Plus courts chemins avec pgRouting et des donnéess OpenStreetMap

Maxence Laurent
HEIG-VD :: IICT/SYSIN

October 23, 2009

1 Introduction

Le but est d'importer des donées OpenStreetMap (OSM) dans une base de données PostGIS afin de calculer des plus cours chemins.

2 Création de la base de données OSM-PgRouting

2.1 Obtention des données

Tout d'abord il faut obtenir un fichier *.osm qui contient les données sur lesquelles on veut travailler. On peut obtenir un fichier de ce type de plusieurs manière :

- Depuis le site http://www.openstreetmap.org, onglet *export*. Attention, la taille de la zone exportable est limitée à une viewbox de moins de 0.25° de côté et un nombre de nœuds $< 50'000$
- Des zones pré-extraintes chaques jours sont téléchargeable sur :
  - http://downloads.cloudmade.com/
  - http://download.geofabrik.de/osm/
  - D'autres site ... Google is your friend
- À l'aide d'un programe comme jOSM

2.2 Création de la base de données

Il faut créer un base de données PostGIS[1] et ensuite y ajouter les fonctionnalités pgRouting.

PgRouting est un ensemble de function PLPGSQL que l'on peut ajouter dans une base de données. Pour cela, il faut suivre ces quelques instructions :

1. Télécharger les sources sur http://pgrouting.postlbs.org/
2. Dans un terminal :
3. $ *tar xvfz pgrouting-X.YY.tgz*

[1] Ajouter le language PLPGSQL et les scripts lwpostgis.sql et spatial_ref_sys.sql

4. $ *cd pgrouting*
5. $ *cmake .*
6. $ *make*
7. $ *sudo cp lib/librouting.so /usr/lib*
8. $ *psql -d YourPostGISDB -u ... -f core/sql/routing_core.sql*
9. $ *psql -d YourPostGISDB -u ... -f core/sql/routing_core_wrappers.sql*
10. $ *psql -d YourPostGISDB -u ... -f core/sql/routing_topology.sql*

Voilà, la base de données possède maintenant les fonctions nécessaires pour faire du routing.

3 Importation des données dans la base

Le programme osm2pgrouting permet d'importer des données OSM directement dans une base de données PostGIS, dans un format que pgRouting peut comprendre.

Pour utiliser osm2pgrouting, suivre ces quelques étapes :

1. $ *svn checkout http://pgrouting.postlbs.org/svn/pgrouting/tools/osm2pgrouting/trunk osm2pgrouting*
2. $ *cd osm2pgrouting*
3. $ *make*

Voilà, le programme osm2pgrouting se trouve maintenant dans le répertoire courant. Il peut être interessant de le mettre dans /usr/bin, ou, pour faire plus propre, dans ~/usr/bin et de vérifier que ~/usr/bin est bien dans le PATH [2].

La commande *osm2pgrouting* prend les paramètres suivants :

- Obligatoires

  -file < $file$ > – fichier.osm

  -conf < $conf$ > – fichier de configuration (fournir avec l'archie osm2pgrouting)

  -dbname < $dbname$ > – nom de la base où mettre les données

  -user < $usename$ > – nom d'utilisateur (accès en écriture à la base)

- Optionnels

  -host < *host* > – hostname du serveur postgresql (*localhost* par défaut)

  -port < *port* > – 5432 par défaut

  -passwd < *password* > – Mot de pas pour l'utilisateur

  -clean – supprime les tâbles si elles existent déjà

Par exemple, la commande *"osm2pgrouting -file switzterland.osm -conf mapconfig.xml -dbname osm_swiss -user postgres"* va importer les données OSM contenue dans switzerland.osm dans la base osm_swiss avec l'utilisateur *postgres*.

[2] *$ env | grep "^PATH" | grep 'echo ~'/usr/bin* , si une ligne le path est affichée, alors c'est bon, sinon, modifiez votre fichier /.bashrc et ajouter-y la ligne "export PATH=~/usr/bin/", puis resourcer ce fichier avec la commande

4 Routing in Action

L'importation des données OSM crée plusieurs tables, celle qui nous interessent sont:

- classes : représente les différents type de routes
- vertices_tmp : représente les carrefours
- ways : représente les routes

Pour calculer des plus court chemins, plusieurs méthodes sont mises à disposition par pgRouting. Elle sont toute décrite à la page :
http://pgrouting.postlbs.org/#LearnaboutpgRouting.

5 Exemple d'utilisation : Arbre de plus courts chemins

5.1 Récupérer un sommet à partir d'une lat/long

La requête SQL suivante permet de retrouver le carrefour le plus proche d'un coordonnées GPS:

```
SELECT id, the_geom, distance(the_geom,
  GeometryFromText( 'POINT(latitude longitude)', 4326)) AS dist
FROM vertices_tmp
WHERE the_geom &&
      expand(GeometryFromText( 'POINT(latitude longitude)',4326),
            box_size)
ORDER BY dist
LIMIT 1
```

5.2 Calcul d'un plus court chemin avec A*

```
SELECT *
FROM shortest_path_astar(
        'SELECT gid AS id,
                source::int4,
                target::int4,
                time::double precision AS cost,
                time_reverse as reverse_cost,
                x1, y1,
                x2, y2
         FROM ways',
     id_noeud_depart, id_noeud_arrivée,
     true, true)
```

Les paramètres de shortest_path_astar sont :

1. Une requête sql qui revoie les routes : id, noeud source, noeud destination, coordonnées des ces deux points, coût source→destination et coût inverse destination→source

- le coût (et le coût inverse) peut varier selon ce que l'on souhaite calculer (à pied, à vélo, en voiture)
- si le coût (ou le coût inverse) est < 0 alors cela signifie que le chemin ne peut pas être parcouru dans ce sens

2. les id des noeuds de départ et d'arrivé, que l'on obtient avec la requête présenté ci-dessus
3. L'avant dernier bouléen indique si le graphe est orienté ou non
4. Le dernier boolean indique s'il faut utiliser les coûts inverses

Cette requête retourne un ensemble d'enregistrement qui décrivent les routes qui sont utilisées par le plus court chemin. La première colonne indique le noeud d'ou par la route. La seconde indique la routes à prendre depuis ce noeud. La dernière indique le coût associé à cette routes.

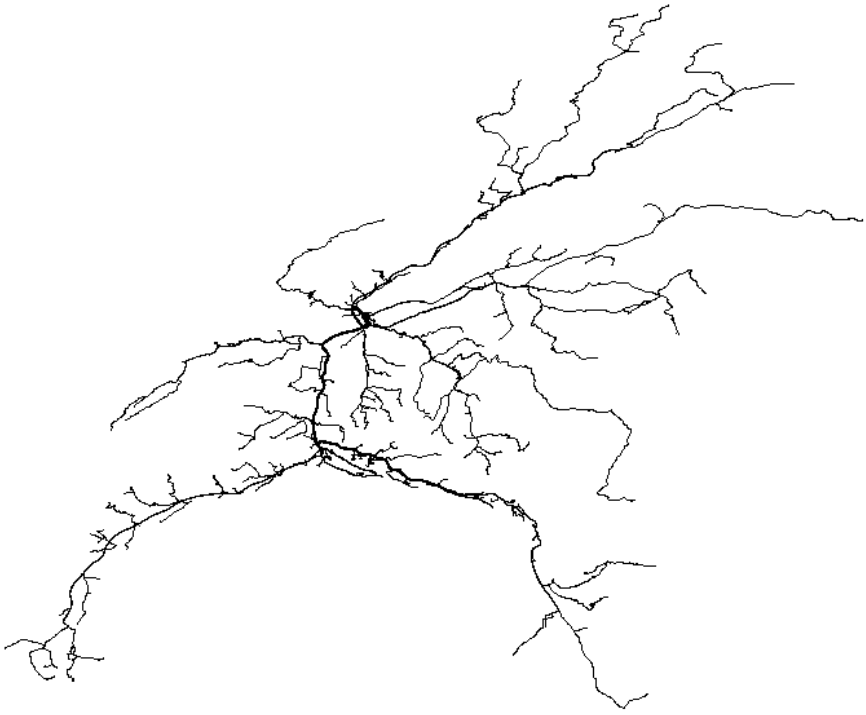
5.3 Construction de l'arbre des plus courts chemin

Algorithm 1 ShortestPathTree

Requiert: $F \Leftarrow \{(lat, long, size)\}$ Ensemble des coordonnées des points de départs

Résultat: Une couches de données géographique qui représente l'arbre des plus court chemins

$edge \Leftarrow \emptyset$
$toNodeId \Leftarrow$ Obtenir l'id du noeud d'arrivée depuis ses coordonnées lat/long
tant que $|F| > 0$ **faire**
  Extraire $(lat, long, size)$ de F
  $fromNodeId \Leftarrow$ Obtenir l'id du noeud de départ depuid $(lat, long)$
  $SP \Leftarrow$ obternir le plus court chemin entre $fromNodeId$ et $toNodeId$
  tant que $|SP| > 0$ **faire**
    extraire $(nodeId, wayId, cost)$ de SP
    $flowSize[wayId] += size$
  fin tant que
fin tant que
retourner une couche géographique $flow$ comprenant toutes les routes contenues dans $flowSize$ ainsi que la taille du flux qui passe par ces routes



6 Conclusion

Simplement une liste des truc bizarre que j'ai rencontré :

- *osm2pgrouting* ne fonctionne pas avec postgis 1.3.1 (buggy version)
- L'importation des données créé deux tables qui contienent les sommets du graphe (*nodes* et *vertices_tmp*)
- Les identifiants OSM sont perdus. Ce n'est pas pratique pour mettre à jour la base de données sans tout ré-importer
- Certaine méthode shortest_path plantent selon leur configuration
  - Dijkstra avec un réseau dirigé et des coûts inverses
  - A* ne trouve pas la première routes si le sommet de départ est la target et pas la source de la routes
  - Il y en a sûrement d'autres des petites surprise comme ça
  - A première vue, pgRoutuing n'est pas robuste lorsque des contraintes sont ajouté à la recherche