Thirsty Salesman Problem

Solving TSP using R in context of {sf} & HERE API

My Alma Mater, Prague School of Economics, is located in Žižkov. A formerly working class neighborhood, now rather gentrified, it has to this day retained some traces of its former rougher edges. One of these is an active night life.

A crawl through the bars of Žižkov is therefore a familiar activity for many VŠE students, and can serve as a gateway drug for serious optimization techniques. Such as the Travelling Salesman Problem.

The TSP is an optimization classic, with a number of well understood and highly standardized solutions available in the context of statistical programming language R.

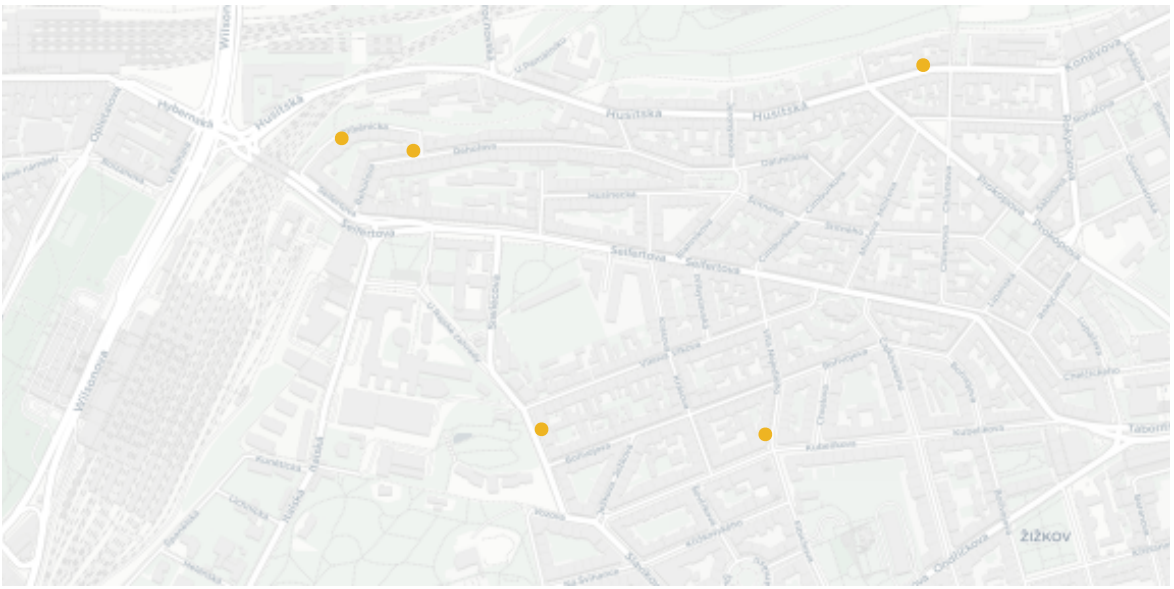
In this blog post I would like to share a practical example of solving the TSP using Open Street Map data of bars via `{osmdata}` and HERE routing engine via `{hereR}`. The actual solution will be found by utilizing the `{TSP}` package.

The first step in our exercise is acquiring data of Žižkov bars. A search is performed over the area of *core Žižkov*, defined as a polygon, using the OSM Overpass API.

As there seems not to be a clear consensus over what constitutes a bar, restaurant or a pub in Prague I am including all three of the possible amenities.

We have located 74 bars, implying a distance matrix of 5476 elements. Not a huge one by today's standards — but big enough to think twice about trying to solve using a pen and a piece of paper.

I have found that while it is not overly difficult to solve the TSP for *all* the Žižkov bars there is educational value in running the TSP over only a small sample. I have found it advantageous to be able to actually show the distance matrix — and this page will easily accommodate only about a 5×5 matrix.



The easiest distance matrix to calculate is plain "as the crow flies" distance. This can be calculated via a `sf::st_distance()` call.

The resulting matrix will be based on pure distance, with some differences in interpretation depending on co-ordinate reference system of underlying data (Euclidean in projected CRS and spherical in unprojected CRS).

```
## Units: [m]
##                      Taiko Ramen Bar Belzepub Palác Akropolis District 3 Bar
## Taiko Ramen Bar               0.0000 793.2878        603.8839       777.3224
## Belzepub                    793.2878   0.0000        336.6404       461.1727
## Palác Akropolis             603.8839 336.6404          0.0000       679.7744
## District 3 Bar              777.3224 461.1727        679.7744         0.0000
## Final Art Music Club        881.3059 530.8086        777.1854       109.3920
##                      Final Art Music Club
## Taiko Ramen Bar                  881.3059
## Belzepub                         530.8086
## Palác Akropolis                  777.1854
## District 3 Bar                   109.3920
## Final Art Music Club               0.0000
```

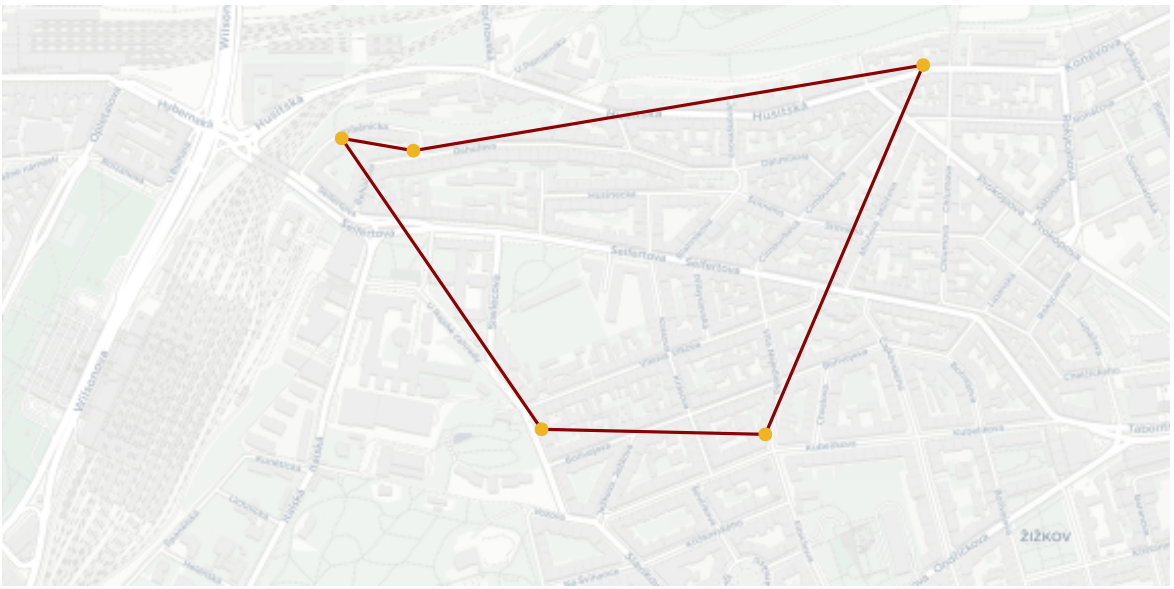
Calculating the distance matrix using plain distance is easy, and the resulting matrix is symmetrical (distance from A to B equals distance from B to A). It is also hollow (distance from A to A itself is zero).

Solving the TSP for such a matrix is straightforward, as the hard work has been outsourced to the `{TSP}` package internals.

The optimal route will thus be:

```
## [1] "District 3 Bar"       "Taiko Ramen Bar"      "Palác Akropolis"
## [4] "Belzepub"             "Final Art Music Club"
```

Once we have the optimal route calculated it can be visualized using `{leaflet}`. The sequence of stops needs to be completed (by repeating the first stop after the last) and cast from points to a linestring.



From the visual overview we can see an obvious shortcoming of the “as the crow flies” approach: it completely ignores other constraints except for distance — such as the road network.

Thus while the route shown is “optimal” in the sense that it forms the shortest path joining the five bars selected, it is not one that we could actually follow (unless we were a flying crow).

This shortcoming can be resolved by using an alternative distance matrix as input, while retaining the techniques of {TSP} for the actual route selection. A possible source of more applicable data are routing engines, available to R users via API interfacing packages.

There are several of them available, my personal favorite being the {hereR} package interfacing to the HERE routing engine. I have found HERE to be very reliable and rich in detail. It does require registration, but its free tier is more than adequate for most individual users.

```
## Rows: 25
## Columns: 16
## $ idx_origin      <int> 1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, ~
## $ idx_destination <int> 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 4, 4, 4, ~
## $ route_name      <chr> "Taiko Ramen Bar >> Taiko Ramen Bar", "Belzepub >> Tai~
## $ id              <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ~
## $ rank            <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ~
## $ section         <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ~
## $ departure       <dttm> 2022-10-05 08:50:18, 2022-10-05 08:50:19, 2022-10-05 ~
## $ arrival         <dttm> 2022-10-05 08:50:18, 2022-10-05 08:55:20, 2022-10-05 ~
## $ type            <chr> "vehicle", "vehicle", "vehicle", "vehicle", "vehicle",~
## $ mode            <chr> "car", "car", "car", "car", "car", "car", "car", "car"~
## $ distance        <int> 0, 1512, 1437, 862, 1011, 1476, 0, 489, 1733, 1882, 15~
## $ duration        <int> 0, 301, 261, 136, 169, 269, 0, 75, 298, 331, 275, 78, ~
## $ duration_base   <int> 0, 208, 217, 113, 146, 208, 0, 68, 235, 268, 260, 76, ~
## $ consumption     <dbl> 0.0000, 0.8513, 0.6418, 0.4665, 0.6192, 1.1147, 0.0000~
## $ tolls           <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
## $ geometry        <GEOMETRY [°]> POLYGON ((14.45237 50.08733 ... , LINESTRING (1~
```

The routing results give us several pieces of data:

- the routes as linestring objects in EPSG:4326 (for visualization later on)

- distance of the route (in meters)
- travel time (in seconds) both raw and adjusted for traffic
- petrol consumption

To these I have added three technical columns: indices of start & destination for easier joining of solved TSP results back and the name of the route as string for visualization purposes.

Having a variety of metrics will be helpful in construction of alternative distance matrices.

The first routing distance matrix will be based on route distance; notice that while the matrix is hollow it is not symmetrical. This is not surprising, as routing is not commutative — optimal route from A to B need not be the same as from B to A, due to constraints such as one way roads. Žižkov is a veritable warren of one way streets.

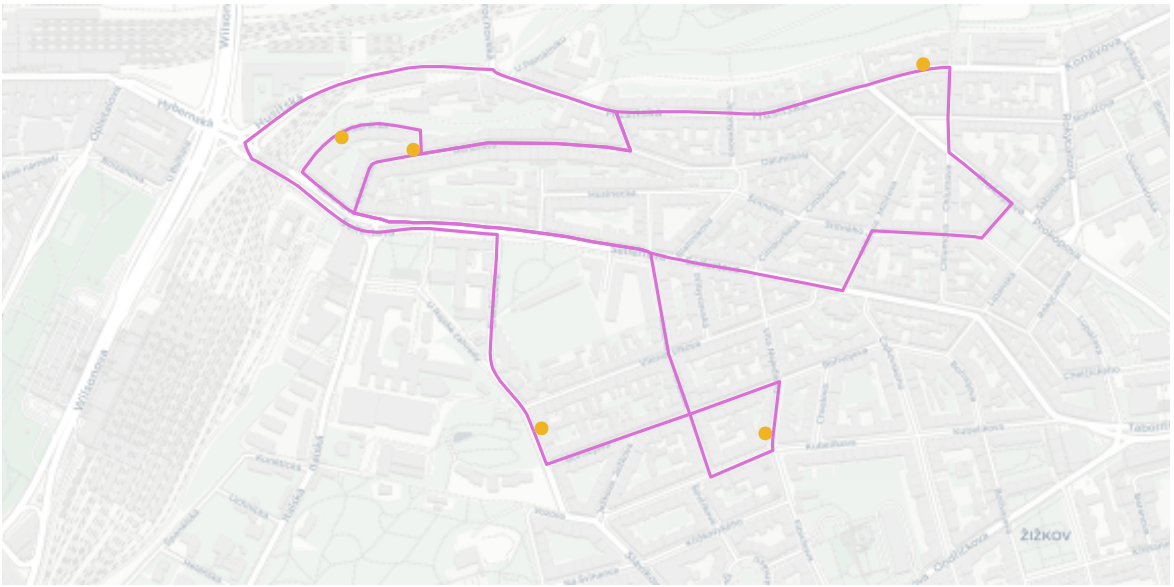
It will need to be declated as asymetrical to `{TSP}` solver; other than that the actual process of solving the matrix will be analogical to the "as the crow flies" matrix.

```
##                       Taiko Ramen Bar Belzepub Palác Akropolis District 3 Bar
## Taiko Ramen Bar                     0     1476            1531           1513
## Belzepub                         1512        0             509            687
## Palác Akropolis                  1437      489               0           1068
## District 3 Bar                    862     1733            2242              0
## Final Art Music Club             1011     1882            2391           2183
##                       Final Art Music Club
## Taiko Ramen Bar                       1551
## Belzepub                               725
## Palác Akropolis                       1106
## District 3 Bar                        2072
## Final Art Music Club                     0

## [1] "Belzepub"             "Palác Akropolis"      "District 3 Bar"
## [4] "Taiko Ramen Bar"      "Final Art Music Club"
```

Once we have solved the TSP and figured the sequence of "cities" to visit it is time to report our results.

For this purpose it is advantageous to prepare a data frame of indices of start and destination, and join it back with the original dataset from HERE API (which contains routes as linestrings).



Since the HERE API is generous in terms of results provided it is not difficult to construct an alternative distance matrix, using a different metric. This could be either trip duration or petrol consumption.

In our specific situation both of these can be expected to be be highly correlated with the plain distance results. All the streets in Žižkov are of very similar type, and the average speed & consumption are unlikely to vary greatly between the routes.

The most significant difference between the distance and time based TSP will be driven by current traffic, which is a factor HERE routing engine considers.

```
##                       Taiko Ramen Bar Belzepub Palác Akropolis District 3 Bar
## Taiko Ramen Bar                     0      269              275            288
## Belzepub                          301        0               78            156
## Palác Akropolis                   261       75                0            211
## District 3 Bar                    136      298              376              0
## Final Art Music Club              169      331              409            374
##                       Final Art Music Club
## Taiko Ramen Bar                        293
## Belzepub                               161
## Palác Akropolis                        216
## District 3 Bar                         346
## Final Art Music Club                     0

## [1] "District 3 Bar"       "Final Art Music Club" "Taiko Ramen Bar"
## [4] "Palác Akropolis"      "Belzepub"
```

Once we have solved the trip duration optimized TSP we again need to report the results; in our use case the output is very similar to the distance based one.

This will not necessarily be the case in other contexts, especially ones with greater variation of road types (city streets vs. highways).

I believe my tongue in cheek example has succeeded in showing two things:

- the ease of applying a standardized solution (the {TSP} package) to a well known and well understood problem (the Travelling Salesman Problem) within the context of R ecosystem

- construction of distance matrices from HERE API routing results, with option to optimize for multiple metrics (such as minimizing the travel distance, travel time and petrol consumption)