Travaux pratiques - Routage FOSS4G avec pgRouting, le réseau routier dOpenStreetMap et GeoExt

Version 2

Daniel Kastl, Frédéric Junod

19 October 2017

Table des matières

CHAPITRE 1

Introduction

Résumé

pgRouting ajoute les fonctionalités de routage à PostGIS. Ces travaux pratiques d'introduction vont vous présenter comment. Il présente un exemple pratique de comment utiliser pgRouting avec les données du réseau routier d'OpenStreetMap. Il explique les étapes permettant la préparation des données, effectuer des requêtes de routage, l'assignation des coûts et l'utilisation de GeoExt pour présenter les chamins depuis une application de web-mapping.

La navigration dans un réseau de routes nécessite des algorithmes de routage complex qui supportent la restriction des virages utilisables et même des attributs dépendant du temps. pgRouting est une librairie OpenSource qui peut être étendue qui fournit une grande variété d'outils pour la recherche de plus courts chemins comme une extension de PortgreSQL et de PostGIS. Ces travaux pratiques vont expliquer la recherche de plus courts chemin avec pgRouting dans un réseau routier réel et comment la structure des données est importante pour obtenir des résultats plus rapidement. Vous apprendrez aussi les difficultés et le limitations de pgRouting dans le cadre d'application SIG.

Pour fournir un exemple pratique, ces travaux pratiques utiliseront les données OpenStreetMap de Denver. Vous apprendrez comment convertir les données dans le format requis et comment calibrer les données avec les attributs de "coût". De plus, nuos allons expliquer la différence entre l'agorithme principal de routage "Dijkstra", "A-Star" et "Shooting-Star". À la fin de ces travaux ratiques vous aurez acquis une bonne compréhension de comment utiliser pgRouting et comment préparer vos données de réseau.

Pour apprendre comment utiliser la liste des résultats afin de les affichées sur une carte, nous allons contruire une interface graphique de base avec GeoExt. Nous avons écouté les remarques qui avaient été faites l'an dernier et voulons vous guider pour les étapes de bases nécessaires à la mise en ligne d'une application de routage. Notre but est de faire cei de la manières la plus simple possible, et de montrer qu'il n'est pas difficile d'intégrer cela avec les autres outils OpenSource FOSS4G. Pour cette raison, nuos avons selectionné GeoExt, qui est une librairies JavaScript fournissant les éléments de base pour créer des application de web-mappping basées sur OpenLayers et Ext.

Note :
- Niveau requis : intermédiaire
- Connaissances requises : SQL (PostgreSQL, PostGIS), Javascript, HTML
- Matériel : ces travaux pratiques utiliseront le LiveDVD de l'OSGeo.

Presenter

- *Daniel Kastl* est le fondateur et le directeur de l'entreprise Georepublic et travail en Alemagne et au Japon. Il modère et pormeut la communauté pgRouting et en assure le développement depuis 5 ans, il est un actif contributeur de la

communauté OSM au Japon.
– *Frédéric Junod* travail au bureau Suisse de l'entreprise Camptocamp depuis environ 5 ans. Il est développeur actif de nombreux projets OpenSource depuis le navigateur (GeoExt, OpenLayers) jusqu'au monde serveur (MapFish, Shapely, TileCache) il est membre du commité de pilotage de pgRouting.

Daniel et Frédéric sont les auteurs des travaxu pratiques pgRouting précédents, qui ont eut lieu au FOSS4G au Canada, en Afrique du Sud, en Espagne et dans des conférences locales au Japon.

License

Support

Camptocamp

Georepublic

CHAPITRE 2

À propos

Ces travaux pratiques utilisent de nombreux outils FOSS4G, beaucoup plus que le titre ne le laisse supposer. Ainsi, de nombreux logiciels FOSS4G sont liés au autres project OpenSource et ce serait aller trop loin de tous les lister. Il y a néanmoins 4 projets OpenSource que ces travaux pratiues présenterons :

2.1 pgRouting

pgRouting est un extension de PostGIS et ajoute les fonctionalités de routing au couple PostGIS/PostgreSQL. pgRouting est un développement antérieur à pgDijkstra (par Camptocamp SA). Il a été étendu par l'entreprise Orkney Inc. et est actuellement développé et maintenu par Georepublic.

pgRouting fournit les fonctions pour :
- Plus court chemin Dikstra : algorithme de routage sans heusitique
- Plus court chemin A-Étoile : routage pour grand un ensemble de données (avec heuristiques)
- Plus court chemin Shooting-Star : routage prenant en compte le sens giratoir (avec heuristiques)

– Problème du voyageur de commerce (TSP)
– Distance de pilotage (Isolines)

De nombreux nouveaux algorithmes vont être ajoutés dans un futur proche :
– Dial-a-Ride-Problem solver (DARP)
– Support du routage multimodal
– Algortihme de recherche de plus court chemin dépendant du temps / dynamique
– A-Étoile utilisant deux direction
– All-Pair shortest path algorithm

Les avantages de l'approche base de données pour les algorithmes de routage sont :
– L'accessibilité par nombreux clients au travers de JDBC, ODBC, our directement en tuililsant Pl/pgSQL. Les clients peuvent être des ordinateurs personnels ou des périfériques mobiles.
– Utilise PostGIS pour le format des données géographiques, qui utilise le Format Text Bien Connu (WKT) de l'OGC et le Format Binaire Bien Connu (WKB).
– Les logiciels OpenSource comme qGIS et uDig peuvent modifier les données / les attributs.
– Les modifications apportées aux données peuvent être intégrées au moteur de routage instantanément. Il n'y pas de besoin de pré-calcul.
– Le paramètre de "coût" peut être calculé dynamiquement à l'aide du SQL et ces valeurs peuvent provenir de différets attributsou tables.

pgRouting est disponible sous licence GPLv2.

Le site web officiel de pgRouting : http ://www.pgrouting.org

2.2 OpenStreetMap

"OpenStreetMap a pour objectif de créer et fournir des informations géographiques libres telles que des plans de rue à toute personne le désirant. Le projet fut démarré car la plupart des cartes qui semblent libres ont en fait des restrictions d'utilisation légales ou techniques, empêchant de les utiliser de manière créative, productive ou tout simplement selon vos souhaits." (Source : http ://wiki.openstreetmap.org/wiki/FR :Portal :Press)

OpenStreetMap est une source de données parfaite pour être utilisées avec pgRouting, car elles sont librement disponibles et n'ont pas de restrictions techniques en terme d'utilisation. La disponibilité des données varie d'un pays à un autre, mais la couverture mondiale s'améliore de jour en jour.

OpenStreetMap utilise une structuration de données topologiques :
– Les noeuds (nodes) sont des points avec une position géographique.
– Les chemins (ways) sont des liste de noeuds, représentant une polyligne ou un polygone.
– Les relations (relations) sont des groupes de noeud, de chemin et d'autre relations auquelles on peut affecter certaines propriétés.
– Les propriétés (tags) peuvent être appliqués à des noeuds, des chemins ou des relation et consistent en un couple nom=valeur.

Site web d'OpenStreetMap : http ://www.openstreetmap.org

2.3 osm2pgrouting

osm2pgrouting est un outils en ligne de commande qui rend simple l'importation de données OpenStreetMap dans une base de données pgRouting. Il contruit le réseau routier automatiquement et crée les tables pour les types de données et les classes de routes. osm2pgrouting a été écrit initialement par Daniel Wendt et est maintenant disponible sur le site du projet pgRouting.

osm2pgrouting est disponible sous licence GPLv2.

Site web du projet : http ://www.pgrouting.org/docs/tools/osm2pgrouting.html

2.4 GeoExt

GeoExt est une librairie JavaScript destinée au applications web avancées. GeoExt rassemble les connaissances géospatiales de OpenLayers avec une interface utilisateurs issue de Ext JS pour aider à construire un application web ressemblant à une application bureautique.

GeoExt est disponible sous licence BSD et est maintenu par une communauté grandissante d'individus, d'ogranisations et d'entreprises.

Le site web de GeoExt : http ://www.geoext.org

CHAPITRE 3

Installation et prérequis

Pour ces travaux pratiques, vous aurez besoin de :
– un serveur web comme Apache avec le support PHP d'activé (et le module PostgreSQL de PHP)
– De préférence un système d'exploitation de type Linux comme Ubuntu
– Un éditeur de texte comme Gedit
– Une connexion internet
Tout les outils requis sont disponibles sur l'OSGeo LiveDVD, donc les références qui suivent représentent un rapide résumé de comment l'installer sur votre propre machine tournant sur Ubuntu 10.04 ou suppérieur.

3.1 pgRouting

L'installation de pgRouting sur Ubuntu est devenue extrêmement simple du fait de la disponibilité d'un paquet binaire dans le dépot Launchpad : .

Tout ce que vous avez à faire maintenant c'est d'ouvrir une fenêtre de terminal et de lancer :

```
# Ajouter le dépôt launchpad
sudo add-apt-repository ppa:georepublic/pgrouting
sudo apt-get update

# Installer les paquets pgRouting
sudo apt-get install gaul-devel \
        postgresql-8.4-pgrouting \
        postgresql-8.4-pgrouting-dd \
        postgresql-8.4-pgrouting-tsp

# Installer le paquet osm2pgrouting
sudo apt-get install osm2pgrouting

# Installé le contenu des travaux pratiques (optionel)
sudo apt-get install pgrouting-workshop
```

Cela installera aussi tout les paquets dépendants comme PostgreSQL et PostGIS s'ils ne sont pas déjà installés.

Note :
– Les paquets "multiverse" doivent être disponibles comme des sources de logiciels. Actuellement les paquets pour Ubuntu 10.04 à 11.04 sont disponibles.

– Pour prendre en compte les nouveaux dépôts et avoir une liste des tout deniers paquets à jour, vous devez lancer `sudo apt-get update && sudo apt-get upgrade` de temps en temps, tout spécialement si vous utilisez une ancienne version du LiveDVD.
– Afin d'éviter les problèmes de permissions, vous pouvez utiliser la méthode de connexion `trust` dans `/etc/postgresql/8.4/main/pg_hba.conf` et redémarrer le serveur PostgreSQL avec `sudo service postgresql-8.4 restart`.

3.2 Travaux pratiques

Suite à l'installation du paquet workshop, vous touverez tout les documents dans `/usr/share/pgrouting/workshop/`.

Nous recommendons de copier l'ensemble de ces fichiers dans le répertoire de votre utilisateur et de créer un lient symbolique vers votre serveur web :

```
cp -R /usr/share/pgrouting/workshop ~/Desktop/pgrouting-workshop
sudo ln -s ~/Desktop/pgrouting-workshop /var/www/pgrouting-workshop
```

Vous pouvez ensuite trouver les fichiers des travaux pratiques dans le répertoire `pgrouting-workshop` et y accéder :
– Répertoire Web : http ://localhost/pgrouting-workshop/web/
– Manuel en ligne : http ://localhost/pgrouting-workshop/docs/html/

Note : Des exemples de données additionelles sont disponibles dans le répertoire `data` des travaux pratique. Ils contiennent un fichier compressé contenant les sauvegardes de base de données ainsi qu'un plus petit ensemble de données du réseau routier du centre ville de Denver. Pour décompresser ce fichier, exécuter la commande `tar -xzf ~/Desktop/pgrouting-workshop/data.tar.gz`.

3.3 Base de données à partir de modèle

C'est une bonne idée de créer un modèle de bases de données PostGIS et pgRouting. Cela rendra plus facile la création de nouvelles bases de données incluant déjà les fonctionnalités requises, sans avoir à charger les fichiers SQL pour chaque nouvelle base.

Un script est disponible dans le répertoire `bin` des travaux pratiques pour ajouter des modèles de bases de données incluant les fonctionnalités de PostGIS et pgRouting. Pour créer une base de données modèles, exécutez les commandes suivantes depuis une fenêtre de terminal :

```
cd ~/Desktop/pgrouting-workshop
bash bin/create_templates.sh
```

Maintenant vous pouvez créer une nouvelle base incluant les fonctionnalités pgRouting en utilsant `template_routing` comme modèle. Lancez la commande suivante dans une fenêtre de terminal :

```
# Création de la base de données "routing"
createdb -U postgres -T template_routing routing
```

Vous povez aussi utiliser **PgAdmin III** et des commandes SQL. Démarrez PgAdmin III (disponible sur le LiveDVD), connectez-vous à n'importe quelle base de données et ouvrez l'éditeur SQL afin de lancer les commandes SQL suivantes :

```
-- Création de la base routing
CREATE DATABASE "routing" TEMPLATE "template_routing";
```

3.4 Chargement des functions

Sans une base de données modèle, de nombreux fichiers contenant les fonctions de pgRouting doivent être chargés dans la base. Pour procéder de la sorte, utilsez les commandes suivantes depuis une fenêtre de terminal :

```
# Passer en utilisateur "postgres" (ou lancez, en tant qu'utilisateur "postgres")
sudo su postgres

# Création d'un base routing
createdb routing
createlang plpgsql routing

# Ajouter les fonctions PostGIS
psql -d routing -f /usr/share/postgresql/8.4/contrib/postgis-1.5/postgis.sql
psql -d routing -f /usr/share/postgresql/8.4/contrib/postgis-1.5/spatial_ref_sys.sql

# Ajouter les fonctions de base de pgRouting
psql -d routing -f /usr/share/postlbs/routing_core.sql
psql -d routing -f /usr/share/postlbs/routing_core_wrappers.sql
psql -d routing -f /usr/share/postlbs/routing_topology.sql
```

Encore un fois, vous pouvez utiliser **PgAdmin III** et des commandes SQL. Démarrez PgAdmin III, connextez-vous à n'importe quelle base de données, ouvrez l'éditeur de commande SQL et saisissez les commandes suivantes :

```
-- Création de la base routing
CREATE DATABASE "routing";
```

Connectez-vous ensuite à la base `routing` et ouvrez une nouvelle fenêtre d'éditeur SQL :

```
-- Ajouter le support plpgsql et les fonctions PostGIS/pgRouting
CREATE PROCEDURAL LANGUAGE plpgsql;
```

Maintenant, ouvrez les fichiers `.sql` contenant les fonctions PostGIS/pgRouting listée précédemment et chargez les dans la base de données `routing`.

Note : PostGIS `.sql` files can be stored in different directories. The exact location depends on your version of PostGIS and PostgreSQL. The example above is valid for PostgeSQL/PostGIS version 1.5 installed on OSGeo LiveDVD.

3.5 Données

Les travaux pratiques pgRouting utiliseront les données de Denver d'OpenStreetMap, quisont déjà disponibles sur le LiveDVD. Si vous n'utilisez pas le LiveDV ou si vous voulez télécharger les dernières données ou des données de votre choix, vous pouvez utiliser l'API OpenStreetMap depuis votre fenêtre de terminal :

```
# Télécharger le fichier sampledata.osm
wget --progress=dot:mega -O "sampledata.osm"
        "http://jxapi.openstreetmap.org/xapi/api/0.6/*
                                        [bbox=-105.2147,39.5506,-104.594,39.9139]"
```

L'API a une limite de taille de téléchargment, ce qui peut être problématique pour télécharger une grande étendu géographique avec de nombreux éléments. Une alternative est d'utiliser l'éditeur JOSM, qui utilisera aussi des appels à l'API pour télécharger les données, mais il fournit un interface facile d'utilisation pour les utilisateurs. Vous pouvez sauvegarder les données come un fichier `.osm` pour l'utiliser avec ces travaux pratiques. JSOM est aussi disponible sur le LiveDVD.

Note :
– OpenStreetMap API v0.6, voir pour plus d'informations http ://wiki.openstreetmap.org/index.php/OSM_Protocol_Version_0.6
– Les données de Denver sont disponibles sur le LiveDVD dans le répertoire `/usr/local/share/osm/`

Une alternative, pour de très grandes étendues est d'utiliser le service de téléchargement de CloudMade. Cette entreprise offre des extractions de cartes pour tous les pays du monde. Pour les données du Colorado par exemple, allez sur le page http ://download.cloudmade.com/americas/northern_america/united_states/colorado et téléchargez le fichier compressé `.osm.bz2` :

```
wget --progress=dot:mega http://download.cloudmade.com/americas/northern_america/united_states/colora
```

Warning : Les données d'un pays complet peuvent être trop grande par rapport à l'espace disponible sur le LiveDVD et nécessité des temps de calculs extrêmement long.

CHAPITRE 4

Outils d'import osm2pgrouting

osm2pgrouting est un outils en ligne de commande qui rend simple l'importation de données OpenStreetMap dans une base de données pgRouting. Il contruit le réseau routier automatiquement et crée les tables pour les types de données et les classes de routes. osm2pgrouting a été écrit initialement par Daniel Wendt et est maintenant disponible sur le site du projet pgRouting : http ://www.pgrouting.org/docs/tools/osm2pgrouting.html

Note : Il y a quelques limitations, particulièrement par rapport à la taille du réseay. La version actuelle nécessite le chargement en mémoire de l'ensemble des données, ce qui le rend rapide mais consome aussi beaucoup de mémoire pour les gros enesemble d'objets. Un outils alternatif n'ayant pas de limitation sur la taille du réseauest **osm2po** (http ://osm2po.de). Il est disponible sous licence "Freeware License".

Les données brutres d'OpenStreetMap contiennent bien plus d'éléments et d'informations qu'il est nécessaire pour du routage. Ainsi le format n'est pas utilisable tel-quel avec pgRouting. Un fichier XML `.osm` contient trois types de données majeurs :
– noeuds
– chemins
– relations
Les données de sampledata.osm par exemple ressemble à ce qui suit :

```
<?xml version='1.0' encoding='UTF-8'?>
<osm version='0.6' generator='xapi: OSM Extended API 2.0' ... >
  ...
  <node id='255405560' lat='41.4917468' lon='2.0257695' version='1'
                changeset='19117' user='efrainlarrea' uid='32823' visible='true'
                timestamp='2008-04-02T17:40:07Z'>
  </node>
  <node id='255405551' lat='41.4866740' lon='2.0302842' version='3'
                changeset='248452' user='efrainlarrea' uid='32823' visible='true'
                timestamp='2008-04-24T15:56:08Z'>
  </node>
  <node id='255405552' lat='41.4868540' lon='2.0297863' version='1'
                changeset='19117' user='efrainlarrea' uid='32823' visible='true'
                timestamp='2008-04-02T17:40:07Z'>
  </node>
  ...
  <way id='35419222' visible='true' timestamp='2009-06-03T21:49:11Z'
                version='1' changeset='1416898' user='Yodeima' uid='115931'>
    <nd ref='415466914'/>
```

```
    <nd ref='415466915'/>
    <tag k='highway' v='unclassified'/>
    <tag k='lanes' v='1'/>
    <tag k='name' v='Carrer del Progrés'/>
    <tag k='oneway' v='no'/>
  </way>
  <way id='35419227' visible='true' timestamp='2009-06-14T20:37:55Z'
                  version='2' changeset='1518775' user='Yodeima' uid='115931'>
    <nd ref='415472085'/>
    <nd ref='415472086'/>
    <nd ref='415472087'/>
    <tag k='highway' v='unclassified'/>
    <tag k='lanes' v='1'/>
    <tag k='name' v='carrer de la mecanica'/>
    <tag k='oneway' v='no'/>
  </way>
  ...
  <relation id='903432' visible='true' timestamp='2010-05-06T08:36:54Z'
                  version='1' changeset='4619553' user='ivansanchez' uid='5265'>
    <member type='way' ref='56426179' role='outer'/>
    <member type='way' ref='56426173' role='inner'/>
    <tag k='layer' v='0'/>
    <tag k='leisure' v='common'/>
    <tag k='name' v='Plaça Can Suris'/>
    <tag k='source' v='WMS shagrat.icc.cat'/>
    <tag k='type' v='multipolygon'/>
  </relation>
  ...
</osm>
```

Une description détaillée de tout les types et classes possibles d'OpenStreetMap peuvent-être trouvé ici : http ://wiki.openstreetmap.org/index.php/Map_features.

Lorsuqe vous utilisez osm2pgrouting, nous devons conserver uniquement les noeuds et les chemin ayant pour types et classes celles stipulée dans le fichier `mapconfig.xml` qui seront improtés dans notre base de données routing :

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <type name="highway" id="1">
    <class name="motorway" id="101" />
    <class name="motorway_link" id="102" />
    <class name="motorway_junction" id="103" />
    ...
    <class name="road" id="100" />
  </type>
  <type name="junction" id="4">
    <class name="roundabout" id="401" />
  </type>
</configuration>
```

Le fichier `mapconfig.xml` par défaut est installé dans le répertoire : `/usr/share/osm2pgrouting/`.

4.1 Création de la base de données routing

Avant de lancer osm2pgrouting nous devons créer la base de données et y charger les fonctionalités de PostGIS et pgRouting . Si vous avez installé le modèle de base de données comme décrit dans le chapitre précédent, créer une

base de données prête à utiliser pgRouting est fait par une simple commande. Ouvrez une fenêtre de terminal et utiliser la commande suivante :

```
createdb -U postgres -T template_routing routing
```

... vous avez terminé.

Une alternative consiste à utiliser l'outil **PgAdmin III** et des requêtes SQL. Démarrez PgAdmin III (disponible sur le LiveDVD), connectez vous à une base de données eyt ouvrez l'éditeur de requêtes afin d'y saisir les requêtes SQL suivantes :

```
-- Créationde la base routing
CREATE DATABASE "routing" TEMPLATE "template_routing";
```

Sinon, vous devez manuellement charger différents fichier dans la base de données. Voir : *previous chapter*.

4.2 Utiliser osm2pgrouting

La prochaine étape c'est de lancer l'outil `osm2pgrouting`, qui est un outil en ligne de commande, donc vous devrez l'utiliser depuis une fenêtre de terminal.

Nous prendrons par défaut le fichier de configuration `mapconfig.xml` et la base de données `routing` que nous avons créer précédemment. De plus nous prendrons le fichier `~/Desktop/pgrouting-workshop/data/sampledata.osm` comme données brutes. Ce fichier contient seulement les données OSM du centre ville de Denver afin d'accélérer le processus de chargement des données.

Les données sont disponibles au format compressé, qui doit donc être décompressé soit en utlisant un navigateur de fichier soit en utilisant la commande suivante :

```
cd ~/Desktop/pgrouting-workshop/
tar -xvzf data.tar.gz
```

Lancer ensuite l'outil :

```
osm2pgrouting -file "data/sampledata.osm" \
                          -conf "/usr/share/osm2pgrouting/mapconfig.xml" \
                          -dbname routing \
                          -user postgres \
                          -clean
```

Liste des paramètres possible :

Paramètre	Valeur	Déscription	Requis
-file	<fichier>	le nom du fichier XML .osm	yes
-dbname	<nom_de_base>	le nom de votre base de données	yes
-user	<utilisateur>	le nom de l'utilisateur, qui a le droit d'accès en écriture sur la base de données	yes
-conf	<fichier>	le nom du fichier XML de configuration	yes
-host	<hôte>	l'hôte de votre base de données postgresql (par défaut : 127.0.0.1)	no
-port	<port>	le numéro de port de votre serveur de base de données(par défaut : 5432)	no
-passwd	<mot_de_passe>	le mot de passe pour se connecter à la base de données	no
-clean		Suprrimer les tables précédemment créées	no

Note : Si vous obtenez un message d'erreur relatif aux droits de l'utilisateur postgres, vous pouvez définir la méthode comme `trust` dans le fichier `/etc/postgresql/8.4/main/pg_hba.conf` et redémarrer le serveur PostgreSQL avec la commande `sudo service postgresql-8.4 restart`.

Suivant la taille de votre réseau le temps de cacul et d'importation de données peut être long. Une fois terminé, connectez à votre base de données et vérifiez les tables qui aurait du être créées :

Lancer : `psql -U postgres -d routing -c "\d"`

Si tout se passe bien, le résultat devrait ressembler à ceci :

```
                  List of relations
 Schema |        Name          |   Type   |  Owner
--------+----------------------+----------+----------
 public | classes              | table    | postgres
 public | geometry_columns     | table    | postgres
 public | nodes                | table    | postgres
 public | spatial_ref_sys      | table    | postgres
 public | types                | table    | postgres
 public | vertices_tmp         | table    | postgres
 public | vertices_tmp_id_seq | sequence | postgres
 public | ways                 | table    | postgres
(8 rows)
```

CHAPITRE 5

Création de la topologie du réseau

osm2pgrouting est un outil pratique, mais c'est aussi une *boîte noire*. Il y a de nombreux cas où *osm2pgrouting* ne peut pas être utilisé. Certaines données de réseau sont fournies avec la topologie du réseau qui peut être utilisé par pgRouting tel-quel. Certaines données de réseau sont stockées au format Shape file (`.shp`) et nous pouvons les charger dans une base de données PostgreSQL à l'aide de l'outil de conversion de PostGIS' ‘‘shape2postgresql‘. But what to do then ?

Dans ce chapitre vous allez apprendre comment créer une topologie de réseau en partant de rien. Pour ce faire, nous allons commencer par les données qui contiennent les attributs minimum requis pour le routage et comment constituer étape par étape des données pour pgRouting.

5.1 Charger les données de réseau

Au début nous allors charger une sauvegarde de base de données à partir du répertoire ‘‘data‘‘des travxu pratiques. Ce répertoire contient un fichier compressé incluant une sauvegarde de base de données ainsi qu'un plus petit ensemble de données de réseau du centre ville de Denver. Si vous n'avez pas encore décompressé, faite le en utilisant la comande :

```
cd ~/Desktop/pgrouting-workshop/
tar -xvzf data.tar.gz
```

La commande suivante permet d'importer la sauvegarde de la base de données. Elle ajoutera les fonctions PostGIS et pgRouting à la base, de la même manière que ce nous avons décrit dans le chapitre précédent. Cela chargera aussi le petit échantillon de données de Denver avec un nombre minimum d'attribut, que vous trouverez habituellement dans l'ensemble des données de réseau :

```
# Optionel: supprimer la base de données
dropdb -U postgres pgrouting-workshop

# Chargement du fichier de sauvegarde
psql -U postgres -f ~/Desktop/pgrouting-workshop/data/sampledata_notopo.sql
```

Regardons quelles tables ont été créées :

Lancer : `psql -U postgres -d pgrouting-workshop -c "\d"`

```
                List of relations
 Schema |       Name        | Type  |  Owner
--------+-------------------+-------+----------
 public | classes           | table | postgres
 public | geography_columns | view  | postgres
 public | geometry_columns  | table | postgres
 public | spatial_ref_sys   | table | postgres
 public | types             | table | postgres
 public | ways              | table | postgres
(6 rows)
```

La table contenant les données du réseau routier onle nom `ways`. Elle possède les attributs suivants :

Lancer : `psql -U postgres -d pgrouting-workshop -c "\d ways"`

```
             Table "public.ways"
  Column   |       Type       | Modifiers
-----------+------------------+-----------
 gid       | integer          | not null
 class_id  | integer          |
 length    | double precision |
 name      | character(200)   |
 the_geom  | geometry         |
Indexes:
    "ways_pkey" PRIMARY KEY, btree (gid)
    "geom_idx" gist (the_geom)
Check constraints:
    "enforce_dims_the_geom" CHECK (ndims(the_geom) = 2)
    "enforce_geotype_the_geom" CHECK (geometrytype(the_geom) =
              'MULTILINESTRING'::text OR the_geom IS NULL)
    "enforce_srid_the_geom" CHECK (srid(the_geom) = 4326)
```

Il est habituel dans des données de réseau routier de retrouver au moins les informations suivantes :
– Identifiant de tronçon routier (gid)
– Classe de tronçon (class_id)
– Longuer du tronçon routier (length)
– Nom du tronçon (name)
– La géométrie du tronçon (the_geom)
Cela permet d'afficher le réseau routier comme une couche PostGIS depuis un logiciel SIG, par exemple dans QGIS. Notez ue les informations ne suffisent pas au calcul de routes étant donné qu'il ne contient aucune information relative à la topolgie du réseau.

La prochaine étape consiste à démarrer l'outil en ligne de commande PostgreSQL

```
psql -U postgres pgrouting-workshop
```

... ou d'utiliser PgAdmin III.

5.2 Calcul de la topologie

Pour avoir vos données importé dans une base de données PostgreSQL requière généralement des étapes supplémentaires pour pgRouting. Vous devez vous assurer que vos données fournissent une topologie correcte du réseau, ce qui correspond aux informations par rapport au début et à la fin d'un tronçon.

Si les données de votre réseau ont une déjà telle information vous devez exécuter la fonctions `assign_vertex_id`. Cette fonction permet l'assignation des valeurs pour les colonnes `source` et `target` pour chaque tronçon et il peut prendre en compte le fait qu'un sommet puisse être éloigné d'un autre suivant une certaine tolérance.

```
assign_vertex_id('<table>', float tolerance, '<geometry column', '<gid>')
```

Premièrement nous devons ajouter les colonnes source et target, pour ensuite utiliser la fonction assign_vertex_id ... et attendre :

```
-- Ajouter les colonnes "source" et "target"
ALTER TABLE ways ADD COLUMN "source" integer;
ALTER TABLE ways ADD COLUMN "target" integer;

-- Utiliser la fonction de contruction de topologie
SELECT assign_vertex_id('ways', 0.00001, 'the_geom', 'gid');
```

Note : Exécuter `psql -U postgres -d pgrouting-workshop` depuis votre terminal afin de vous connecter àal a base de données et lancer des commandes PostgreSQL en ligne. Quiter la session avec la commande `\q` .

> **Warning :** La dimension du paramètre tolérance dépends du système de projection de vos données. Habituellement c'est soit "degrés" soit "mètres".

5.3 Ajouter des indexes

Heureusement nous n'avons pas à attendre longtemps étant donné que notre jeu de données est très petit. Mais la quantité de données d'un réseau pourrait être beaucoup plus importante, donc il vaut mieux ajouter des indexes pour les colonnes `source` et `target`.

```
CREATE INDEX source_idx ON ways("source");
CREATE INDEX target_idx ON ways("target");
```

Suite à ces étapes, notre base de données routing ressemble à ceci :

Lancer : `\d`

```
                    List of relations
 Schema |         Name          |   Type    |  Owner
--------+-----------------------+-----------+----------
 public | geography_columns     | view      | postgres
```

```
 public | geometry_columns    | table    | postgres
 public | spatial_ref_sys     | table    | postgres
 public | vertices_tmp        | table    | postgres
 public | vertices_tmp_id_seq | sequence | postgres
 public | ways                | table    | postgres
(6 rows)
```

Lancer : `\d ways`

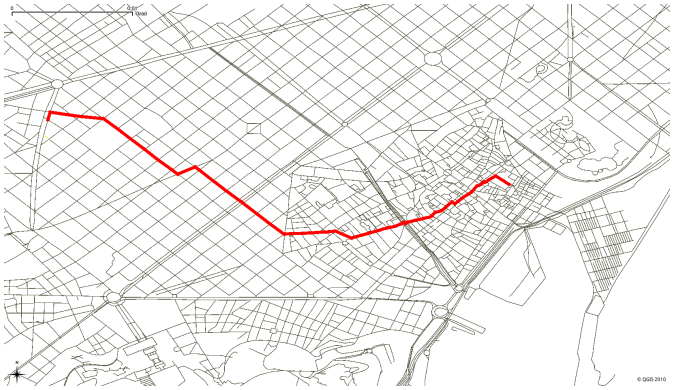
```
                Table "public.ways"
  Column   |       Type        | Modifiers
-----------+-------------------+-----------
 gid       | integer           | not null
 class_id  | integer           |
 length    | double precision  |
 name      | character(200)    |
 the_geom  | geometry          |
 source    | integer           |
 target    | integer           |
Indexes:
    "ways_pkey" PRIMARY KEY, btree (gid)
    "geom_idx" gist (the_geom)
    "source_idx" btree (source)
    "target_idx" btree (target)
Check constraints:
    "enforce_dims_the_geom" CHECK (ndims(the_geom) = 2)
    "enforce_geotype_the_geom" CHECK (geometrytype(the_geom) =
                'MULTILINESTRING'::text OR the_geom IS NULL)
    "enforce_srid_the_geom" CHECK (srid(the_geom) = 4326)
```

Nous sommes fin prêts pour notre première requête de routage avec *l'algorithme de Dijkstra* <*shortest_path*> !

CHAPITRE 6

Plus courts chemins

pgRouting été initialement appelé *pgDijkstra*, puisque il implémentait seulement la recherche de plus court chemin à l'aide de l'agorithme de *Dijkstra*. Plus tard, d'uatres fonctions se sont ajoutées et la bibliotèque fut renommée.



Ce chapitre explique les trois différents algorithmes et les attributs nécessaires.

Note : Si vous lancez l'outils *osm2pgrouting* pour importer des données *OpenStreetMap*, la table des chemins (`ways`) contient déjà tout les attributs nécessaires pour utiliser les fonctions de recherche de plus court chemins. Au contraire, la table `ways` de la base de données `pgrouting-workshop` du *chapitre précédent* manque d'un certain nombre d'attributs, qui sont présentés dans ce chapitre dans les **Prérequis**.

6.1 Dijkstra

L'algorithme de Dijkstraa été la première implémentation disponible dans pgRouting. Il ne nécessite pas d'autre attributs que les champs `source` et `target`, les attributs `id` et `cost`. Il peut être utilisé sur des graphes orientés ou non. Vous pouvez spécifier que votre réseau à un coût de `parcours inverse` (`reverse cost`) ou non.

Prérequis

Pour être en mesure d'utiliser un coût de parcuors invers, vous devez ajouter une colonne de coût. Nous pouvons affecter la longuer au coût de parcours inverse.

```
ALTER TABLE ways ADD COLUMN reverse_cost double precision;
UPDATE ways SET reverse_cost = length;
```

Fonction avec paramètres

```
shortest_path( sql text,
                   source_id integer,
                   target_id integer,
                   directed boolean,
                   has_reverse_cost boolean )
```

Note :
– Les identifiant pour source et target sont les identifiant des noeuds.
– Graphes non-orientés (“directed=false”) implique que le paramètre “has_reverse_cost” est ignoré

6.1.1 Bases

Chaque algorithme a une fonction de *base*.

```
SELECT * FROM shortest_path('
                SELECT gid as id,
                         source::integer,
                         target::integer,
                         length::double precision as cost
                        FROM ways',
                5700, 6733, false, false);
```

```
 vertex_id | edge_id |        cost
-----------+---------+--------------------
      5700 |    6585 |   0.175725539559539
      5701 |   18947 |   0.178145491343884
      2633 |   18948 |   0.177501253416424
       ... |     ... |                 ...
      6733 |      -1 |                   0
 (38 rows)
```

6.1.2 Enveloppe

Enveloppe SANS limite d’étendue

les fonctions enveloppes sont des fonctions qui étendent les fonctions de bases en y ajoutant des transformations, ajoutant des limites de l’étendue géograhpique de la recherhe, etc.. Les enveloppes peuvent changer le format et l’ordre des résultats. Il affectent aussi automatiquement certains paramètre et rend l’utilisation de pgRouting encore plsu simple.

```
SELECT gid, AsText(the_geom) AS the_geom
        FROM dijkstra_sp('ways', 5700, 6733);
```

```
  gid   |                                the_geom
--------+-----------------------------------------------------------
   5534 | MULTILINESTRING((-104.9993415 39.7423284, ... ,-104.9999815 39.7444843))
   5535 | MULTILINESTRING((-104.9999815 39.7444843, ... ,-105.0001355 39.7457581))
   5536 | MULTILINESTRING((-105.0001355 39.7457581,-105.0002133 39.7459024))
    ... | ...
  19914 | MULTILINESTRING((-104.9981408 39.7320938,-104.9981194 39.7305074))
(37 rows)
```

Note : Il est possible de visualiser le chemin dans QGIS. Cela fonctionne pour la requête de recherche du plus court chemin qui retourne une colonne géométrique.
– Créer la connexion à la base de données et ajoutez la table route comme couche de fond.
– Ajoutez une autre couche de la table "ways" mais selectionnez l'option `Build query` avant de l'ajouter.
– Saissez `"gid" IN ( SELECT gid FROM dijkstra_sp('ways',5700,6733))` dans le champ **SQL where clause**.
`SQL query` peut aussi être sélectionné depuis le menu contextuel de la couche.

Enveloppe AVEC une étendue limite

Vous pouvez limiter votre recherche à une zone précise en ajoutant un cadre limite. Cela améliorera les performances tout spécialement pour les réseaux volumineux.

```
SELECT gid, AsText(the_geom) AS the_geom
        FROM dijkstra_sp_delta('ways', 5700, 6733, 0.1);
```

```
  gid   |                                the_geom
--------+-----------------------------------------------------------
   5534 | MULTILINESTRING((-104.9993415 39.7423284, ... ,-104.9999815 39.7444843))
   5535 | MULTILINESTRING((-104.9999815 39.7444843, ... ,-105.0001355 39.7457581))
   5536 | MULTILINESTRING((-105.0001355 39.7457581,-105.0002133 39.7459024))
    ... | ...
  19914 | MULTILINESTRING((-104.9981408 39.7320938,-104.9981194 39.7305074))
(37 rows)
```

Note : La projection des données OSM est en "degrés", donc nous définirons un cadre limite contenant le point de départ et celui d'arrivée plus une zone tampon de `0.1` degrés par exemple.

6.2 A-Étoile

L'algortihme A-Étoile est un autre algrithme bien connu. Il ajoute l'information de la position géographique du début et la fin de chaque tronçon. Cela permet une recherche privilégiant les tronçons proches du point d'arrivée de la recherche.

Prérequis

Pour A-* vous avez besoin de préparer votre table de réseau et d'y ajouter les colonnes latitute/longitude (`x1`, `y1` et `x2`, `y2`) et de calculer leur valeurs.

```
ALTER TABLE ways ADD COLUMN x1 double precision;
ALTER TABLE ways ADD COLUMN y1 double precision;
ALTER TABLE ways ADD COLUMN x2 double precision;
ALTER TABLE ways ADD COLUMN y2 double precision;

UPDATE ways SET x1 = x(ST_startpoint(the_geom));
UPDATE ways SET y1 = y(ST_startpoint(the_geom));

UPDATE ways SET x2 = x(ST_endpoint(the_geom));
UPDATE ways SET y2 = y(ST_endpoint(the_geom));

UPDATE ways SET x1 = x(ST_PointN(the_geom, 1));
UPDATE ways SET y1 = y(ST_PointN(the_geom, 1));

UPDATE ways SET x2 = x(ST_PointN(the_geom, ST_NumPoints(the_geom)));
UPDATE ways SET y2 = y(ST_PointN(the_geom, ST_NumPoints(the_geom)));
```

Note : La fonction `endpoint()` ne fonctionne pas avec certaines versions de PostgresQL (par exemple les version 8.2.5 et 8.1.9). Un moyen de résoudre ce problème consiste à utiliser la fonction `PointN()` à la place :

Fonction avec paramètres

La fonction de recherche de plus court chemin A-* est très semblable à la fonction Dijkstra, bien qu'elle préfère les tronçons qui sont plus proche du point d'arrivée de la recherche. Les heuristiques de cette recherche sont prédéfinis, donc vous aurez besoin de recompiler pgRouting si vous souhaitez modifier ces heuristiques.

```
shortest_path_astar( sql text,
                     source_id integer,
                     target_id integer,
                     directed boolean,
                     has_reverse_cost boolean )
```

Note :
– Les identifiants source et target sont les identifiant des sommets IDs.
– Graphes non-orienté ("directed=false") ne prennent pas en compte le paramètre "has_reverse_cost"

6.2.1 Bases

```
SELECT * FROM shortest_path_astar('
                SELECT gid as id,
                         source::integer,
                         target::integer,
                         length::double precision as cost,
                         x1, y1, x2, y2
                        FROM ways',
                5700, 6733, false, false);
```

```
 vertex_id | edge_id |        cost
-----------+---------+---------------------
      5700 |    6585 |   0.175725539559539
      5701 |   18947 |   0.178145491343884
      2633 |   18948 |   0.177501253416424
```

```
      ... |     ... |                 ...
     6733 |      -1 |                   0
(38 rows)
```

6.2.2 Enveloppe

Fonction envelopper AVEC cadre limite

```
SELECT gid, AsText(the_geom) AS the_geom
        FROM astar_sp_delta('ways', 5700, 6733, 0.1);
```

```
  gid   |                              the_geom
--------+-----------------------------------------------------------------
   5534 | MULTILINESTRING((-104.9993415 39.7423284, ... ,-104.9999815 39.7444843))
   5535 | MULTILINESTRING((-104.9999815 39.7444843, ... ,-105.0001355 39.7457581))
   5536 | MULTILINESTRING((-105.0001355 39.7457581,-105.0002133 39.7459024))
    ... | ...
  19914 | MULTILINESTRING((-104.9981408 39.7320938,-104.9981194 39.7305074))
(37 rows)
```

Note :
– Il n'y a pas actuellement de fonction pour a-* sans cadre limite, étant donnée que le cadre limite permet d'améliorer grandement les performances. Si vous n'avez pas besoin de cadre limite, Dijkstra sera suffisant.

6.3 Shooting-Star

L'algorithme Shooting-Star est le dernier algorithme de recherche de plus court chemin. Sa spécialité c'est qu'il recherche un parcours d'un tronçon à un autre, pas d'un sommet à un sommet comme les agorithmes de Dijkstra et A-Star le font. Cela rend possible la définition de relations entre les tronçons par exemple, et cela résoud certains problèmes liés aux recherches d'un sommets à un autre comme les "tronçons parallèles", qui ont les même sommet de début et de fin mais des coût différents.

Prérequis

Pour Shooting-Star vous avez besoin de préparer votre table de réseau et d'ajouter les colonnes `rule` et `to_cost`. Come l'algorithme A-* il a aussi un fonction heuristique, qui favorise les tronçons plus proche du point d'arrivée.which prefers links closer to the target of the search.

```
-- Ajouter les colonnes rule et to_cost
ALTER TABLE ways ADD COLUMN to_cost double precision;
ALTER TABLE ways ADD COLUMN rule text;
```

L'algorithme Shooting-Star introduit deux nouveaux attributs

Attribut	Déscription
rule	une chaine de caractères contenant une liste d'identifiants de tronçno séparés par une virgule, qui décrivent le sens giratoir (si vous venez de cet tronçon, vous pouvez rejoindre le suivant en ajoutant un coût défini dans la colonne to_cost)
to_cost	un coût pour passer d'un tronçon à un autre (peut être très élevé s'il est interdit de trouner vers ce tronçon ce qui est comparable au coût de parcours d'un tronçon en cas de feu de signalisation)

Fonction avec paramètres

```
shortest_path_shooting_star( sql text,
                    source_id integer,
                    target_id integer,
                    directed boolean,
                    has_reverse_cost boolean )
```

Note :
– Identifiant du départ et de l'arrivée sont des identifiants de tronçons.
– Graphes non-orientés ("directed=false") ne prennent pas en compte le paramètre "has_reverse_cost"

Pour décrire une interdiction de tourner :

```
 gid | source | target | cost | x1 | y1 | x2 | y2 | to_cost | rule
-----+--------+--------+------+----+----+----+----+---------+------
  12 |      3 |     10 |    2 |  4 |  3 |  4 |  5 |    1000 | 14
```

... signifie que le coût pour aller du tronçon 14 au tronçon 12 est de 1000, et

```
 gid | source | target | cost | x1 | y1 | x2 | y2 | to_cost | rule
-----+--------+--------+------+----+----+----+----+---------+------
  12 |      3 |     10 |    2 |  4 |  3 |  4 |  5 |    1000 | 14, 4
```

... signifie que le coût pour aller du tronçon 14 au tronçon 12 via le tronçon 4 est de 1000.

Si vous avez besoin de plusieurs restrictions pour une arrête donnée you devez ajouter plusieurs enregistrements pour ce tronçon avec un restriction particulière.

```
 gid | source | target | cost | x1 | y1 | x2 | y2 | to_cost | rule
-----+--------+--------+------+----+----+----+----+---------+------
  11 |      3 |     10 |    2 |  4 |  3 |  4 |  5 |    1000 | 4
  11 |      3 |     10 |    2 |  4 |  3 |  4 |  5 |    1000 | 12
```

... signifie que le coût pour aller soit du tronçon 4 soit du 12 au 11 est de 1000. Et donc vous devez ordoner vos données par gid lorsque vous chargez vos données dans la fonction de recherche de plus court chemin...

6.3.1 Bases

Un exemple d'utilisation de l'algorithme Shooting Star :

```
SELECT * FROM shortest_path_shooting_star('
                SELECT gid as id,
                         source::integer,
                         target::integer,
                         length::double precision as cost,
                         x1, y1, x2, y2,
                         rule, to_cost
                        FROM ways',
                6585, 8247, false, false);

 vertex_id | edge_id |        cost
-----------+---------+--------------------
     15007 |    6585 |   0.175725539559539
     15009 |   18947 |   0.178145491343884
      9254 |   18948 |   0.177501253416424
       ... |     ... |   ...
      1161 |    8247 |   0.051155648874288
 (37 rows)
```

Warning : L'algorithme Shooting Star calcul un chemin d'un tronçon à un autre (pas d'un sommet à un autre). La colonnes vertex_id contienr le point de départ du tronçon de la colonne edge_id.

6.3.2 Envelopper

```
SELECT gid, AsText(the_geom) AS the_geom
        FROM shootingstar_sp('ways', 6585, 8247, 0.1, 'length', true, true);

  gid   |                                the_geom
--------+-----------------------------------------------------------------
   6585 | MULTILINESTRING((-104.9975345 39.7193508,-104.9975487 39.7209311))
  18947 | MULTILINESTRING((-104.9975487 39.7209311,-104.9975509 39.7225332))
  18948 | MULTILINESTRING((-104.9975509 39.7225332,-104.9975447 39.7241295))
    ... | ...
   8247 | MULTILINESTRING((-104.9978555 39.7495627,-104.9982781 39.7498884))
(37 rows)
```

Note : Il n'y a actuellement pas de fonction enveloppe pour Shooting-Star sans cadre limite, puisque le cadre limite améliore grandement les performances.

CHAPITRE 7

Requêtes de routage avancées

Comme expliqué dans le chapitre précédent une requête de recherche de plus court chemine ressemble généralement à ce qui suit :

```
SELECT * FROM shortest_path_shooting_star(
        'SELECT gid as id, source, target, length as cost, x1, y1, x2, y2, rule,
        to_cost, reverse_cost FROM ways', 6585, 8247, true, true);
```

On parle généralement de **plus court** chemin, ce qui signifie que la longueur d'un arc est son coût. Mais le coût n'est pas nécessairement une longueur, il peut représenter n'importe quoi, par exemple le temps, la surface, le type de routes, etc ... Ou il peut être la combinaison de plusieurs paramètres ("coûts pondérés").

Note : Si vous souhaitez continuer avec une base de données contenant les fonctions pgRouting, les données exemples ainsi que les attributs nécessaires, vous pouvez charger le fichier de sauvegarded la manière suivante.

```
# Optionel: supprimer la base de données
dropdb -U postgres pgrouting-workshop

# Charger le fichier de sauvegarde
psql -U postgres -f ~/Desktop/pgrouting-workshop/data/sampledata_routing.sql
```

7.1 Coûts pondérés

Dans un vrai réseau il y a différents types de limitations ou de préférences suivant les types de routes par exemple. En d'autre termes, nous ne voulons pas calculer *le plus court* chemin mais le chemin *le moins cher* - un chemin avec un coût minimum. Il n'y aucune limitation dans ce qui peut êtreutilsié pour définir le coût.

Lorsque nous avons convertis les données au format OSM en utilisant l'outil osm2pgrouting, nous avons deux autres tables permettant de déifinir les `types` de routes et les `classes`.

Note : Nous passons maintenant à la base de données que nuos avons générée avec osm2pgrouting. Depuis l'invite de commandes de PostgreSQL ceci est possible avec la commande `\c routing`.

Lancer : `SELECT * FROM types;`

```
 id |   name
----+------------
  2 | cycleway
  1 | highway
  4 | junction
  3 | tracktype
```

Lancer : `SELECT * FROM classes;`

```
 id  | type_id |        name         |  cost
-----+---------+---------------------+--------
 201 |       2 | lane                |
 204 |       2 | opposite            |
 203 |       2 | opposite_lane       |
 202 |       2 | track               |
 117 |       1 | bridleway           |
 113 |       1 | bus_guideway        |
 118 |       1 | byway               |
 115 |       1 | cicleway            |
 116 |       1 | footway             |
 108 |       1 | living_street       |
 101 |       1 | motorway            |
 103 |       1 | motorway_junction   |
 102 |       1 | motorway_link       |
 114 |       1 | path                |
 111 |       1 | pedestrian          |
 106 |       1 | primary             |
 107 |       1 | primary_link        |
 107 |       1 | residential         |
 100 |       1 | road                |
 100 |       1 | unclassified        |
 106 |       1 | secondary           |
 109 |       1 | service             |
 112 |       1 | services            |
 119 |       1 | steps               |
 107 |       1 | tertiary            |
 110 |       1 | track               |
 104 |       1 | trunk               |
 105 |       1 | trunk_link          |
 401 |       4 | roundabout          |
 301 |       3 | grade1              |
 302 |       3 | grade2              |
 303 |       3 | grade3              |
 304 |       3 | grade4              |
 305 |       3 | grade5              |
```

La classe de route est liée avec la tables des cheminspar le champ `class_id`. Suite à l'importation des données la valeur de la colonne `cost` n'est pas encore attribuée. Sa valeur peut être modifiée à l'aide d'une requête `UPDATE`. Dans cet exemple les valeurs de coût pour la table des classe sont assigné de façon arbitraire, donc nous exécutons :

```
UPDATE classes SET cost=1 ;
UPDATE classes SET cost=2.0 WHERE name IN ('pedestrian','steps','footway');
UPDATE classes SET cost=1.5 WHERE name IN ('cicleway','living_street','path');
UPDATE classes SET cost=0.8 WHERE name IN ('secondary','tertiary');
```

```
UPDATE classes SET cost=0.6 WHERE name IN ('primary','primary_link');
UPDATE classes SET cost=0.4 WHERE name IN ('trunk','trunk_link');
UPDATE classes SET cost=0.3 WHERE name IN ('motorway','motorway_junction','motorway_link');
```

Pour de meilleures performances, tout spécialement si le réseau est important, il est préférable de créer un index sur la colonnes `class_id` de la table des chemins et eventuellement le champ `id` de la table `types`.

```
CREATE INDEX ways_class_idx ON ways (class_id);
CREATE INDEX classes_idx ON classes (id);
```

L'idée de ces deux tables est de les utiliser afin de spécifier un facteur qui sera multiplié par le coût de parcour d'un tronçon (habituellement la longueur) :

```
SELECT * FROM shortest_path_shooting_star(
        'SELECT gid as id, class_id, source, target, length*c.cost as cost,
                x1, y1, x2, y2, rule, to_cost, reverse_cost*c.cost as reverse_cost
        FROM ways w, classes c
        WHERE class_id=c.id', 6585, 8247, true, true);
```

7.2 Restriction d'accès

Une autre possibilité est de restreindre l'accès à des routes d'un certains types soit en affectant un coût très élevé à un tronçon ayant un certain attribut soit en s'assurant de ne sélectionner aucun de ces tronçons :

```
UPDATE classes SET cost=100000 WHERE name LIKE 'motorway%';
```

En utilisant des sous-requêtes vous pouvez "mixer" vos coût comme bon vous semble et cela modifiera le résultat obtenu imédiatement. Les changements de coûts affecteront la prochaine recherche de plus courts chemins, sans avoir à reconstruire le votre réseau.

Bien entendu, certaines classes de tronçon peuvent aussi être exclues à l'aide d'une clause `WHERE` dans la requête, par exemple pour exclure la classe "living_street" :

```
SELECT * FROM shortest_path_shooting_star(
        'SELECT gid as id, class_id, source, target, length*c.cost as cost,
                x1, y1, x2, y2, rule, to_cost, reverse_cost*c.cost as reverse_cost
        FROM ways w, classes c
        WHERE class_id=c.id AND class_id != 111', 6585, 8247, true, true);
```

Bien entendu, pgRouting vus permet tout types de requêtes SQL supportées par PostgreSQL/PostGIS.

CHAPITRE 8

Script PHP coté serveur

Nous utiliserons un script PHP pour exécuter les requêtes de routafe et renverons le résultat au client web.

Les étapes suivantes sont nécessaires :
– Retrouver les coordonnées du point de départ et de celui d'arrivée.
– Trouver l'arrête la plus proche d'un point de départ ou d'arrivée.
– Prendre soit le noeud de début ou de fin de l'arrête (for Dijkstra/ A-Star) ou l'arrête elle-même (Shooting-Star) comme début ou fin du parcours.
– Exécuter les requête de recherche de plus court chemin.
– Retourner le résultat de la requête en XML ou mieux encore en GeoJSON au client web.

Note : Pour conserver cet exemple aussi simple que possiblle et de mettre en évidence les requête de routafe, ce script PHP ne valide pas les paramètres des requêtes et ne prend pas en compte les problèmes de sécurités de PHP.

Commençons avec quelque modèles PHP et plaçon ces fichiers dans un répertoire accessible par le serveur Apache :

```
<?php

   // Paramétrage de la connexion à la base de données
   define("PG_DB"  , "routing");
   define("PG_HOST", "localhost");
   define("PG_USER", "postgres");
   define("PG_PORT", "5432");
   define("TABLE",   "ways");

   // Récupérer le point de départ
   $start = split(' ',$_REQUEST['startpoint']);
   $startPoint = array($start[0], $start[1]);

   // Récupérer le point d'arrivée
   $end = split(' ',$_REQUEST['finalpoint']);
   $endPoint = array($end[0], $end[1]);

?>
```

8.1 L'arrête la plus proche

Habituellement les points de départ et d'arrivée, qui sont récupérer depuis le client, n'est pas le point de départ ou d'arrivée d'un tronçon. Il est plus simple de retrouver l'arrête la plus proche que le sommet le plus proche, parce que l'algorithme "Shooting-Star" est basé sur les arrêtes. Pour les algorithmes basés sur les sommets (Dijkstra, A-Étoile) nous pouvons choisir le opint de départ ou d'arrivée de l'arrête sélectionnée.

```
<?php

   // Trouver le tronçon le plus proche
   $startEdge = findNearestEdge($startPoint);
   $endEdge   = findNearestEdge($endPoint);

   // FONCTION findNearestEdge
   function findNearestEdge($lonlat) {

      // Connexion à la base de données
      $con = pg_connect("dbname=".PG_DB." host=".PG_HOST." user=".PG_USER);

      $sql = "SELECT gid, source, target, the_geom,
                           distance(the_geom, GeometryFromText(
                            'POINT(".$lonlat[0]." ".$lonlat[1].")', 4326)) AS dist
                    FROM ".TABLE."
                    WHERE the_geom && setsrid(
                           'BOX3D(".($lonlat[0]-0.1)."
                                  ".($lonlat[1]-0.1).",
                                  ".($lonlat[0]+0.1)."
                                  ".($lonlat[1]+0.1).")'::box3d, 4326)
                    ORDER BY dist LIMIT 1";

      $query = pg_query($con,$sql);

      $edge['gid']      = pg_fetch_result($query, 0, 0);
      $edge['source']   = pg_fetch_result($query, 0, 1);
      $edge['target']   = pg_fetch_result($query, 0, 2);
      $edge['the_geom'] = pg_fetch_result($query, 0, 3);

      // Fermer la connexion
      pg_close($con);

      return $edge;
   }

?>
```

8.2 Requête de routage

```
<?php

   // Choisir un algorithme de parcours
   switch($_REQUEST['method']) {

      case 'SPD' : // Shortest Path Dijkstra

        $sql = "SELECT rt.gid, ST_AsGeoJSON(rt.the_geom) AS geojson,
```

```
                    length(rt.the_geom) AS length, ".TABLE.".gid
                FROM ".TABLE.",
                    (SELECT gid, the_geom
                        FROM dijkstra_sp_delta(
                            '".TABLE."',
                            ".$startEdge['source'].",
                            ".$endEdge['target'].",
                            0.1)
                    ) as rt
                WHERE ".TABLE.".gid=rt.gid;";
      break;

    case 'SPA' : // Shortest Path A*

      $sql = "SELECT rt.gid, ST_AsGeoJSON(rt.the_geom) AS geojson,
                    length(rt.the_geom) AS length, ".TABLE.".gid
                FROM ".TABLE.",
                    (SELECT gid, the_geom
                        FROM astar_sp_delta(
                            '".TABLE."',
                            ".$startEdge['source'].",
                            ".$endEdge['target'].",
                            0.1)
                    ) as rt
                WHERE ".TABLE.".gid=rt.gid;";
      break;

    case 'SPS' : // Shortest Path Shooting*

      $sql = "SELECT rt.gid, ST_AsGeoJSON(rt.the_geom) AS geojson,
                    length(rt.the_geom) AS length, ".TABLE.".gid
                FROM ".TABLE.",
                    (SELECT gid, the_geom
                        FROM shootingstar_sp(
                            '".TABLE."',
                            ".$startEdge['gid'].",
                            ".$endEdge['gid'].",
                            0.1, 'length', true, true)
                    ) as rt
                WHERE ".TABLE.".gid=rt.gid;";
      break;

  } // fin switch

  // Connexion à la base de données
  $dbcon = pg_connect("dbname=".PG_DB." host=".PG_HOST." user=".PG_USER);

  // Exécuter une requête
  $query = pg_query($dbcon,$sql);

?>
```

8.3 Sortie au format GeoJSON

OpenLayers permet l'affichage de lignes en utilisant directement des données au format GeoJSON, donc notre script retourne un objet FeatureCollection au format GeoJSON :

```php
<?php

   // Renvoit un chemin au format GeoJSON
   $geojson = array(
      'type'      => 'FeatureCollection',
      'features'  => array()
   );

   // Ajouter un tronçon au tableau GeoJSON
   while($edge=pg_fetch_assoc($query)) {

      $feature = array(
         'type' => 'Feature',
         'geometry' => json_decode($edge['geojson'], true),
         'crs' => array(
            'type' => 'EPSG',
            'properties' => array('code' => '4326')
         ),
         'properties' => array(
            'id' => $edge['id'],
            'length' => $edge['length']
         )
      );

      // Ajouter un tableau d'éléments au tableau de collection d'éléments
      array_push($geojson['features'], $feature);
   }


   // Fermeture de la connexion
   pg_close($dbcon);

   // Renvoyer le résultat
   header('Content-type: application/json',true);
   echo json_encode($geojson);

?>
```

CHAPITRE 9

Client en GeoExt

GeoExt est une librairie JavaScript pour le développement d'applications internet avancées. GeoExt rassemble les capacités d'OpenLayers avec l'interface utilsiateur savvy de Ext JS pour aider au développement d'applications similaires à des applications bureautiques.

Commençons avec un exemple simple d'utilisation de GeoExt et ajoutons-y les fonctionalités de routage :

```
<html>
<head>

<title>Une page GeoExt de base</title>
<script src="ext/adapter/ext/ext-base.js" type="text/javascript"></script>
<script src="ext/ext-all.js"  type="text/javascript"></script>
<link rel="stylesheet" type="text/css" href="ext/resources/css/ext-all.css" />
<script src="OpenLayers/OpenLayers.js" type="text/javascript"></script>
<script src="GeoExt/script/GeoExt.js" type="text/javascript"></script>
<link rel="stylesheet" type="text/css"
      href="GeoExt/resources/css/geoext-all.css" />

<script type="text/javascript">
    Ext.onReady(function() {
        var panel = new GeoExt.MapPanel({
            renderTo: 'gxmap',
            map: {
                layers: [new OpenLayers.Layer.OSM("Simple OSM Map")]
            },
            center: [-11685000, 4827000],
            zoom: 12,
            height: 400,
            width: 600,
            title: 'A Simple GeoExt Map'
        });
    });
</script>
</head>
<body>
<div id="gxmap"></div>
</body>
</html>
```

Dans l'entête nous chargeons l'ensemble des fichiers javascript et css requis pour l'application. Nous définissons aussi une fonction qui sera exécutée à la fin du chargement de la page (Ext.onReady).

Cette fonction cée une instance de'GeoExt.MapPanel <http ://www.geoext.org/lib/GeoExt/widgets/MapPanel.html>'_ avec une couche de fond OpenStreetMap centrée sur Denver. Dans ce code, aucun objet OpenLayers.Map est explicitemet créé ; le GeoExt.MapPanel le fait de façon camouflé : il récupère les options de la carte, le centre, le seuil de zoom et crée une instance de manière appropriée.

Pour permettre à nos utilisateurs de trouver leur direction, nous devons fournir :

- un moyen de sélectionner un algorithme de routage (Dijkstra, A* ou Shooting*),
- un moyen de sélectionner la position de départ et d'arrivée.

Note : Ce chapitre présente uniquement des extraits de codes, le code source complet de la page peut être récupérée depuis `pgrouting-workshop/web/routing-final.html` qui devrait être sur votre bureau. Le fichier complet se trouve à la fin de ce chapitre.

9.1 Outil de sélection de la méthode de routage

Pour sélectionner une méthode de routage, nous utiliserons un Ext.form.ComboBox : il se comporte simplement comme un select html mais est plus simple à controler.

Comme pour le GeoExt.MapPanel, nous avons besoin d'un éléments html pour placer notre control, créons un div dans le body (ayant 'method' comme identifiant) :

```
<body>
  <div id="gxmap"></div>
  <div id="method"></div>
</body>
```

Créons ensuite le combo :

```
var method = new Ext.form.ComboBox({
    renderTo: "method",
    triggerAction: "all",
    editable: false,
    forceSelection: true,
    store: [
        ["SPD", "Shortest Path Dijkstra"],
        ["SPA", "Shortest Path A*"],
        ["SPS", "Shortest Path Shooting*"]
    ]
});
```

Dans l'option `store`, nous avons défini toutes les valeurs pour les méthodes de routages ; les formats sont dans un tableau d'options ou une option est de la forme `[clef,valeur]`. La `clef` sera envoyée au serveur (the script php dans note cas) et la `valeur` affichée dans la liste déroulante.

L'option `renderTo` défini où la liste déroulante doit être affichée , nous utilisons ici notre élément div.

Et pour finir, une valeur par défaut est définie :

```
method.setValue("SPD");
```

Cette partie utilise le composant ExtJS : ni code OpenLayers ni GeoExt.

9.2 Sélectionner un point de départ et d'arrivée

Nous souhaitons permettre à nos utilisateurs de déssiner et déplacer un point de départ et d'arrivée. C'est plus ou moins le comportement des applications comme google map et les autres : l'utilsateur choisi un point de départ à l'aide d'un moteur de recherche ou en cliquant sur la carte. L'applicationinterroge ensuite le serveur afin d'afficher le chemin sur la carte. L'utilisateur peut ensuite modifier son opint de départ et d'arrivée afin que le chemin soit mis à jour automatiquement.

Dans ces travaux pratiques, nous implémenterons seulement la saisie de point sur la carte (dessiner des points et les déplacer) mais il est parfaitement possible d'implémenter un moteur de recherche en utilisant un service web dédié tel que GeoNames ou tout autre service de geocodage.

Pour faire ceci nous aurons besoin d'un outil permettant de dessiner des points (nous utiliserons le control OpenLayers.Control.DrawFeatures) et un outil pour déplacer les points (OpenLayers.Control.DragFeatures sera parfait pour ce travail). Comme leur noms le suppose ces controls sont disponibles dans OpenLayers.

Ces deux controls auront besoin d'un emplacement pour afficher et manipuler les points ; nous aurons besoin d'une couche OpenLayers.Layer.Vector. Dans OpenLayers, une couche vecteur est un endroit où des élements (une geométrie et des attributs) peuvent être affffichée (contrairement à la couche OSM qui est une couche raster).

Comme les couches vecteurs sont légères, nous en utiliserons une seconde pour afficher la route retourner par le service web. L'initialisation de la couche se fait de la manière suivante :

```
// Création de la couche où le chemin sera affiché
var route_layer = new OpenLayers.Layer.Vector("route", {
    styleMap: new OpenLayers.StyleMap(new OpenLayers.Style({
        strokeColor: "#ff9933",
        strokeWidth: 3
    }))
});
```

`"route"` est le nom de la couche, n'importe qu'elle chaîne de caractères peut être utilisée. `styleMap` fournis à la couche un mimum de style avec une couleur de contour particulière et un largeur (en pixel).

La seconde couche est simplement initialiser comme suit :

```
// Création de la couche ou seront affiché les points de départ et d'arrivée
var points_layer = new OpenLayers.Layer.Vector("points");
```

Les deux couches sont ajoutée à l'objet OpenLayers.Map avec :

```
// Ajouter la cocuhe à la carte
map.addLayers([points_layer, route_layer]);
```

Regardons le control pour afficher les points : puisque ce composant à un comportement particulier il est plus simple de créer une nouvelle classe basée sur le control standard OpenLayers.Control.DrawFeatures. Ce control (nommé DrawPoints) est enregistré dans un fichier javascript à part (`web/DrawPoints.js`) :

```
DrawPoints = OpenLayers.Class(OpenLayers.Control.DrawFeature, {

    // this control is active by default
    autoActivate: true,

    initialize: function(layer, options) {
        // only points can be drawn
        var handler = OpenLayers.Handler.Point;
        OpenLayers.Control.DrawFeature.prototype.initialize.apply(
                                this, [layer, handler, options]
                        );
```

```
    },

    drawFeature: function(geometry) {
        OpenLayers.Control.DrawFeature.prototype.drawFeature.apply(
                                this, arguments
                        );
        if (this.layer.features.length == 1) {
            // we just draw the startpoint
            // note: if we want to apply a special style to the
            //       start point we should do this here
        } else if (this.layer.features.length == 2) {
            // we just draw the finalpoint
            // note: if we want to apply a special style to the
            //       final point we should do this here

            // we have all what we need; we can deactivate ourself.
            this.deactivate();
        }
    }
});
```

Dans la fonction `initialize` (qui est le constructeur de la classe) nous spécifions que ce control peut seulement dessiner des point (la variable handler est OpenLayers.Handler.Point).

Le comportement spécifique est implémenté dans la fonction `drawFeature` : puisque nous avons seulement besoin des points de départ et d'arrivée, le control se desactive automatiquemnt lorsque deux points ont été saisie. La désactivation se fait via `this.deactivate()`.

Notre control est ensuite créé avec :

```
// Création du control pour dessiner les points (voir le fichier DrawPoints.js)
var draw_points = new DrawPoints(points_layer);
```

`points_layer` est la couche vecteur créée précédemment.

Et maintenant pour le control DragFeature :

```
// Création du control pour déplacer les points
var drag_points = new OpenLayers.Control.DragFeature(points_layer, {
    autoActivate: true
});
```

Encore une fois, la couche `points_layer``est de type vecteur, ``autoActivate: true` dit à OpenLayers que nous voulons que ce control soit automatiquement activé.

```
// Ajouter le control à la carte
map.addControls([draw_points, drag_points]);
```

Ajouter le control à la carte.

9.3 Envoyer et recevoire des données du service web

Le cheminement de base pour obtenir un chemin depuis le service web est le suivant :

1. transformer nos points en coordonnées de EPSG :900913 en EPSG :4326
2. appeler le service web avec les arguments correctes (nom de la méthode et deux points)
3. lire le résultat retourné par le service web : transformer le GeoJSON en OpenLayers.Feature.Vector

4. transformer toutes les coordonnées de EPSG :4326 à EPSG :900913
5. ajouter le résultat à la couche vecteur

La première étape est quelque chose de nouveau : notre carte utilise le système de projection EPSG :900913 (parce que nous utilisons une couche OSM) mais le service web attends des coordonnées en EPSG :4326 : nous devons re-projeter les données avant de les envoyer. Ce n'est pas bien difficile : nous aurons simplement besoin de la ' librairie javascript Proj4js <http ://trac.osgeo.org/proj4js/>'_.

(La deuxième étape *appeler le service web* est étudié au chapitre suivant.)

Le service web de routage dans pgrouting.php renvoit un objet FeatureCollection au format GeoJSON. Un objet FeatureCollection est simplement un tableau d'élément : un éléments pour chaque tronçon de route. Ceci est très pratique carOpenLayers et GeoExt ont tout ce dont nous avons besoin pour gérer ce format. Pour nous rendre la tâche encore plus simple, nous utiliserons le GeoExt.data.FeatureStore suivant :

```
var store = new GeoExt.data.FeatureStore({
    layer: route_layer,
    fields: [
        {name: "length"}
    ],
    proxy: new GeoExt.data.ProtocolProxy({
        protocol: new OpenLayers.Protocol.HTTP({
            url: './php/pgrouting.php',
            format: new OpenLayers.Format.GeoJSON({
                internalProjection: epsg_900913,
                externalProjection: epsg_4326
            })
        })
    })
});
```

Un Store est simplement un conteneur qui stocke des informations : nous pouvons y ajouter des éléments et les récupérer.

Expliquons les options :

`layer` : le paramètre est une couche vecteur : en spécifiant une couche, le FeatureStore affichera automatiquement les données qu'elle contient. C'est exactement ce dont nous avons besoin pour la dernière étape (*ajouter le résultat à la couche vecteur*) dans la liste ci-dessus.

`fields` : liste tout les attributs renvoyés avec la géométrie : pgrouting.php renvoit la longueurdu segment donc nous le spécifions ici. Notez que cette information n'est pas utilisée dans ces travaux pratiques.

`proxy` : le paramètre proxy specifie où nous devons récupérer les données : dans notre cas depuis le serveur HTTP. Le type de proxy est GeoExt.data.ProtocolProxy : cette classe connecte le monde ExtJS (le Store) et le monde OpenLayers (l'objet protocol).

`protocol` : ce composant OpenLayers est capable d'exécuter des requêtes à un''url'' (notre script php) et de lire la réponse (option `format`). En ajoutant les options `internalProjection` et `externalProjection`, les coordonnées sont reprojetées par l'objet format.

Nous avons maintenant tout ce qu'il nous faut pour gérer les données renvoyées par le service web : le prochain chapitre expliquera comment et quand l'appeler.

9.4 Déclancher l'appel au service web

Nous devons appeler le service web lorsque :

– les deux points sont dessinés

- un point à été déplacé
- la méthode à utiliser a changé

Notre couche vecteur génère une événement (appelé `featureadded`) lorsqu'un nouvel élément est ajouté, nous pouvons utiliser cet événement pour appeler la fonction pgrouting (cette fonction sera présenté dans peu de temps) :

```
draw_layer.events.on({
    featureadded: function() {
        pgrouting(store, draw_layer, method.getValue());
    }
});
```

Note : Avant de continuer quelque mots sur les événements : un événement dans OpenLayers (la même chose s'applique pour ExtJS et les autres frameworks), est un système qui permet à une fonction d'être appelée lorsque *quelque-chose* se passe. Par exemple lorsqu'une couche est ajoutée à la carte ou quand la souris se trouve au dessus d'un objet de la carte. Plusieurs fonctions peuvent être liées à un même événement.

Aucune événemenet n'est généré lorsqu'un point est déplacé, heureusement nous pouvons définir une fonction à notre control DragFeature à appeler lorsqu'un point est déplacé :

```
drag_points.onComplete = function() {
    pgrouting(store, draw_layer, method.getValue());
};
```

Pour la liste déroulante *method*, nous pouvons ajouter une option *select* au contructeur (c'est l'événement déclencher lorsqu'un utilisateur change sa sélection) :

```
var method = new Ext.form.ComboBox({
    renderTo: "method",
    triggerAction: "all",
    editable: false,
    forceSelection: true,
    store: [
        ["SPD", "Shortest Path Dijkstra"],
        ["SPA", "Shortest Path A*"],
        ["SPS", "Shortest Path Shooting*"]
    ],
    listeners: {
        select: function() {
            pgrouting(store, draw_layer, method.getValue());
        }
});
```

Il est maintenant temps de présenter la fonction pgrouting :

```
// global projection objects (uses the proj4js lib)
var epsg_4326 = new OpenLayers.Projection("EPSG:4326"),
    epsg_900913 = new OpenLayers.Projection("EPSG:900913");

function pgrouting(store, layer, method) {
      if (layer.features.length == 2) {
          // erase the previous route
          store.removeAll();

          // transform the two geometries from EPSG:900913 to EPSG:4326
          var startpoint = layer.features[0].geometry.clone();
          startpoint.transform(epsg_900913, epsg_4326);
```

```
            var finalpoint = layer.features[1].geometry.clone();
            finalpoint.transform(epsg_900913, epsg_4326);

            // load to route
            store.load({
                params: {
                    startpoint: startpoint.x + " " + startpoint.y,
                    finalpoint: finalpoint.x + " " + finalpoint.y,
                    method: method
                }
            });
        }
    }
```

La fonction pgrouting appèle me service web à travers l'argument store.

Au début, la fonction vérifie si deux points sont présent dans les paramètres. Ensuite, *select* est appelée pour éffacer le résultat précédent de la couche (souvenez-vous que le Store et la couche vecteur sont lié). Les deux points sont projetés en utilisant une instance de OpenLayers.Projection.

Pour finir, `store.load()` est appelée avec un argument `params` (ils sont passés via un appèle HTTP utilisant la méthode GET).

9.5 Que faire maintenant ?

Possibles améliorations :

- Utiliser un service de géocodage pour récupérer le point de départ / d'arrivée
- Support de plusieurs points
- De jolies icônes pour le point de départ et celui d'arrivée
- Direction du parcour (carte de voyage) : nous avons déjà la distance

9.6 Code source complet

```
<html>
<head>

<title>Une page GeoExt de base</title>
<script src="ext/adapter/ext/ext-base.js" type="text/javascript"></script>
<script src="ext/ext-all.js"  type="text/javascript"></script>
<link rel="stylesheet" type="text/css" href="ext/resources/css/ext-all.css" />
<script src="OpenLayers/OpenLayers.js" type="text/javascript"></script>
<script src="GeoExt/script/GeoExt.js" type="text/javascript"></script>
<link rel="stylesheet" type="text/css"
      href="GeoExt/resources/css/geoext-all.css" />

<script src="DrawPoints.js" type="text/javascript"></script>

<script src="proj4js/lib/proj4js.js" type="text/javascript"></script>

<script type="text/javascript">

     // Objets globaux projection (utilise la librairie proj4js)
     var epsg_4326 = new OpenLayers.Projection("EPSG:4326"),
         epsg_900913 = new OpenLayers.Projection("EPSG:900913");
```

```
 function pgrouting(store, layer, method) {
     if (layer.features.length == 2) {
         // Effacer le chemin précédent
         store.removeAll();

         // Re-projète les deux géométries de EPSG:900913 et EPSG:4326
         var startpoint = layer.features[0].geometry.clone();
         startpoint.transform(epsg_900913, epsg_4326);
         var finalpoint = layer.features[1].geometry.clone();
         finalpoint.transform(epsg_900913, epsg_4326);

         // Charge le chemin
         store.load({
             params: {
                 startpoint: startpoint.x + " " + startpoint.y,
                 finalpoint: finalpoint.x + " " + finalpoint.y,
                 method: method
             }
         });
     }
}

Ext.onReady(function() {
    // Création du paneau carte
    var panel = new GeoExt.MapPanel({
        renderTo: "gxmap",
        map: {
            layers: [new OpenLayers.Layer.OSM("Simple OSM Map")]
        },
        center: [-11685000, 4827000],
        zoom: 12,
        height: 400,
        width: 600,
        title: "A Simple GeoExt Map"
    });
    var map = panel.map;

    // Création de la couche où le chemin sera dessiné
    var route_layer = new OpenLayers.Layer.Vector("route", {
        styleMap: new OpenLayers.StyleMap(new OpenLayers.Style({
            strokeColor: "#ff9933",
            strokeWidth: 3
        }))
    });

    // Création de la couche où le point de départ et d'arrivée sront dessinés
    var points_layer = new OpenLayers.Layer.Vector("points");

    // Lorsqu'un nouveau point est ajouté à la couche, appeler la fonction pgrouting
    points_layer.events.on({
        featureadded: function() {
            pgrouting(store, points_layer, method.getValue());
        }
    });

    // Ajouter la couche à la carte
    map.addLayers([points_layer, route_layer]);
```

```
        // Création du control pour dessiner les point (voir le fichier DrawPoints.js)
        var draw_points = new DrawPoints(points_layer);

        // Création du control pour déplacer les points
        var drag_points = new OpenLayers.Control.DragFeature(points_layer, {
            autoActivate: true
        });

        // Lorsqu'un point est déplacé, appeler la fonction pgrouting
        drag_points.onComplete = function() {
              pgrouting(store, points_layer, method.getValue());
        };

        // Ajouter les controls à la carte
        map.addControls([draw_points, drag_points]);

        // Création du store pour interroger le service web
        var store = new GeoExt.data.FeatureStore({
            layer: route_layer,
            fields: [
                {name: "length"}
            ],
            proxy: new GeoExt.data.ProtocolProxy({
                protocol: new OpenLayers.Protocol.HTTP({
                    url: "./php/pgrouting.php",
                    format: new OpenLayers.Format.GeoJSON({
                        internalProjection: epsg_900913,
                        externalProjection: epsg_4326
                    })
                })
            })
        });

        // Création de la liste déroulante
        var method = new Ext.form.ComboBox({
            renderTo: "method",
            triggerAction: "all",
            editable: false,
            forceSelection: true,
            store: [
                ["SPD", "Shortest Path Dijkstra"],
                ["SPA", "Shortest Path A*"],
                ["SPS", "Shortest Path Shooting*"]
            ],
            listeners: {
                select: function() {
                    pgrouting(store, points_layer, method.getValue());
                }
            }
        });
        // Définir Disjkstra comme méthode par défaut
        method.setValue("SPD");
    });
</script>
</head>
<body>
<div id="gxmap"></div>
<div id="method"></div>
```

```
</body>
</html>
```