A Routable Path in Postgres

--Research on GIS with Postgres, a Relational Database
By Tiffany Yang

In this tutorial, I will demonstrate how to build the shortest path between two locations using PostgreSQL a relationship database. The steps will include GIS tools utilities and data manipulation with SQL. By following the steps below, a routing path will be created. The route also considers the direction of streets when calculating the shortest distance between two locations.

Requirement

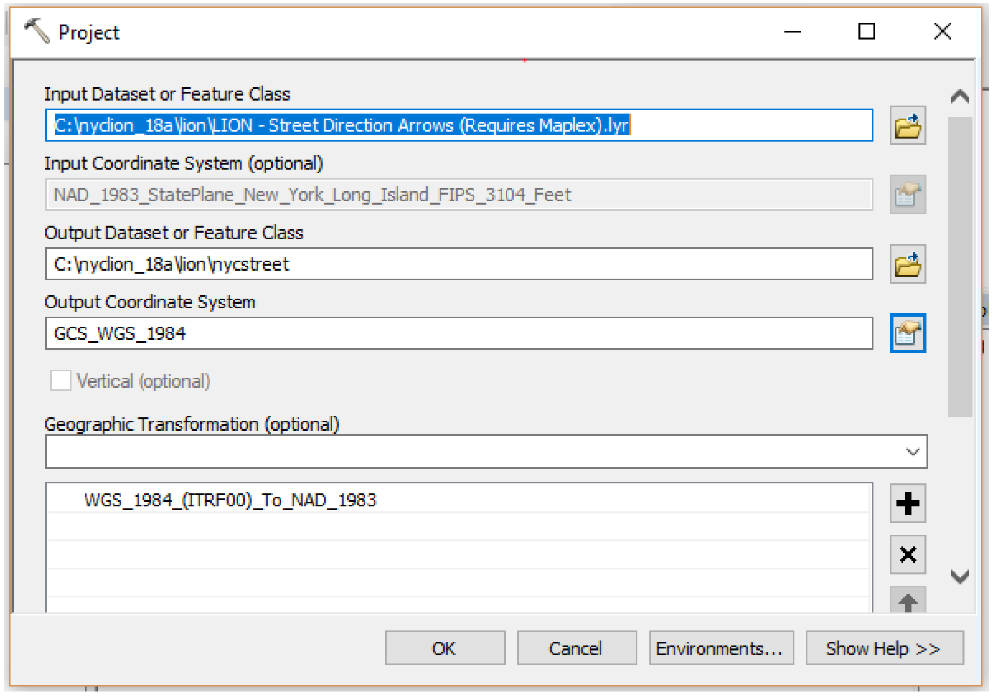
Before we start, we will need the following software installed:

- **Postgres with PostGIS and pgAdmin**
- **PostGIS Shapefile Import/ Export Manger**
- **ArcGIS/ QGIS**

Data Acquisition

1. **Download Data**
   For the NYC street data, you can download from NYC Planning.
   For other cities data map, you may go to OpenStreetMap Data Extracts

2. **Setup shapefile**
   Transform the LION file to the Postgres supported shapefile.
   - **Add Data to Your Map**: click the **Add Data** button. On the Add Data window, open the **nyclion_18a folder →lion → LION - Street Direction Arrows (Requires Maplex)**, then select **Add**.
   - **Change Coordinate System**: open **ArcToolbox** (on the second row of toolbar), select **Data Management Tools → Projections and Transformations → Project**.
     a. **Under Input Data Set or Feature Class**, choose the Map Layer for which you want to change the Projection.
     b. **Under Output Data Set or Feature Class**, check to make sure that the file is being saved in the same folder as the rest of your Map and Data Layers.
     c. **Under Output Coordinate System**, choose **GCS_WGS_1984**



Input Dataset or Feature Class: C:\nyclion_18a\lion\LION - Street Direction Arrows (Requires Maplex).lyr

Input Coordinate System (optional): NAD_1983_StatePlane_New_York_Long_Island_FIPS_3104_Feet

Output Dataset or Feature Class: C:\nyclion_18a\lion\nycstreet

Output Coordinate System: GCS_WGS_1984

Vertical (optional)

Geographic Transformation (optional): WGS_1984_(ITRF00)_To_NAD_1983

3. **Upload shapefile to pgAdmin**
   - After the project is finished loading, you will see the nycstreet.shp file in the same folder. Then, you can use that file to upload to the database.
   - The simplest way to upload shapefile to pgAdmin is using PostGIS Shapefile Import/ Export Manger tool.
     Please see tutorial here.

Data manipulation

1. **Create Extensions**
   Before we start writing queries, let’s enable PostGIS and pgRouting extensions in a Postgres database.

```
CREATE EXTENSION postgis;

CREATE EXTENSION pgrouting;
```

2. **Building a New Table & Changing datatype**

```
select gid, street,trafdir,rboro,rw_type,shape_leng, geom into nyc from nycstreet;


alter table nyc rename gid to id;

alter table nyc alter column rw_type type varchar;

alter table nyc alter column trafdir type varchar;

alter table nyc alter column shape_leng type integer;
```

3. **Data Clean up**

```
delete from nyc where rboro='0';            /*Delete the edge streets data of Brooklyn*/

delete from nyc where rboro='2';            /* Delete the Bronx dataset*/

delete from nyc where rboro='3';            /* Delete the Brooklyn dataset */

delete from nyc where rboro='4';            /* Delete the Queens dataset*/

delete from nyc where rboro='5';            /* Delete the Staten Island dataset*/


delete from nyc where rw_type= '5';                  /* Delete the Boardwalk dataset*/

delete from nyc where rw_type= '6';                  /* Delete the Path/ Trail dataset*/

delete from nyc where rw_type='7';                   /* Delete the Step Street dataset*/

delete from nyc where rw_type='11';                  /* Delete the Unknown dataset*/

delete from nyc where rw_type='12';                  /* Delete the Non- Physical Street Segment dataset*/

delete from nyc where rw_type='14';                  /* Delete the Ferry Route dataset*/
```

Cost – The Direction of Streets

Some streets are bi-directional, but some streets are only one-way. How to tell pgr_digkstra function which one is an one-way street? When the cost is positive 1, it indicates that street is from source to target. But when the cost is negative 1, that street is from target to source. The pgr_digkstra function only takes the cost data which is larger than 0 into consideration. That is how the algorithm finds the corrected direction from source to target.

```
alter table nyc add column cost integer;

alter table nyc add column reverse_cost integer;


UPDATE nyc SET trafdir = CASE WHEN (trafdir='T') THEN 'B'

                              WHEN (trafdir='W') THEN 'FT'

                              WHEN (trafdir='A') THEN 'TF'

                              ELSE '' END;

UPDATE nyc SET cost = CASE WHEN (trafdir='B') THEN 1

                              WHEN (trafdir='FT') THEN 1

                              WHEN (trafdir='TF') THEN -1

                              ELSE '0' END;

UPDATE nyc SET reverse_cost = CASE WHEN (trafdir='B') THEN 1

                              WHEN (trafdir='FT') THEN -1

                              WHEN (trafdir='TF') THEN 1

                              ELSE '0' END;
```

Network Topology

Converting the actual street network to a graph, we need to add the **source** and **target** columns which indicate the head and tail of the street.

```
ALTER TABLE nyc ADD COLUMN source integer;

ALTER TABLE nyc ADD COLUMN target integer;

CREATE INDEX nyc_source_idx ON nyc (source);

CREATE INDEX nyc_target_idx ON nyc (target);
```

Then use pgr_createTopology function which builds a network topology based on the geometry information. It analyzes road geometry and automatically assigns node ids to the source and target columns.

```
SELECT pgr_createTopology('nyc', 0.00001, 'geom', 'id');
```

Diagnose on our data table

To ensure our network topology is working properly, we need to run few topology functions to check. It will return **OK**, if the street network is good.

```
SELECT  pgr_createVerticesTable('nyc','geom','source','target');

SELECT  pgr_analyzeGraph('nyc',0.001,'geom','id','source','target');

SELECT pgr_analyzeOneway('nyc',

ARRAY['', 'B', 'TF'],

ARRAY['', 'B', 'FT'],

ARRAY['', 'B', 'FT'],

ARRAY['', 'B', 'TF'],

oneway:='trafdir');
```

Navigation on Database

After the previous steps, the data is available for use. To return a route we will use the prg_digkstra function. It will automatically calculate the shortest path for us. Please be aware that the arguments of the function are the longitude and latitude of a location.

Here is the example from **BMCC Fiterman Building** to **CUNY Welcome center**.

```
SELECT seq, path_seq, node, edge, st_length(geom) AS cost, agg_cost, geom
FROM pgr_dijkstra(
   'SELECT id, source, target, cost, reverse_cost FROM nyc',
   (SELECT source
                         FROM nyc
                         ORDER BY ST_Distance(
                         nyc.geom,
                                      ST_SetSRID(ST_MakePoint(-74.011287, 40.714294), 4326),
                                      true
            ) ASC
            LIMIT 1),
   (SELECT source
                         FROM nyc
                         ORDER BY ST_Distance(
                         nyc.geom,
                                      ST_SetSRID(ST_MakePoint(-73.973682,40.751486), 4326),
                                      true
            ) ASC
            LIMIT 1)
) as pt
JOIN nyc rd ON pt.edge = rd.id;
```

Stored Procedure

Next, we are going to create a stored Procedure function, so the path delivers to the front-end.

```
 CREATE OR REPLACE FUNCTION nyc_shortest_path(
            IN tbl varchar,
            IN x1 double precision,
            IN y1 double precision,
            IN x2 double precision,
            IN y2 double precision,
            OUT seq integer,
            OUT path_seq integer,
            OUT node bigint,
            OUT edge bigint,
            OUT cost double precision,
            OUT agg_cost double precision,
            OUT geom geometry,
            OUT heading integer
      )
      RETURNS SETOF record AS
$BODY$
DECLARE
      sql     text;
      rec     record;
      source  integer;
      target  integer;
      point   integer;
```

```
BEGIN
                        -- Find nearest node
      EXECUTE 'SELECT source AS id FROM nyc
                  ORDER BY geom <->  ST_GeomFromText(''POINT(' || y1 || ' ' || x1 || ')'') LIMIT 1' INTO rec;
      source := rec.id;

      EXECUTE 'SELECT source AS id FROM nyc
                  ORDER BY geom <->  ST_GeomFromText(''POINT(' || y2 || ' ' || x2 || ')'') LIMIT 1' INTO rec;
      target := rec.id;

             -- Shortest path query (TODO: limit extent by BBOX)
      seq     := 0;
       sql     := 'SELECT * FROM ' ||
                  'pgr_dijkstra(''SELECT id, source, target, cost, reverse_cost FROM '
                              || quote_ident(tbl) || ''', '
                              || source || ', '
                              || target || ') as pt'
                                                  ' JOIN nyc rd on pt.edge=rd.id ORDER BY seq';
                        -- Remember start point
      point := source;

      FOR rec IN EXECUTE sql
      LOOP
            -- Flip geometry (if required)
           IF ( point != rec.source ) THEN
                  rec.geom := rec.geom;
                  point := rec.source;
            ELSE
                  point := rec.target;
            END IF;
```

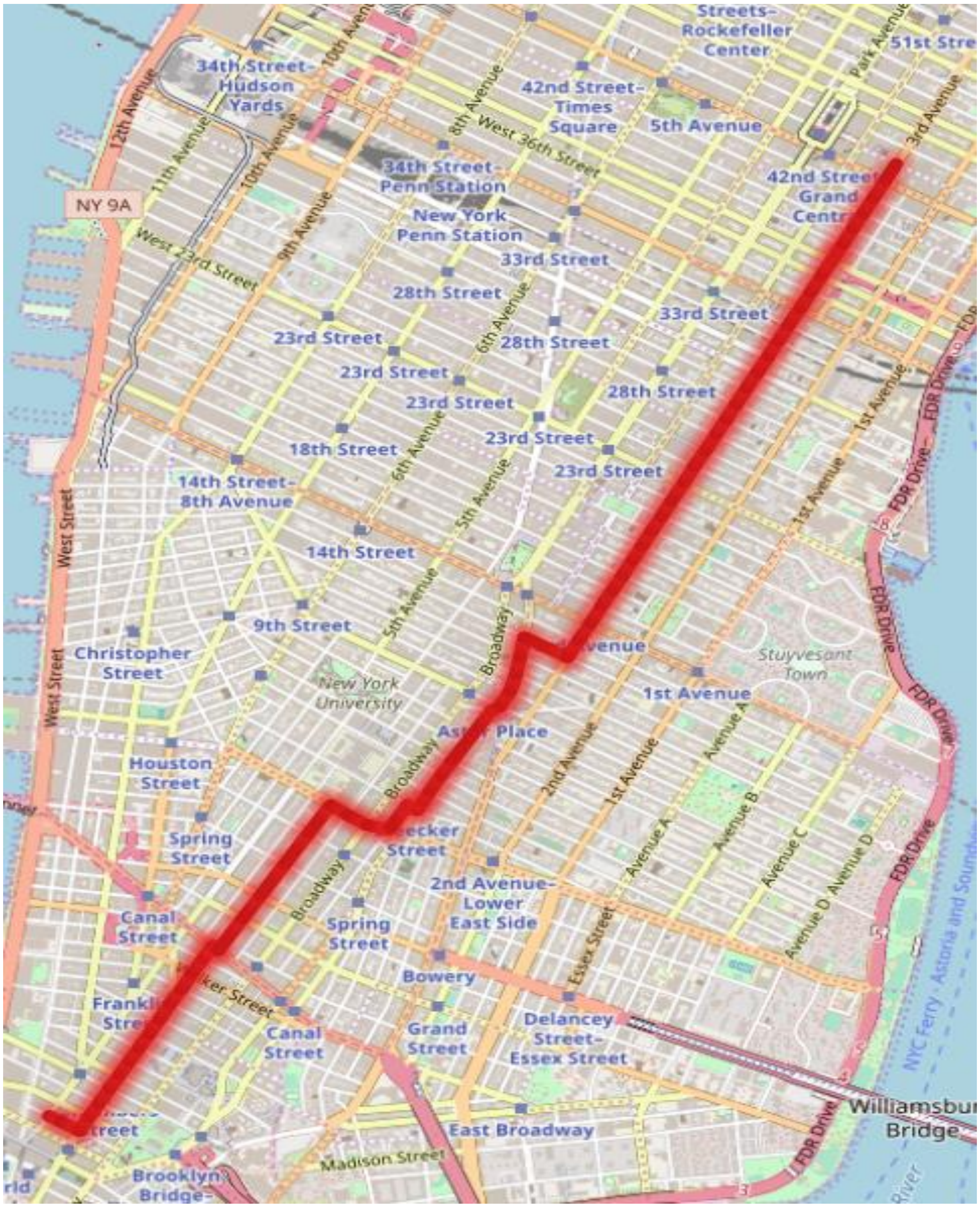
```
  -- Calculate heading (simplified)
            EXECUTE 'SELECT degrees( ST_Azimuth(
                      ST_StartPoint(st_lineMerge(''' || rec.geom::text || '''::geometry)),
                      ST_EndPoint(st_lineMerge(''' || rec.geom::text || '''::geometry) ) ))'
                 INTO heading;
            --Return record
                                seq     := seq + 1;
            path_seq     := rec.path_seq;
            node   := rec.node;
                                          edge       := rec.edge;
            cost    := rec.cost;
                                          agg_cost:=rec.agg_cost;
            geom    := rec.geom;
            RETURN NEXT;
       END LOOP;
       RETURN;
END;
$BODY$
LANGUAGE 'plpgsql' VOLATILE STRICT;
```

Example query

```
select * from nyc_shortest_path('nyc', 40.714294, -74.011287, 40.751486, -73.973682)
```

Shortest Path Showing in QGIS

When QGIS connects to PostGIS, this is the returned table shown in the New York City map.



Conclusion

From this tutorial, I hope you learn how to use PostGIS and PostgreSQL to create routing paths with the geometry data map. With the tools like PostGIS, pgRouting, and the digkstra function, we can easily create an optimized routing. There would be so much fun to create your own routing path.

Please feel free to contact me if you have any questions. Any feedback is valuable to me!