Traveling Salesman with Perl, R and Google Maps

2017-07-15 BY GENE

tl;dr: ggplot-nyc & googlemap-nyc & TSP-Map (the Dancer app)

One day I decided to glue-together a couple cool Perl modules and the visualization capabilities of R to generate a map of locations and the computed path of a traveling salesman (TSP) – who in this case is a restaurant critic.

The prerequisite is to have MongoDB installed and loaded with the freely available restaurant test data at https://www.w3resource.com/mongodb-exercises/retaurants.zip.

With that data loaded (with `mongoimport –db test –collection restaurants –drop –file restaurants.json`), the first thing is that the program loads essential perl pragmas and CPAN modules:

```
#!/usr/bin/env perl
use strict;
use warnings;

use YAML;
use MongoDB;
```

```
use Algorithm::TravelingSalesman::BitonicTour;
use Math::Geometry::Planar;
use Statistics::R;
```

Next, handy variables are declared and initialized. This includes getting the MongoDB database configuration with YAML and using it to instantiate a new mongo client object:

```
my $borough = shift || 'Manhattan';  # Also: Queens, Brooklyn, Bronx, etc.
my $grade   = shift || 'A';
my $score   = shift // 70;
my $config  = shift || 'creds.yaml';

my $cfg = YAML::LoadFile($config);

my $client = MongoDB::MongoClient->new(
 host     => $cfg->{mongo}{dbhost},
 db_name  => $cfg->{mongo}{dbname},
 username => $cfg->{mongo}{dbuser},
 password => $cfg->{mongo}{dbpass},
);
```

Here is creds.yaml by the way:

```
mongo:
    dbhost: 'mongodb://localhost'
    dbuser: gene
    dbpass: abc123
    dbname: admin
```

With those set, a database query can be made to select the restaurants to inspect:

```
my $criteria = {
    borough        => $borough,
    'grades.grade' => $grade,
    'grades.score' => { '$gt' => 0 + $score },
};

my $collection = $client->ns('test.restaurants');
my $docs       = $collection->find($criteria);
```

The next set of steps is the traveling-salesman logic. The selected restaurant address coordinates are added, the TSP algorithm is solved and the resulting polygon centroid ("center of gravity") of the points is found:

```
my $tsp = Algorithm::TravelingSalesman::BitonicTour->new;

my %coord_name;  # Index of coordinates to restaurant names

while ( my $doc = $docs->next) {
    my @point = @{ $doc->{address}{coord} };
    $coord_name{ join ',', @point } = $doc->{name};
    $tsp->add_point(@point);
}

my ( undef, @coords ) = $tsp->solve;

my $i = 0;  # Counter

print "Optimal path:\n";
for my $coord (@coords) {
    my $key = join ',', @$coord;
    printf "\t%d. %s [%s]\n", ++$i, $coord_name{$key}, $key;
}

my $polygon = Math::Geometry::Planar->new;
$polygon->points( [ @coords[ 1 .. $#coords ] ] );
my $centroid = $polygon->centroid;
```

Now the coordinates are mapped with R, to a PNG file, with the excellent ggmap package:

```
my $R = Statistics::R->new();

$R->run( 'library(ggmap)' );

my $file = "$0.png";
$R->run( "png('$file')" );
```

Next, some R variables are declared (i.e. latitude/longitude vectors and the data.frame to plot):

```
$R->set( 'X', [ map { $_->[0] } @coords ] );
$R->set( 'Y', [ map { $_->[1] } @coords ] );
$R->run( 'df <- data.frame(X, Y)' );
$R->run( "names(df) <- c('lon','lat')" );
```

Now, the map is generated based on the polygon centroid computed earlier:

```
$R->run( "center <- c( $centroid->[0], $centroid->[1] )" );
$R->run( "centroid <- get_googlemap( center = center, zoom = 12 )" );

my $gg_cmd = 'ggmap( centroid, aes(lon, lat) )';
$gg_cmd .= ' + geom_point( data = df, colour = "darkred" )';
$gg_cmd .= ' + geom_path( data = df, color = "darkgray", size = 0.5 )';
$R->run($gg_cmd);
```

Finally, the R graphics device is closed and the perl-R session is stopped:

```
$R->run( 'dev.off()' );

$R->stop();
```

Here is the output of the program for Manhattan grade-A restaurants with a rating of 70 or higher:

```
Optimal path:
        1. Brothers Fish Market [-73.9435296,40.8361392]
        2. Murals On 54/Randolphs'S [-73.9782725,40.7624022]
        3. Mars Cafe [-73.9850196,40.7524629]
        4. Midtown Buffet [-73.987977,40.755195]
        5. B.B. Kings [-73.9889479,40.7568894]
        6. Cafe R [-73.9897851,40.7487912]
        7. Concrete Restaurant [-73.9934047,40.7544014]
        8. West 79Th Street Boat Basin Cafe [-74.0163793,40.7167671]
        9. Wonton Noodle Garden [-73.9983631,40.7158265]
        10. Live Bait Bar & Restaurant [-73.9883909,40.740735]
        11. Gandhi [-73.9864626,40.7266739]
        12. Bella Napoli [-73.984758,40.7457939]
        13. East Japanese Restaurant [-73.981843,40.741207]
        14. Two Boots Grand Central [-73.9772294,40.7527262]
        15. Everyday Gourmet Deli [-73.976478,40.7504097]
        16. Bistro Caterers [-73.9747277,40.7536114]
        17. La Trattoria [-73.970671,40.7515735]
```

```
18. Baluchi'S Indian Food [-73.94981,40.780043]
19. Lexington Restaurant [-73.941892,40.7982236]
20. Brothers Fish Market [-73.9435296,40.8361392]
```

And here is the map that is produced:

(I can't seem to figure out how to make the map larger, yet. And adding get_googlemap(…, size = c(x,y)) is not the answer. Grrrr)

UPDATE: I have found a much better way to render a map! (Updated code here.)

This code throws out R in favor of the perl module HTML::GoogleMaps::V3. Instead of generating a small PNG, the new code makes an HTML file with a handy Google Map.

Ok. After the code above, up to the TSP algorithm solving, we find the path centroid as before:

```
...
my ( undef, @coords ) = $tsp->solve;

# Find the path polygon centroid
my $polygon = Math::Geometry::Planar->new;
$polygon->points( [ @coords[ 1 .. $#coords ] ] );
```

```
my $centroid = $polygon->centroid;
```

Next comes the beginning of the updated logic:

```
my $map = HTML::GoogleMaps::V3->new( height => 800, width => 800 );
$map->zoom(12);
$map->center($centroid);
```

Then we add markers to this map, for each optimal coordinate in order. Also the path is added to the map:

```
for my $coord (@coords) {
    my $point = join ',', $coord->[1], $coord->[0];  # Google maps wants la
    my $key   = join ',', @$coord;

    $coord_name{$key} =~ s/'/`/g;  # Single quotes conflict with the marker

    $map->add_marker(
        point => $coord,
        html  => qq|<div id="content"><h3 id="firstHeading" class="firstHea
    );
}

# Add a path for all but the last (redundant) coordinate.
$map->add_polyline( points => [ @coords[ 0 .. $#coords - 1 ] ] );
```

Finally we print-out the resulting HTML to save to a file:

```
my ($head, $map_div) = $map->onload_render;

print qq|<html><head><title>Test</title>$head</head>|;
print qq|<body onload="html_googlemaps_initialize()">$map_div</body></html>
```

This is run as:

```
$ perl googlemap-nyc > googlemap.html

$ open googlemap.html
```

Here is a screenshot of the resulting map:

Note: Although not reflected here, the updated code includes output of the optimal path, directions to each waypoint, and the driving directions for the whole route! Here is a screenshot of the route driving directions map:

FILED UNDER: DATA, SOFTWARE
TAGGED WITH: GOOGLE MAPS, MAPPING, PERL, R, TRAVELING SALESMAN PROBLEM, VISUALIZATION

Epistemologist-at-large

^ Top