGRICULTURAL & APPLIED ECONOMICS

The World's Largest Open Access Agricultural & Applied Economics Digital Library

This document is discoverable and free to researchers across the globe due to the work of AgEcon Search.

Help ensure our sustainability.

Give to AgEcon Search

AgEcon Search
http://ageconsearch.umn.edu
aesearch@umn.edu

*Papers downloaded from **AgEcon Search** may be used for non-commercial purposes and personal study only. No other use, including posting to another Internet site, is permitted without permission from the copyright owner (not AgEcon Search), or as allowed under the provisions of Fair Use, U.S. Copyright Act, Title 17 U.S.C.*

No endorsement of AgEcon Search or its fundraising activities by the author(s) of the following work or their employer(s) is intended or implied.

Check for updates

The Stata Journal (2017)
17, Number 4, pp. 962–971

A simple command to calculate travel distance and travel time

Sylvain Weber
University of Neuchâtel
Institute of Economic Research
Neuchâtel, Switzerland
sylvain.weber@unine.ch

Martin Péclat
University of Neuchâtel
Institute of Economic Research
Neuchâtel, Switzerland
martin.peclat@unine.ch

Abstract. Obtaining the routing distance between two addresses should not be a hassle with current technology. Unfortunately, this is more complicated than it first seems. Recently, several commands have been implemented for this purpose (`traveltime`, `traveltime3`, `mqtime`, `osrmtime`), but most of them became obsolete only a few months after their introduction or appear complicated to use. In this article, we introduce the community-contributed command `georoute`, which retrieves travel distance and travel time between two points defined either by their addresses or by their geographical coordinates. Compared with other existing commands, it is simple to use, efficient in terms of computational speed, and versatile regarding the information that can be provided as input.

Keywords: dm0092, georoute, georoutei, geocoding, travel distance, travel time

1 Introduction

The demand for calculating routing distance between two geographical points is growing. Researchers in energy economics (such as the authors of this article) might be interested in knowing the travel distance between two places for various reasons. Numerous applications in spatial econometrics also rely on such data. The development of large surveys containing addresses (for example, of homes and workplaces) has increased the usefulness of systems allowing to retrieve distances based on such information.

In this article, we follow 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 community-contributed commands available via Statistical Software Components (Anderson 2013; Ansari 2015; Heß 2015; Picard 2010; Zeigermann 2016). However, because the geocoding field is progressing quickly, most of these commands are now obsolete (see Huber and Rust [2016] for a detailed account about what commands are obsolete and why).

In this article, we introduce the community-contributed command `georoute`, which retrieves 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. These definitions clarify that the purpose of `georoute` is to provide relevant information for socioeconomic

© 2017 StataCorp LLC

research. Other existing commands, such as `geodist` (Picard 2010), calculate straight-line distance between two geographical coordinates, which might be relevant in different contexts.

The command `georoute` is close to `mqtime` (Voorheis 2015), which is principally also capable of retrieving travel distances and geographical coordinates from addresses. However, `mqtime` does not always function correctly. Huber and Rust (2016) apparently tested `mqtime` on a bad day and concluded that "`mqtime` no longer works". The inconsistency of `mqtime` is probably due to MapQuest's open application programming interface (API), which previously allowed 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 that allow large numbers of requests per month. Thus, `georoute` is many times faster than `mqtime`. `georoute` cannot be guaranteed to remain operational in the long run, because it depends on the stability of the HERE API. The risk of depreciation is minimized by forcing users to create and use their own HERE accounts.

Our command is also closely related to `osrmtime`, implemented by Huber and Rust (2016), but there are two major differences: First, `osrmtime` accepts only 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 (particularly downloading and preparing the map files), some of which are quite involved and imply a substantial up-front time investment from the user.[1] Contrarily, `georoute` is user friendly and delivers reliable results. Starting from a database of addresses, a user simply needs a one-line command. The only prerequisites are to register for a HERE account and obtain an *App ID* and an *App Code*.

2 The georoute and georoutei commands

2.1 Prerequisite: Get an HERE account

To use `georoute`, one needs a HERE account (https://developer.here.com/). 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. However, `georoute`

1. When we tried to `osrmprepare` the maps for Europe as a whole, the process got stuck while `trying to extend the external memory space...` and then froze completely while `building node id map ....` Preparing the maps for a single country (in our case, a small one, Switzerland) was less problematic. Nevertheless, `osrmtime` yielded some strange (not to say harmful) outcomes when coordinates outside the country were specified. Instead of excluding such observations, `osrmtime` calculated a clearly incorrect travel distance and duration. 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-day free trial once it expires, using the same HERE account.

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). Thus, the maximal number of travel distances that can be calculated in this case is one-third of the above-mentioned limits. If geographical coordinates are directly specified instead of addresses, only one routing request will be necessary.

After registering in HERE, the user should create a project and get an *App ID* and *App Code* ("JavaScript/REST"). These two elements are necessary for `georoute`.[3] 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

Syntax

The syntax of `georoute` is as follows:

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

Options

`hereid(`*string*`)` and `herecode(`*string*`)` indicate the *App ID* and *App Code* of the user. `hereid()` and `herecode()` are required.

`startaddress(`*varlist*`)` and `endaddress(`*varlist*`)` 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 that the presence of special characters (for example, French accents) in addresses might cause errors in the geocoding process. Such characters should be transformed before running `georoute`, for example, using `subinstr()`.

`startxy(`*varlist*`)` and `endxy(`*varlist*`)` 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 (latitude) and y (longitude) coordinates of the starting and ending points should be provided in `startxy()` and

3. The *App ID* should be a 20-character series such as `BfSfwSlKMCPHj5WbVJ1g`, and the *App Code* a 22-character series such as `bFw1UDZM3Zgc4QM8lyknVg`. We find it useless to provide our own *App ID* and *App Code*, because the maximal number of requests would be exceeded quickly if these were made available to all Stata users.

`endxy()`. Note that x (latitude) must be between -90 and 90 and that 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(`*newvar*`)` creates a new variable containing the travel distances between pairs of addresses or geographical points. By default, travel distances will be stored in a variable named `travel_distance`.

`time(`*newvar*`)` creates a new variable containing the travel times (by car and under normal traffic conditions) between pairs of addresses or geographical points. By default, travel times will be stored in a variable named `travel_time`.

`diagnostic(`*newvar*`)` 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`. By default, 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`, which 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. By default, coordinates and match codes 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()`, `diagnostic()`, and `coordinates()` be replaced if they already exist in the database. It should be used cautiously because it might cause some data to be lost.

`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 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 be printed while geocoding. If specified, a dot is printed for every geocoded centile of the dataset, 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 inform the user on the expected end time.

`pause` can be used to slow the geocoding process by asking Stata to sleep for 30 seconds every 100th 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

Syntax

For quick requests for a single pair of addresses or coordinates, we implemented the immediate command `georoutei`, where all arguments must be specified interactively. The syntax of `georoutei` is

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

Options

`hereid()`, `herecode()`, `km`, and `herepaid` are exactly as described 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.

Stored results

`georoutei` stores the following 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

To illustrate `georoute`, let's build a small dataset:[4]

```
. * Starting points
. input str25 strt1 zip1 str15 city1 str11 cntry1
                        strt1       zip1            city1       cntry1
  1. "Rue de la Tambourine 17" 1227 "Carouge" "Switzerland"
  2. "" 1003 "Lausanne" "Switzerland"
  3. "" . "Paris" "France"
  4. "" 1003 "Lausanne" "Switzerland"
  5. end
. * Ending points
. input str25 strt2 zip2 str15 city2 str11 cntry2
                        strt2       zip2            city2       cntry2
  1. "Rue Abram-Louis Breguet 2" 2000 "Neuchatel" "Switzerland"
  2. "" 74500 "Evian" "France"
  3. "" . "New York" "USA"
  4. "" 1203 "Geneva" "Switzerland"
. *Compute distances using georoute
. georoute, hereid(BfSfwSlKMCPHj5WbVJ1g) herecode(bFw1UDZM3Zgc4QM8lyknVg)
> startad(strt1 zip1 city1 cntry1)
> endad(strt2 zip2 city2 cntry2) km distance(dist) time(time) coordinates(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 article. Both of them live close to the city where the other works; the outcome essentially reveals their daily back-and-forth travel distance.

The second observation was chosen to demonstrate an essential feature of `georoute`. Both the cities of Lausanne and Evian are located on the shores of Geneva's Lake: Lausanne in the north and Evian in the south. By car, one would have to drive around the lake, a 73.22-kilometer distance. However, connecting these 2 cities by a straight line would result in 17.75 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
```

4. 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 invalid. To replicate the results, include your own *App ID* and *App Code* (see section 2.1).

```
. list 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 purposes of these two commands are 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`, the variable `georoute_diagnostic` could offer some guidance:

```
. list 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

Note also 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 the 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, we see another interesting feature of `georoute`. While routing distances are comparable for Lausanne–Evian and Lausanne–Geneva, one may notice that travel time is much lower for the latter. This is because most 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 travel, and both might be useful in empirical applications.

Finally, let us assume we want to check one of the results obtained above. In this case, the immediate command `georoutei` would be 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-Louis 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 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 both travel distance and time are the same as before. 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. Consequently, new commands appear and depreciate rapidly. In this article, we introduce the community-contributed command `georoute`, which computes travel distance and time between two points defined by their addresses or their geographical coordinates. Like its predecessors, the longevity of `georoute` 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 users to use their own HERE accounts, which are free and benefit from a substantial number of requests.

Compared with existing commands that have a similar purpose and are still in operation, `georoute` possesses several advantages. Compared with `mqtime`, it is computationally efficient and versatile regarding how addresses can be specified, and it encompasses many additional options. Moreover, `mqtime` does not always work, whereas many checks have not revealed any inconsistency in `georoute`. Compared with `osrmtime`, `georoute` is objectively simpler to use and can be used on addresses, while `osrmtime` can obtain distances only between coordinates and might thus require a first step to geocode ad-

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

dresses if this is the only information initially available. Hopefully, `georoute` should aid researchers for a long time.

5 Acknowledgments

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

6 References

Anderson, M. L. 2013. geocodeopen: Stata module to geocode addresses using MapQuest Open Geocoding Services and Open Street Maps. Statistical Software Components S457733, Department of Economics, Boston College. https://ideas.repec.org/c/boc/bocode/s457733.html.

Ansari, M. R. 2015. gcode: Stata module to download Google geocode data. Statistical Software Components S457969, Department of Economics, Boston College. https://ideas.repec.org/c/boc/bocode/s457969.html.

Heß, S. 2015. geocodehere: Stata module to provide geocoding relying on Nokia's Here Maps API. Statistical Software Components S457969, Department of Economics, Boston College. https://ideas.repec.org/c/boc/bocode/s458048.html.

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

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

Picard, R. 2010. geodist: Stata module to compute geodetic distances. Statistical Software Components S457147, Department of Economics, Boston College. https://ideas.repec.org/c/boc/bocode/s457147.html.

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

Zeigermann, L. 2016. opencagegeo: Stata module for forward and reverse geocoding using the OpenCage Geocoder API. Statistical Software Components S458155, Department of Economics, Boston College. https://ideas.repec.org/c/boc/bocode/s458155.html.

About the authors

Sylvain Weber is a postdoctoral researcher at the Institute of Economic Research at the University of Neuchâtel (Switzerland). His main field of research is energy economics. In particular, he studies private mobility, hence his interest for calculating travel distances precisely and easily.

Martin Péclat is a PhD student at the University of Neuchâtel (Switzerland) and at the Geneva School of Business Administration HES-SO, University of Applied Sciences of Western Switzerland (Switzerland). His thesis is about the determinants of spatial diffusion and adoption of solar photovoltaic technology in Switzerland.