Workshop - FOSS4G routing with pgRouting

リリース *5*

Daniel Kastl, Eric Lemoine

2014 年 11 月 12 日

Contents

Chapter 1

序章

要約

pgRouting は PostGIS にルート検索の機能を追加します。この初歩的なワークショップではどのように利用できるかを示します。このワークショップでは OpenStreetMap の道路のネットワークデータと共に新しい pgRouting のリリースの使い方の実用的な例を提供します。このワークショップではデータの準備、ルート検索クエリの作り方、コストの割り当て、‘plpgsql’ のカスタム関数の作成、そして新しい OpenLayers 3 を使った ウェブマッピングアプリケーション上でのルート表示について順番に説明していきます。

道路ネットワークのナビゲーションは曲がる方向の制限や時間に依存した属性をサポートした複雑なルート検索アルゴリズムを要求します。 pgRouting は拡張可能なオープンソースのライブラリで、最短経路探索のための多様なツールを PostgreSQL 及び PostGIS の拡張として提供します。このワークショップでは実際の道路ネットワーク上で pgRouting を用いた最短経路探索について、また早く結果を取得するにはどのようなデータ構造が重要なのかを説明します。また、 GIS アプリケーションにおける pgRouting では難しいことや制限についても学ぶでしょう。

実用的な例として、ワークショップでは OpenStreetMap のデータを使用します。あなたはデータを必要なフォーマットに変換する方法やデータを“ コスト”の属性を用いて調整する方法を学ぶでしょう。さらに、“ Dijkstra ”、“ A-Star ”以外の pgRouting が提供しているものや、最近、何がライブラリに追加されたのかを紹介します。ワークショップの終わりまでにはあなたは pgRouting の使い方や利用可能なネットワークデータの取得方法について十分理解するでしょう。

行と列から地図に描画するための出力を取得する方法を学ぶため、私達は OpenLayers 3 を用いて簡単な地図の GUI を構築します。私達は昨年の受講者たちのフィードバックを受け、簡単なブラウザアプリケーションを構築する基本的な手順を通してあなたに教えたいと考えています。私達のゴールはこれをできる限り簡単に作り、他の FOSS4G のツールと統合するのは難しくないというのを示すことです。‘plpgsql’ 上で独自の PostgreSQL のストアドプロシージャを書くことは、 Geoserver に最短経路クエリを通す便利な手段となるでしょう。

前提条件

- ワークショップの難易度: 中級
- 参加者の予備知識: SQL (PostgreSQL, PostGIS), Javascript, HTML
- 備品: このワークショップでは OSGeo Live を使用します。

プレゼンター及び著者

- *Daniel Kastl* は Georepublic の創立者及び CEO で、ドイツと日本で働いています。 pgRouting コミュニティのモデレート及びプロモートを行っていて、プロジェクト開始時からの開発を行っており、また、日本におけるアクティブな OSM の貢献者でもあります。

- *Frederic Junod* は Camptocamp のスイスオフィスに 6 年間勤務しています。ブラウザ (GeoExt, OpenLayers) からサーバ (MapFish, Shapely, TileCache) にいたる多くのオープンソース GIS プロジェクトのアクティブな開発者であり、 pgRouting PSC のメンバーでもあります。
- *Eric Lemoine* は Camptocamp のフランスのオフィスで働いています。 OpenLayers プロジェクトのコア開発者で、 PSC メンバーでもあり、 OpenLayers 3 のメイン開発者の一人でもあります。

ライセンス

Supported by

Camptocamp

Georepublic

Chapter 2

使用するFOSS4Gツールについて

このワークショップでは、ワークショップのタイトルに記述されたものの他にも、いくつかの FOSS4G ツールを使用します。また、FOSS4G ソフトウェアの多くは他のオープンソースプロジェクトに関連していますが、それらを全てリストしようとすると際限がなくなるでしょう。このワークショップでは 4 つの FOSS4G プロジェクトに焦点をあてます。

2.1 pgRouting

経路探索と他のネットワーク解析機能を追加します。pgRouting の前身である pgDijkstra は、 Camptocamp の Sylvain Pasche により書き起こされ、後に Orkney により拡張されて、pgRouting に名称変更されました。このプロジェクトは、今は Georepublic 、 iMaptools と広範なユーザコミュニティにより、サポート、維持されています。

pgRouting は OSGeo 財団 の OSGeo Labs プロジェクトで、 OSGeo Live に含まれています。

pgRouting は以下の機能を提供します。

- 全点対間最短経路探索 - ジョンソンのアルゴリズム [1]
- 全点対間最短経路探索 - ワーシャル-フロイド法 [1]
- A* アルゴリズムによる最短経路探索
- 双方向ダイクストラ法による最短経路探索 [1]
- 双方向 A* アルゴリズムによる最短経路探索 [1]
- ダイクストラ法による最短経路探索
- 到達圏探索
- K-最短経路探索 - 複数の代替経路探索 [1]
- K-ダイクストラ法 - 1 対多の最短経路探索 [1]
- 巡回セールスマン問題
- 交差点での進入制限付き最短経路探索 (TRSP) [1]

- Shooting Star アルゴリズムによる最短経路探索 [2]

データベース上で経路探索を行う方法の利点には、以下のようなものがあります。

- データや属性を、多くのクライアント、例えば QGIS や uDig から、JDBC や ODBC、もしくは 直接 Pl/pgSQL を発行することで、変更することが可能です。クライアントは PC やモバイル端末でもかまいません。
- データの変更は、経路探索エンジン経由ですぐに反映させることが可能です。事前の計算処理は必要ありません。
- “コスト” パラメータは SQL 経由で動的に計算可能で、複数の列やテーブルからの値を使用することも可能です。

pgRouting は GPLv2 ライセンスで提供され、個人、企業及び団体からなる、成長中のコミュニティによってサポートされています。

pgRouting ウェブサイト: http://www.pgrouting.org

[1] pgRouting 2.0.0 の **新機能**
[2] pgRouting 2.0.0 で廃止された機能

2.2 OpenStreetMap

“OpenStreetMap (OSM) **は道路地図などの地理データを全世界で自由に作成し提供することを目的としています。***OSM* **プロジェクトはもともと、あなたが** *”* **フリー** *(***訳注：無料, 自由の両義***)”* **だと思っているほとんどの地図という物が、実は法的あるいは技術的な利用制限がある、ということから始まりました。***OSM* **からみると、こういった制限は、誰もが生産的あるいは想像だにしない方法で地理データを使ってイノベーションを産み出そうとすることを妨げています。** ” (原文: http://wiki.openstreetmap.org/wiki/JA:Portal:Press)

OpenStreetMap は pgRouting で使用するにあたり、完全なデータソースです。なぜなら、自由に利用可能で、データを処理する上で技術的な制限が無いからです。データの入手可能性は、まだ国によって変わりますが、世界的な網羅範囲は日に日に拡大していっています。

OpenStreetMap は以下のトポロジカルデータ構造を使用しています。

- ノードは地理情報的な位置を持つポイントです。
- ウェイはノードのリストで、ポリラインまたはポリゴンを表現します。
- リレーションは、ノードやウェイ、他のリレーションをグループ化する要素で、プロパティを割り当てることが可能です。
- タグは、ノードやウェイ、リレーションに対して適用可能で、名称=値のペアから構成されています。

OpenStreetMap ウェブサイト: http://www.openstreetmap.org

2.3 osm2pgrouting

osm2pgrouting は OpenStreetMap データを pgRouting データベースにインポートするのを容易にするコマンドラインツールです。このツールは経路探索ネットワークトポロジーを自動的に構築し、地物種別と道路クラスのテーブルを作成します。osm2pgrouting は最初に Daniel Wendt により実装され、今は pgRouting プロジェクトサイトでホストされています。

osm2pgrouting は GPLv2 ライセンスで提供されています。

プロジェクト ウェブサイト: http://www.pgrouting.org/docs/tools/osm2pgrouting.html

2.4 OpenLayers 3

OpenLayers 3 は地理空間データを、すべての現行のデスクトップ・モバイル Web ブラウザに表示します。ol3 は完全に書き直され、WebGL と 3D を特徴としています。OpenLayers 2 と同様に、様々なデータフォーマットとレイヤ種別をサポートしています。ただし、OpenLayers 2 と異なり、スクラッチから構築されており、最新の HTML5、WebGL や CSS3 のような最新のブラウザ技術に依存しています。

OpenLayers 3 ウェブサイト: http://www.ol3js.org

Chapter 3

インストール及び必要なもの

このワークショップでは次のものが必要です:

- Ubuntu のような Linux オペレーションシステムが望ましい
- Gedit または Medit のようなエディタ
- ルート検索アプリケーションに使う Geoserver
- インターネット接続

全ての必要なツールは OSGeo Live にありますが、このあとのリファレンスで Ubuntu 12.04 もしくはそれ以降が動作しているコンピュータで必要なものをインストールする簡単な概要を紹介します。

3.1 pgRouting

Ubuntu 上の pgRouting は Launchpad レポジトリ から入手できるパッケージが利用可能です:

インストールするのに必要な手順は、ターミナルのウィンドウを開いて以下を実行することです:

```
# Add pgRouting launchpad repository
sudo apt-add-repository -y ppa:ubuntugis/ppa
sudo apt-add-repository -y ppa:georepublic/pgrouting
sudo apt-get update

# Install pgRouting package (for Ubuntu 14.04)
sudo apt-get install postgresql-9.3-pgrouting

# Install osm2pgrouting package
sudo apt-get install osm2pgrouting

# Install workshop material (optional, but maybe slightly outdated)
sudo apt-get install pgrouting-workshop

# For workshops at conferences and events:
# Download and install from http://trac.osgeo.org/osgeo/wiki/Live_GIS_Workshop_Install
wget --no-check-certificate https://launchpad.net/~georepublic/+archive/pgrouting/+files/pgroutinc
sudo dpkg -i pgrouting-workshop_[version]_all.deb
```

上記の作業では PostgreSQL や PostGIS のような全ての必要なパッケージがまだインストールされていなければ一緒にインストールされます。

ノート:

- 最新の変更や更新を得るには時々 `sudo apt-get update & sudo apt-get upgrade` を実行します、古いバージョンの LiveDVD を使っている場合は特にです。

- PostgreSQL のローカルユーザで permission denied エラーを回避するには /etc/postgresql/<version>/main/pg_hba.conf のコネクションメソッドを trust に設定した後、sudo service postgresql restart で PostgreSQL サーバを再起動します。

```
local   all             postgres                                trust
local   all             all                                     trust
host    all             all             127.0.0.1/32            trust
host    all             all             ::1/128                 trust
```

pg_hba.conf は “スーパーユーザ” 権限でのみ編集が可能なので、ターミナルウィンドウから以下を実行します

```
sudo nano /etc/postgresql/9.3/main/pg_hba.conf
```

エディタを閉じるには CTRL-X を押します。

- ワークショップではコマンドを OSGeo Live のデフォルトユーザである user ユーザとして実行します。

3.2 ワークショップ

ワークショップのパッケージをインストールしていれば、/usr/share/pgrouting/workshop/ で全てのドキュメントをみつけることができます。

私達はあなたのホームディレクトリにファイルをコピーして、つぎにあなたの webserver のルートフォルダにシンボリックリンクを貼るのを推奨します:

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

これで pgrouting-workshop フォルダからワークショップのファイルを探す事ができ、次のようにアクセスできます

- Web のディレクトリ: http://localhost/pgrouting-workshop/web/
- オンラインマニュアル: http://localhost/pgrouting-workshop/docs/html/

ノート: 追加のサンプルデータがワークショップの data ディレクトリにあります。このファイルを展開するには tar -xzf ~/Desktop/pgrouting-workshop/data.tar.gz を実行します。

3.3 データベースに pgRouting の機能を追加する

version 2.0 から pgRouting の機能は拡張として簡単にインストールすることが可能です。以下が必要となります:

- PostgreSQL 9.1 もしくはそれ以上
- PostGIS 2.x が拡張としてインストールされていること

もしこれらの必要な条件が揃っていれば、ターミナルウィンドウを開いて次のコマンドを実行します (もしくはこれらのコマンドを pgAdmin 3 の上で実行します:

```
# login as user "user"
psql -U user

-- create routing database
CREATE DATABASE routing;
\c routing

-- add PostGIS functions
```

```
CREATE EXTENSION postgis;

-- add pgRouting core functions
CREATE EXTENSION pgrouting;
```

ノート: もし pgRouging の機能を提供している SQL を調べたいのであれば、これらは `/usr/share/postgresql/<version>/contrib/pgrouting-2.0/` で見つける事ができます:

```
-rw-r--r-- 1 root root  4126 Jun 18 22:30 pgrouting_dd_legacy.sql
-rw-r--r-- 1 root root 43642 Jun 18 22:30 pgrouting_legacy.sql
-rw-r--r-- 1 root root 40152 Jun 18 22:30 pgrouting.sql
```

3.4 データ

The pgRouting workshop will make use of OpenStreetMap data, which is already available on OSGeo Live. If you don't use the OSGeo Live or want to download the latest data or the data of your choice, you can make use of OpenStreetMap's API from your terminal window:

```
# Download using Overpass XAPI (larger extracts possible than with default OSM API)
BBOX="-122.8,45.4,-122.5,45.6"
wget --progress=dot:mega -O "sampledata.osm" "http://www.overpass-api.de/api/xapi?*[bbox=${BBOX}]
```

OSM のデータの取得方法に関する多くの情報はこちらにあります:

- OpenStreetMap のダウンロードについての情報は http://wiki.openstreetmap.org/wiki/Downloading_data にあります
- OSGeo Live の `/usr/local/share/osm/` に OpenStreetMap のデータがあります

もっと広いエリアのデータを Geofabrik のダウンロードサービスから取得するという方法もあります。国ごとに抽出したデータをダウンロードして展開するには以下のようにします:

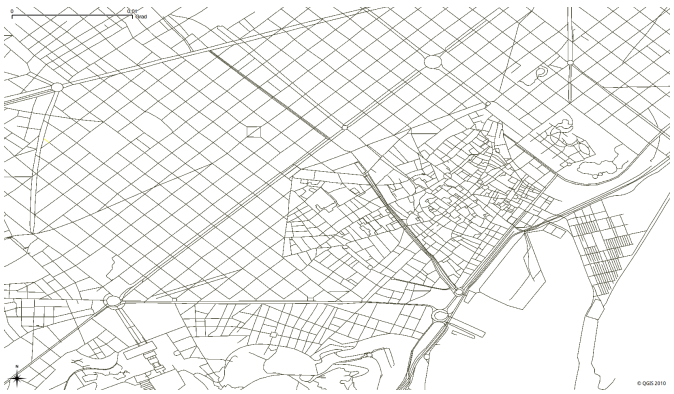
```
wget --progress=dot:mega http://download.geofabrik.de/[path/to/file].osm.bz2
bunzip2 [file].osm.bz2
```

警告: OSGeo Live では各国のデータは大きすぎて処理にすごい時間がかかるでしょう。

Chapter 4

ネットワーク・トポロジーを作成する

osm2pgrouting は便利なツールですが、 **ブラックボックス** でもあります。 *osm2pgrouting* が使えないケースもいくらかあります。取り込むデータが OpenStreetMap のデータでなければ当然使えません。いくつかのネットワークトポロジーを持っているネットワークデータは pgRouting で設定なしで使えます。ネットワークデータはたびたび Shape ファイルフォーマット (`.shp`) に含まれており、データを PostgreSQL データベースにインポートするのに PostGIS の `shape2postgresql` コンバータを使うことができます。では次に何をしましょう？



この章ではスクラッチからネットワーク・トポロジーを作成する方法を学びます。そのために、ルート検索に必要な最小限の属性を含むデータから初めて pgRouting でルート検索可能なデータを構築していく手順を順番に見ていきます。

4.1 ネットワークデータを読み込む

まず最初にワークショップの `data` ディレクトリからデータベースのダンプをロードします。このディレクトリはデータベースのダンプおよび小さいサイズのネットワークデータが入った圧縮ファイルを含んでいます。もしまだデータを解凍していなければ、ファイルの展開をします

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

次のコマンドでデータベースのダンプをインポートします。このコマンドは PostGIS と pgRouting の機能を前の章で説明したのと同じ方法で追加します。このコマンドはまた最小限の属性を持ったサンプルデータを読み込み、これでいくらかのネットワークデータを見つけられるでしょう:

```
# Optional: Drop database
dropdb -U user pgrouting-workshop

# Load database dump file
psql -U user -d postgres -f ~/Desktop/pgrouting-workshop/data/sampledata_notopo.sql
```

さっそく作成されたテーブルを見てみましょう:

次を実行: `psql -U user -d pgrouting-workshop -c "\d"`

```
              List of relations
 Schema |       Name         | Type  |  Owner
--------+--------------------+-------+----------
 public | geography_columns  | view  | user
 public | geometry_columns   | view  | user
 public | raster_columns     | view  | user
 public | raster_overviews   | view  | user
 public | spatial_ref_sys    | table | user
 public | ways               | table | user
(7 rows)
```

このテーブルは `ways` という名前で道路ネットワークデータが含まれています。これは以下の属性から成ります:

次を実行: `psql -U user -d pgrouting-workshop -c "\d ways"`

```
              Table "public.ways"
  Column   |            Type             | Modifiers
-----------+-----------------------------+-----------
 gid       | bigint                      |
 class_id  | integer                     | not null
 length    | double precision            |
 name      | character(200)              |
 osm_id    | bigint                      |
 the_geom  | geometry(LineString,4326)   |
Indexes:
    "ways_gid_idx" UNIQUE, btree (gid)
    "geom_idx" gist (the_geom)
```

普通は道路ネットワークデータは最低限、以下の情報を提供します:

- 道路のリンク ID (gid)
- 道路のクラス (class_id)
- 道路のリンクの長さ (length)
- 道路の名前 (name)
- 道路のジオメトリ (the_geom)

これにより QGIS のような GIS ソフトウェアで道路ネットワークデータが PostGIS レイヤとして表示できます。にもかかわらず、まだルート検索を行うには十分でありません。なぜなら、ネットワーク・トポロジーの情報を含んでいないからです。

次のステップでは PostgreSQL のコマンドラインツールを開始する必要があります

```
psql -U user pgrouting-workshop
```

... もしくは PgAdmin III を使います。

4.2 トポロジーを計算する

あなたのデータを PostgreSQL データベースにインポートした段階では pgRouting を使うためにいくらかの処理がたいてい必要です。あなたのデータが正確なネットワーク・トポロジーを提供できるか確認する必要があり、そしてそれはそれぞれの道路リンクの source ID と target ID の情報から成り立っているということです。

もしあなたのネットワークデータがまだそのようなネットワーク・トポロジーの情報をもっていなければ pgr_createTopology 関数を実行する必要があります。この関数はそれぞれのリンクの source ID と target ID を関連付けて、一定の許容範囲 (tolerance) 内の近くの頂点を “スナップ” することができます。

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

最初に source と target の列を追加し、次に pgr_createTopology 関数を実行し ... そして待ちます。この処理はネットワークのサイズに依存していて、数分から数時間ぐらいかかるでしょう。この処理はまた一時的なデータを保持するために十分なメモリ (RAM または SWAP パーティション) を必要とするでしょう。

```
-- Add "source" and "target" column
ALTER TABLE ways ADD COLUMN "source" integer;
ALTER TABLE ways ADD COLUMN "target" integer;

-- Run topology function
SELECT pgr_createTopology('ways', 0.00001, 'the_geom', 'gid');
```

ノート: データベースに接続して PostgreSQL のシェルをスタートするには psql -U user -d pgrouting-workshop をターミナルで実行して下さい。 \q でシェルから離れます。

警告: 許容範囲のパラメータの大きさはあなたのデータの投影法に依存します。大抵は “度 (degrees)” もしくは “メートル (meters)” です。

4.3 インデックスを追加

あなたのネットワークのテーブルが、source、target 列にインデックスを持つことを確認してください。

```
CREATE INDEX ways_source_idx ON ways("source");
CREATE INDEX ways_target_idx ON ways("target");
```

このステップの後に私達のルート検索データベースは次のようになります:

次を実行: \d

```
                List of relations
 Schema |            Name            |   Type   |  Owner
--------+----------------------------+----------+----------
 public | geography_columns          | view     | user
 public | geometry_columns           | view     | user
 public | raster_columns             | view     | user
 public | raster_overviews           | view     | user
 public | spatial_ref_sys            | table    | user
 public | ways_vertices_pgr          | table    | user
 public | ways_vertices_pgr_id_seq   | sequence | user
 public | ways                       | table    | user
(9 rows)
```

- geography_columns はぞれぞれのテーブルの “geometry” 属性とそのテーブルの SRID のレコードを含むでしょう。
- ways_vertices_pgr は全てのネットワークのノードのリストを含みます。

次を実行: \d ways

```
              Table "public.ways"
  Column   |           Type             | Modifiers
-----------+----------------------------+-----------
```

```
 gid      | integer                   |
 class_id | integer                   | not null
 length   | double precision          |
 name     | text                      |
 osm_id   | bigint                    |
 the_geom | geometry(LineString,4326) |
 source   | integer                   |
 target   | integer                   |
Indexes:
    "ways_gid_idx" UNIQUE, btree (gid)
    "geom_idx" gist (the_geom)
    "ways_source_idx" btree (source)
    "ways_target_idx" btree (target)
```

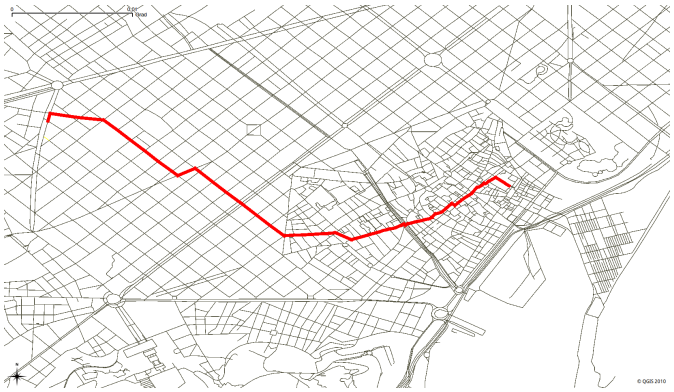
- source と target 列は、今はノードの ID にアップデートされています。
- name はたぶん通りの名称を含むか空になるでしょう。
- length はキロメートルで表された道路のリンクの長さです。

これで **ダイクストラ法** を使った最初のルート検索クエリの準備が整いました！

Chapter 5

pgRouting のアルゴリズム

pgRouting は初めは *pgDijkstra* と呼ばれており、これは **ダイクストラ** 法による最短経路探索のみ実装していたからです。のちに他の機能が追加され、ライブラリの名前が変更されました。



この章では厳選された pgRouting のアルゴリズムやそれらがどの属性を必要とするかを説明します。

ノート: もし *osm2pgrouting* ツールを *OpenStreetMap* のデータをインポートするのに実行していたら、`ways` テーブルにすでに全ての最短経路の機能を実行するためのすべての属性が含まれています。 **前の章** の `pgrouting-workshop` データベースの `ways` テーブルはいくつかの属性が無い代わりに、この章の **前提条件** として紹介します。

5.1 ダイクストラ法による最短経路探索

ダイクストラ法は pgRouting で最初に実装されたアルゴリズムです。このアルゴリズムは `source` ID と `target` ID と `id` と `cost` 以外の属性は必要としません。このアルゴリズムは 有向グラフ と無向グラフを識別することができます。あなたはあなたのネットワークに `reverse cost` があるかどうかを指定することが可能です。

前提条件

`reverse cost` を使えるようにするには追加の cost 列を追加する必要があります。私達は reverse cost を length としてセットすることができます。

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

説明

pgr_costResult (seq, id1, id2, cost) の行セットを返し、これでパスを作り出せます。

```
pgr_costResult[] pgr_dijkstra(text sql, integer source, integer target, boolean directed, boolean
```

パラメータ

sql SQL のクエリです。以下に続く列からなる行セットを返します:

```
SELECT id, source, target, cost [,reverse_cost] FROM edge_table
```

id エッジの識別子 [int4]

source int4 型の始点ノードの識別子

target int4 型の終点ノードの識別子

cost float8 型のエッジにかかる重み。負の重みはエッジがグラフに挿入されることを防ぎます。

reverse_cost (オプション) エッジの反対方向のコスト。この値は directed および ‘‘has_rcost‘‘ パラメータが true の場合のみ使用されます。(負のコストについては前述の通りです)

source int4 始点ノードの ID

target int4 終点ノードの ID

directed 有向グラフの場合は true を指定

has_rcost true の場合、SQL で生成される行セットの reverse_cost 列は、エッジの逆方向にかかる重みとして使用されます。

pgr_costResult のセットを返します:

seq 行の連番

id1 ノード ID

id2 エッジ ID (最終行は -1)

cost id1 から id2 を横断するコスト

ノート:

- 多くの pgRouting の関数は sql::text を 1 つ目の引数として持っています。最初は戸惑うかもしれないですが、このことは、返される結果が要求された数の属性と正確な属性名を含む限り、ユーザがどんな SELECT 文でも関数の引数として渡すことができるため、関数をとてもフレキシブルにしています。
- ダイクストラ法はネットワークジオメトリを必要としません。
- この関数はジオメトリを返しません、返すのはノードの順序付きリストだけです。

クエリの例

pgr_costResult はいくつかの pgRouting の関数で使われている共通の返り型です。 pgr_dijkstra の場合、最初の列はシーケンシャル ID で、そのあとノード ID 、エッジ ID 、このエッジを通過するコストというように続きます。

```
SELECT seq, id1 AS node, id2 AS edge, cost FROM pgr_dijkstra('
                SELECT gid AS id,
                         source::integer,
                         target::integer,
                         length::double precision AS cost
                        FROM ways',
                30, 60, false, false);
```

クエリの結果

```
 seq | node | edge |        cost
-----+------+------+--------------------
   0 |   30 |   53 |  0.0591267653820616
   1 |   44 |   52 |  0.0665408320949312
   2 |   14 |   15 |  0.0809556879332114
   ...
   6 |   10 | 6869 |  0.0164274192597773
   7 |   59 |   72 |  0.0109385169537801
   8 |   60 |   -1 |                    0
(9 rows)
```

ノート:

- もっと複雑な SQL 文、例えば JOIN を使った場合は結果が間違った順番になるでしょう。この場合は `ORDER BY seq` がパスが再度正しい順番になるのを保証するでしょう。
- 返り値の cost 属性は `sql::text` 引数で指定されたコストを表しています。この例の cost は “キロメートル” 単位の `length` です。コストは時間や距離、それらの組み合わせ、または他の属性や独自の計算式になります。

5.2 A* アルゴリズムによる最短経路探索

A-Star 探索アルゴリズムはもう一つのよく知られたルート検索アルゴリズムです。これはそれぞれのネットワークリンクの source と target に地理的な情報を追加します。このアルゴリズムは、ルート検索クエリが最短経路探索の終点に近いリンクを好むようにします。

前提条件

A-Star はあなたのネットワークのテーブルに事前に latitude/longitude 列 (`x1`, `y1` と `x2`, `y2`) を追加してそれらの値を計算しておく必要があります。

```
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 = ST_x(ST_PointN(the_geom, 1));
UPDATE ways SET y1 = ST_y(ST_PointN(the_geom, 1));

UPDATE ways SET x2 = ST_x(ST_PointN(the_geom, ST_NumPoints(the_geom)));
UPDATE ways SET y2 = ST_y(ST_PointN(the_geom, ST_NumPoints(the_geom)));
```

ノート:

- 以前のバージョンの PostgreSQL はバグのため `ST_startpoint` と `ST_endpoint` を使うことができません。

- PostGIS 2.x から ST_startpoint と ST_endpoint は LINESTRING のジオメトリ型のみ有効で、MULTILINESTING では失敗するでしょう。

そのためちょっとばかり変わったクエリが使われます。もしネットワークデータがマルチジオメトリのラインストリングを実際に含んでいたらクエリは間違った start と end のポイントを与えてしまうでしょう。しかし、たとえ LINESTRING のジオメトリのみしか含まれていなかったとしていても、一般的にはデータは MULTILINESTING としてインポートされます。

説明

A-Star 最短経路関数は Dijkstra 関数ととても似ていて、しかしながらこの関数は検索の対象が近いリンクを好みます。この検索のヒューリスティックは事前に定義され、もしヒューリスティック関数自身に変更を加えたい場合は pgRouting を再コンパイルする必要があります。

pgr_costResult (seq, id1, id2, cost) の行セットを返し、これでパスを作り出せます。

```
pgr_costResult[] pgr_astar(sql text, source integer, target integer, directed boolean, has_rcost
```

パラメータ

sql SQL のクエリです。以下に続く列からなる行セットを返します:

```
SELECT id, source, target, cost, x1, y1, x2, y2 [,reverse_cost] FROM edge_table
```

id エッジの識別子 [int4]

source int4 型の始点ノードの識別子

target int4 型の終点ノードの識別子

cost float8 型のエッジにかかる重み。負の重みはエッジがグラフに挿入されることを防ぎます。

x1 エッジの始点の x 座標

y1 エッジの始点の y 座標

x2 エッジの終点の x 座標

y2 エッジの終点の y 座標

reverse_cost (オプション) エッジの反対方向のコスト。この値は directed および ‘‘has_rcost‘‘ パラメータが true の場合のみ使用されます。(負のコストについては前述の通りです)

source int4 始点ノードの ID

target int4 終点ノードの ID

directed 有向グラフの場合は true を指定

has_rcost true の場合、SQL で生成される行セットの reverse_cost 列は、エッジの逆方向にかかる重みとして使用されます。

pgr_costResult のセットを返します:

seq 行の連番

id1 ノード ID

id2 エッジ ID (最終行は -1)

cost id1 から id2 を横断するコスト

クエリの例

```
SELECT seq, id1 AS node, id2 AS edge, cost FROM pgr_astar('
                SELECT gid AS id,
                         source::integer,
                         target::integer,
                         length::double precision AS cost,
                         x1, y1, x2, y2
                        FROM ways',
                30, 60, false, false);
```

クエリの結果

```
 seq | node | edge |        cost
-----+------+------+---------------------
   0 |   30 |   53 |  0.0591267653820616
   1 |   44 |   52 |  0.0665408320949312
   2 |   14 |   15 |  0.0809556879332114
   ...
   6 |   10 | 6869 |  0.0164274192597773
   7 |   59 |   72 |  0.0109385169537801
   8 |   60 |   -1 |                   0
(9 rows)
```

ノート:

- 上記の Dijkstra と A-Star の結果は同じになります。結果に相違が出るかは場合によります。
- A-Star はネットワークのサイズが大きくなるほどダイクストラ法よりも早くなると想定されます。しかし pgRouting の場合、アルゴリズムのスピードの優位性は、ネットワークデータを選択し、グラフを構築するのに要求される時間に比べると、有意な差としては表れません。

5.3 kDijkstra による複数の最短経路探索

kDijkstra 関数は Dijkstra 関数ととても似ていますが一回の関数呼び出しで複数の目的地を設定することができます。

前提条件

kDijkstra はダイクストラ法から追加の属性は必要ありません。

説明

例えば距離の行列から複数のルートを計算する場合など、全てのコストを計算するのが主な目的であれば、`pgr_kdijkstraCost` はよりコンパクトな結果を返します。パスが重要な場合 `pgr_kdijkstraPath` 関数はそれぞれの行き先における A* または Dijkstra と似た結果を返します。

どちらの関数も `pgr_costResult` (seq, id1, id2, cost) の行セットを返し、パスのコストを集約したものか、パスを返します。

```
pgr_costResult[] pgr_kdijkstraCost(text sql, integer source,
                integer[] targets, boolean directed, boolean has_rcost);

pgr_costResult[] pgr_kdijkstraPath(text sql, integer source,
                integer[] targets, boolean directed, boolean has_rcost);
```

パラメータ

sql SQL のクエリです。以下に続く列からなる行セットを返します:

```
SELECT id, source, target, cost [,reverse_cost] FROM edge_table
```

id エッジの識別子 [int4]

source int4 型の始点ノードの識別子

target int4 型の終点ノードの識別子

cost float8 型のエッジにかかる重み。負の重みはエッジがグラフに挿入されることを防ぎます。

reverse_cost (オプション) エッジの反対方向のコスト。この値は directed および ‘‘has_rcost‘‘ パラメータが true の場合のみ使用されます。(負のコストについては前述の通りです)

source int4 始点ノードの ID

targets int4[] 終点ノードの ID の配列

directed 有向グラフの場合は true を指定

has_rcost true の場合、SQL で生成される行セットの reverse_cost 列は、エッジの逆方向にかかる重みとして使用されます。

pgr_kdijkstraCost は pgr_costResult の集合を返します:

seq 行の連番

id1 始点ノードの ID のパス識別子 (これは常にクエリの始点ポイントとなるでしょう)。

id2 終点ノードの ID のパス識別子

cost id1 から id2 へ横断するコスト。もし終点識別子 ID へのパスが無ければコストは -1.0 になるでしょう。

pgr_kdijkstraPath は pgr_costResult のセットを返します。

seq 行の連番

id1 終点ノードの ID のパス識別子 (終点のパスを特定します)。

id2 エッジ ID のパス

cost このエッジを横断するコストもしくはこの終点へのパスがなければ -1.0

pgr_kdijkstraCost のクエリの例

```
SELECT seq, id1 AS source, id2 AS target, cost FROM pgr_kdijkstraCost('
                SELECT gid AS id,
                         source::integer,
                         target::integer,
                         length::double precision AS cost
                        FROM ways',
                10, array[60,70,80], false, false);
```

クエリの結果

```
 seq | source | target |       cost
-----+--------+--------+------------------
   0 |     10 |     60 | 13.4770181770774
   1 |     10 |     70 | 16.9231630493294
```

```
   2 |     10 |     80 | 17.7035050077573
(3 rows)
```

pgr_kdijkstraPath のクエリの例

```
SELECT seq, id1 AS path, id2 AS edge, cost FROM pgr_kdijkstraPath('
                SELECT gid AS id,
                         source::integer,
                         target::integer,
                         length::double precision AS cost
                        FROM ways',
                10, array[60,70,80], false, false);
```

クエリの結果

```
 seq | path | edge |        cost
-----+------+------+--------------------
   0 |   60 | 3163 |   0.427103399132954
   1 |   60 | 2098 |   0.441091435851107
...
  40 |   60 |   56 |  0.0452819891352444
  41 |   70 | 3163 |   0.427103399132954
  42 |   70 | 2098 |   0.441091435851107
...
 147 |   80 |  226 |  0.0730263299529259
 148 |   80 |  227 |  0.0741906229622583
(149 rows)
```

新しい pgRouting 2.0 リリースで他にも多くの関数が利用可能になりましたが、その多くは同じような方法で動作し、またこのワークショップでその全てに言及するにはとても時間が足りません。 pgRouting の関数の完全なリストは API ドキュメントを参照してください: http://docs.pgrouting.org/

Chapter 6

osm2pgrouting インポートツール

osm2pgrouting は OpenStreetMap のデータを pgRouting データベースにインポートすることができるコマンドラインツールです。これはルート検索ネットワークトポロジを自動的に構築し、フィーチャタイプと道路クラスのテーブルを作成します。 osm2pgrouting は最初 Daniel Wendt によって書かれ、現在は pgRouting プロジェクトのサイトでホストされています: http://www.pgrouting.org/docs/tools/osm2pgrouting.html

---

ノート: いくつかの制約が、特にネットワークサイズに関してあります。現行バージョンの osm2pgrouting は全てのデータをメモリに読み込む必要があり、このため動作は速いのですが、大きなデータセットのためにたくさんのメモリが必要となります。osm2pgrouting の代わりとなるツールとして、ネットワークサイズの制約がない **osm2po** (http://osm2po.de) があります。これは "Freeware License" の下で提供されています。

生の OpenStreetMap のデータは、ルート検索に必要とされるものよりも多くのフィーチャと情報を含んでいます。またフォーマットは pgRouting でそのまま使うのに適していません。 `.osm` XML ファイルは以下の 3 つの主要なフィーチャタイプから成ります:

- ノード
- ウェイ
- リレーション

例えば sampledata.osm のデータは以下のようなものです:

```
<?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 Progrs'/>
    <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='Plaa Can Suris'/>
    <tag k='source' v='WMS shagrat.icc.cat'/>
    <tag k='type' v='multipolygon'/>
  </relation>
  ...
</osm>
```

OpenStreetMap の取り得る全てのタイプとクラスの詳細な説明はこちらで見つけることができます: http://wiki.openstreetmap.org/index.php/Map_features.

osm2pgrouting を使う時は、ルート検索データベースにインポートされるものを定義した `mapconfig.xml` ファイルに指定されているタイプとクラスのノードとウェイのみ処理します。

```
<?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>
```

標準では `mapconfig.xml` は `/usr/share/osm2pgrouting/` にインストールされています。

6.1 ルート検索データベースを作成

osm2pgrouting を実行できるようにするには先にデータベースを作成し、 PostGIS と pgRouting の関数をこのデータベースに読み込まなくてはなりません。もし前の章で記述されているテンプレートデータベースをインストールしていれば、 pgRouting で使えるデータベースはコマンド一発で作ることができます。ターミナルウィンドウを開いて以下を実行します:

```
# Optional: Drop database
dropdb -U user pgrouting-workshop

# Create a new routing database
createdb -U user pgrouting-workshop
psql -U user -d pgrouting-workshop -c "CREATE EXTENSION postgis;"
psql -U user -d pgrouting-workshop -c "CREATE EXTENSION pgrouting;"
```

... これでできました。

別の方法として **PgAdmin III** と SQL コマンドを使うことができます。 PgAdmin III を起動して (LiveDVD にあります)、いずれかのデータベースに接続して SQL エディタを開いてから次の SQL コマンドを実行します:

```
-- create pgrouting-workshop database
CREATE DATABASE "pgrouting-workshop";
\c pgrouting-workshop

CREATE EXTENSION postgis;
CREATE EXTENSION pgrouting;
```

6.2 osm2pgrouting を実行

次のステップは `osm2pgrouting` を実行することです。これはコマンドラインツールなので、ターミナルウィンドウを開く必要があります。

私達は標準の `mapconfig.xml` 設定ファイルと私達が先ほど作った `pgrouting-workshop` データベースを使います。また `~/Desktop/pgrouting-workshop/data/sampledata.osm` を生データとして使います。このファイルはデータを処理するスピードを重視して小地域の OSM のデータのみ含みます。

このワークショップのデータは圧縮ファイルとして提供しているので、最初にファイルマネージャか次のコマンドを使って解凍する必要があります:

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

そしてコンバータを実行します:

```
osm2pgrouting -file "data/sampledata.osm" \
                          -conf "/usr/share/osm2pgrouting/mapconfig.xml" \
                          -dbname pgrouting-workshop \
                          -user user \
  -host localhost \
                          -clean
```

利用可能な全てのパラメータのリスト:

パラメータ	値	詳細	必須かどうか
-file	<file>	あなたの osm xml ファイルの名前	はい
-dbname	<db-name>	あなたのデータベースの名前	はい
-user	<user>	ユーザの名前で、データベースに書き込み権限があること	はい
-conf	<file>	あなたの設定 xml ファイルの名前	はい
-host	<host>	あなたの postgresql データベースのホスト (標準では: 127.0.0.1)	いいえ
-port	<port>	あなたのデータベースのポート番号 (標準では: 5432)	いいえ
-passwd	<passwd>	データベースにアクセスするためのパスワード	いいえ
-prefixtables	<prefix>	テーブル名の先頭に追加する名前	いいえ
-skipnodes		ノードテーブルをインポートしません	いいえ
-clean		前に作られていたテーブルをドロップします	いいえ

計算とインポートはネットワークのサイズに依存し、しばらく時間がかかるでしょう。終わったら、データベースに接続してテーブルが作成されたかチェックしましょう:

次を実行: `psql -U user -d pgrouting-workshop -c "\d"`

もし全てうまく行っていれば以下の様な結果が得られるでしょう:

```
                List of relations
 Schema |           Name            |   Type   | Owner
--------+---------------------------+----------+-------
 public | classes                   | table    | user
 public | geography_columns         | view     | user
 public | geometry_columns          | view     | user
 public | nodes                     | table    | user
 public | raster_columns            | view     | user
 public | raster_overviews          | view     | user
 public | relation_ways             | table    | user
 public | relations                 | table    | user
 public | spatial_ref_sys           | table    | user
 public | types                     | table    | user
 public | way_tag                   | table    | user
 public | ways                      | table    | user
 public | ways_vertices_pgr         | table    | user
 public | ways_vertices_pgr_id_seq | sequence | user
(14 rows)
```

ノート: osm2pgrouting はこのワークショップで使われるものより多くのテーブルを作成し、多くの属性をインポートします。最近追加されたものもあり、いまだに正確なドキュメントがないものもあります。

Chapter 7

より高度なルート検索クエリ

ノート: この章は時間がなければスキップするか、あとで戻ってくることも可能です。

ルート検索アルゴリズムに関する章 に説明があるように、最短経路探索のクエリは一般的に次のようになります:

```
SELECT seq, id1 AS node, id2 AS edge, cost FROM pgr_dijkstra('
                SELECT gid AS id,
                         source::integer,
                         target::integer,
                         length::double precision AS cost
                        FROM ways',
                30, 60, false, false);
```

これは一般的に **最短** 経路と呼ばれていて、あるエッジの長さがそのコストという意味になります。しかし、コストは長さである必要はなく、コストはだいたい何でも、例えば時間、傾斜、舗装状態、道路の種類などになるでしょう。もしくはコストは複数のパラメータの組み合わせ (“重み付けされたコスト”) とすることもできます。

ノート: もし pgRouting の関数、サンプルデータ、そして全ての必要な属性を含んだルート検索データベースを使って進めたいのであれば、次のデータベースダンプファイルを読み込むことができます。

```
# Optional: Drop database
dropdb -U user pgrouting-workshop

# Load database dump file
psql -U user -d postgres -f ~/Desktop/pgrouting-workshop/data/sampledata_routing.sql
```

7.1 重み付けされたコスト

“現実の” ネットワークであれば例えば道路の種類によって違う制約や優先などがあるでしょう。言い換えれば、我々は別に **最短** 経路が欲しいのではなく、 **一番安価な** 経路、最小限のコストで行ける経路が欲しいのです。我々が何をコストとするかについては制限がありません。

私達が osm2pgrouting ツールを使って OSM フォーマットからデータをコンバートしたとき、私達は道路の `types` と道路の `classes` の二つの追加されたテーブルを得ました:

ノート: 前に osm2pgrouting で生成したデータベースに切り替えましょう。 PostgreSQL のシェルからでは `\c routing` コマンドでできます。

次を実行: `SELECT * FROM types ORDER BY id;`

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

次を実行: `SELECT * FROM classes ORDER BY id;`

```
 id  | type_id |       name         | cost | priority | default_maxspeed
-----+---------+--------------------+------+----------+------------------
 100 |       1 | road               |      |        1 |               50
 101 |       1 | motorway           |      |        1 |               50
 102 |       1 | motorway_link      |      |        1 |               50
 103 |       1 | motorway_junction  |      |        1 |               50
 104 |       1 | trunk              |      |        1 |               50
 105 |       1 | trunk_link         |      |        1 |               50
 106 |       1 | primary            |      |        1 |               50
 107 |       1 | primary_link       |      |        1 |               50
 108 |       1 | secondary          |      |        1 |               50
 109 |       1 | tertiary           |      |        1 |               50
 110 |       1 | residential        |      |        1 |               50
 111 |       1 | living_street      |      |        1 |               50
 112 |       1 | service            |      |        1 |               50
 113 |       1 | track              |      |        1 |               50
 114 |       1 | pedestrian         |      |        1 |               50
 115 |       1 | services           |      |        1 |               50
 116 |       1 | bus_guideway       |      |        1 |               50
 117 |       1 | path               |      |        1 |               50
 118 |       1 | cycleway           |      |        1 |               50
 119 |       1 | footway            |      |        1 |               50
 120 |       1 | bridleway          |      |        1 |               50
 121 |       1 | byway              |      |        1 |               50
 122 |       1 | steps              |      |        1 |               50
 123 |       1 | unclassified       |      |        1 |               50
 124 |       1 | secondary_link     |      |        1 |               50
 125 |       1 | tertiary_link      |      |        1 |               50
 201 |       2 | lane               |      |        1 |               50
 202 |       2 | track              |      |        1 |               50
 203 |       2 | opposite_lane      |      |        1 |               50
 204 |       2 | opposite           |      |        1 |               50
 301 |       3 | grade1             |      |        1 |               50
 302 |       3 | grade2             |      |        1 |               50
 303 |       3 | grade3             |      |        1 |               50
 304 |       3 | grade4             |      |        1 |               50
 305 |       3 | grade5             |      |        1 |               50
 401 |       4 | roundabout         |      |        1 |               50
(36 rows)
```

道路の class は ways テーブルと `class_id` フィールドによって紐付けされています。データをインポートしたあとでは `cost` 属性はまだセットされていません。この値は `UPDATE` クエリで変更できます。次の例では classes テーブルの cost の値を任意に与えています。このように実行します:

```
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');
```

よりよいパフォーマンスを得るため、特にネットワークデータが大きい場合は、ways テーブルの class_id フィールドと最終的には types テーブルの id フィールドにインデックスを作成するとよいです。

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

これら二つのテーブルの背景にあるアイデアはそれぞれのリンクのコスト (たいていは長さ) によってその要因を複数指定するためです。

```
SELECT seq, id1 AS node, id2 AS edge, cost FROM pgr_dijkstra('
                SELECT gid AS id,
                         source::integer,
                         target::integer,
                         length * c.cost AS cost
                        FROM ways, classes c
                        WHERE class_id = c.id',
                30, 60, false, false);
```

7.2 制限されたアクセス

他に可能なこととして、ある属性から道路のリンクがとても高いコストになるよう設定するか全体からある道路のリンクを選択しないことである種類の道路を制限されたアクセスとすることです:

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

サブクエリに好きなようにコストを “混ぜる” 事ができ、これによってあなたのルート検索の結果を即座に変えることができます。コストは次の最短経路探索で反映され、これにはあなたのネットワークを再構築する必要がありません。

もちろんある道路のクラスをクエリの WHERE 句で除外することもよくあり、次の例では “living_street” クラスを除外しています:

```
SELECT seq, id1 AS node, id2 AS edge, cost FROM pgr_dijkstra('
                SELECT gid AS id,
                         source::integer,
                         target::integer,
                         length * c.cost AS cost
                        FROM ways, classes c
                        WHERE class_id = c.id AND class_id != 111',
                30, 60, false, false);
```

もちろん pgRouting は PostgreSQL/PostGIS が可能などんな SQL も実行可能です。

Chapter 8

pl/pgsql のラッパーを記述する

多くの pgRouting の関数はアルゴリズムの “低レベル” なインターフェースを提供し、例えばジオメトリを含んだルートではなく ID の順番を返します。 “ラッパー関数” は、違う入力パラメータを提供したり、返された結果を他の形式に変換したりし、これによって読みやすく、またはアプリケーションから使いやすくなります。

ラッパー関数のマイナス面は、たびたび特定のユースケースかネットワークデータに対してだけ有効と想定されることです。このため、 pgRouting 自身は低レベルの関数のみサポートし、各ユーザにユースケースにそって独自のラッパー関数を書いてもらうことに決定しました。

今回扱うラッパー関数は例として共通の変換をしています:

8.1 ネットワークのジオメトリと共にルートを返す

経路のラインのジオメトリと共にルートを返すにはラッパー関数を記述する必要はありません。オリジナルの道路ネットワークテーブルに検索結果に必要十分なリンクがあるからです。

ダイクストラ法による最短経路探索

```
SELECT seq, id1 AS node, id2 AS edge, cost FROM pgr_dijkstra('
                SELECT gid AS id,
                         source::integer,
                         target::integer,
                         length::double precision AS cost
                        FROM ways',
                30, 60, false, false);
```

ジオメトリを含んだ結果

```
SELECT seq, id1 AS node, id2 AS edge, cost, b.the_geom FROM pgr_dijkstra('
                SELECT gid AS id,
                         source::integer,
                         target::integer,
                         length::double precision AS cost
                        FROM ways',
                30, 60, false, false) a LEFT JOIN ways b ON (a.id2 = b.gid);
```

ノート: この JOIN の最後のレコードは、最短経路探索の関数がルートの終わりを示すレコードとして -1 を返すのでジオメトリの値を含みません。

8.2 結果を可視化する

ターミナルのスクリーン上で行や列、数字を眺める代わりに、地図上にルートを可視化する方が面白いでしょう。いくつかの方法があります:

- `CREATE TABLE <table name> AS SELECT ...` を使って **結果をテーブルに保存して** その結果を QGIS で見る方法です。例えば:

```
CREATE TABLE route AS SELECT seq, id1 AS node, id2 AS edge, cost, b.the_geom FROM pgr_dijkstra('
                SELECT gid AS id,
                         source::integer,
                         target::integer,
                         length::double precision AS cost
                        FROM ways',
                30, 60, false, false) a LEFT JOIN ways b ON (a.id2 = b.gid);
```

- **PostGIS レイヤを追加して QGIS の SQL where clause を使う方法:**
  - データベースへのコネクションを作成して背景レイヤとして “ways” テーブルを追加します。
  - “ways” テーブルの違うレイヤを追加しますが、`Build query` を追加する前に選択します。
  - 次の内容を **SQL where clause** フィールドに入力します:

```
"gid" IN ( SELECT id2 AS gid FROM pgr_dijkstra('
                SELECT gid AS id,
                         source::integer,
                         target::integer,
                         length::double precision AS cost
                        FROM ways',
                30, 60, false, false) a LEFT JOIN ways b ON (a.id2 = b.gid)
        )
```

- 次の章の Geoserver での WMS サーバの設定方法を見てください。

8.3 入力パラメータとジオメトリの出力を単純化

次の関数は最短経路探索の Dijkstra 関数を呼ぶときに単純化 (そして標準の値をセット) しています。

ノート: 新しい関数の名前は同じスキーマ内で同じ入力引数を持ったどの既存の関数とも一致してはいけません。しかしながら、違う引数の関数であれば名前をシェアできます (これはオーバーローディングと呼ばれています)。

ダイクストラ法のラッパー

```
--DROP FUNCTION pgr_dijkstra(varchar,int,int);

CREATE OR REPLACE FUNCTION pgr_dijkstra(
                IN tbl varchar,
                IN source integer,
                IN target integer,
                OUT seq integer,
                OUT gid integer,
                OUT geom geometry
        )
        RETURNS SETOF record AS
$BODY$
DECLARE
        sql     text;
```

```
        rec     record;
BEGIN
        seq     := 0;
        sql     := 'SELECT gid,the_geom FROM ' ||
                        'pgr_dijkstra(''SELECT gid as id, source::int, target::int, '
                                        || 'length::float AS cost FROM '
                                        || quote_ident(tbl) || ''', '
                                        || quote_literal(source) || ', '
                                        || quote_literal(target) || ' , false, false), '
                                || quote_ident(tbl) || ' WHERE id2 = gid ORDER BY seq';

        FOR rec IN EXECUTE sql
        LOOP
                seq     := seq + 1;
                gid     := rec.gid;
                geom    := rec.the_geom;
                RETURN NEXT;
        END LOOP;
        RETURN;
END;
$BODY$
LANGUAGE 'plpgsql' VOLATILE STRICT;
```

クエリの例

```
SELECT * FROM pgr_dijkstra('ways',30,60);
```

8.4 緯度/経度 のポイントの間のルートで順番にならんだジオメトリと方位を返す

ここで扱う関数は入力パラメータとして 緯度/経度 のポイントを扱い、 QGIS または Mapserver や Geoserver のような WMS サービスで表示ができるルートを返します。

入力パラメータ

- テーブル名
- 始点として `x1`, `y1` と終点として `x2`, `y2`

出力する列

- シーケンス (例えば結果の順番をあとで変えたりします)
- Gid (例えばオリジナルのテーブルに戻って結果をリンクしたりします)
- ストリート名
- 度単位の方位 (リンクの始点ノードと終点ノードの間で方位角を計算して単純化します)
- コストとしてキロメートル単位の長さ
- 道路のリンクのジオメトリ

関数は内部で何をしているのか:

1. 始点と終点の座標に最も近いノードを探します
2. 最短経路探索の Dijkstra のクエリを実行します

3. 必要に応じてジオメトリを入れ替えます、これは前の道路のリンクの終点ノードが次の道路のリンクの始点だった場合です
4. それぞれの道路のリンクの始点と終点のノードの方位角を計算します
5. レコードのセットとして結果を返します

```
--
--DROP FUNCTION pgr_fromAtoB(varchar, double precision, double precision,
--                          double precision, double precision);

CREATE OR REPLACE FUNCTION pgr_fromAtoB(
                IN tbl varchar,
                IN x1 double precision,
                IN y1 double precision,
                IN x2 double precision,
                IN y2 double precision,
                OUT seq integer,
                OUT gid integer,
                OUT name text,
                OUT heading double precision,
                OUT cost double precision,
                OUT geom geometry
        )
        RETURNS SETOF record AS
$BODY$
DECLARE
        sql     text;
        rec     record;
        source        integer;
        target        integer;
        point        integer;

BEGIN
        -- Find nearest node
        EXECUTE 'SELECT id::integer FROM ways_vertices_pgr
                        ORDER BY the_geom <-> ST_GeometryFromText(''POINT('
                        || x1 || ' ' || y1 || ')'',4326) LIMIT 1' INTO rec;
        source := rec.id;

        EXECUTE 'SELECT id::integer FROM ways_vertices_pgr
                        ORDER BY the_geom <-> ST_GeometryFromText(''POINT('
                        || x2 || ' ' || y2 || ')'',4326) LIMIT 1' INTO rec;
        target := rec.id;

        -- Shortest path query (TODO: limit extent by BBOX)
        seq := 0;
        sql := 'SELECT gid, the_geom, name, cost, source, target,
                                ST_Reverse(the_geom) AS flip_geom FROM ' ||
                        'pgr_dijkstra(''SELECT gid as id, source::int, target::int, '
                                        || 'length::float AS cost FROM '
                                        || quote_ident(tbl) || ''', '
                                        || source || ', ' || target
                                        || ' , false, false), '
                                || quote_ident(tbl) || ' WHERE id2 = gid ORDER BY seq';

        -- Remember start point
        point := source;

        FOR rec IN EXECUTE sql
        LOOP
                -- Flip geometry (if required)
                IF ( point != rec.source ) THEN
                        rec.the_geom := rec.flip_geom;
```

```
                point := rec.source;
        ELSE
                point := rec.target;
        END IF;

        -- Calculate heading (simplified)
        EXECUTE 'SELECT degrees( ST_Azimuth(
                        ST_StartPoint(''' || rec.the_geom::text || '''),
                        ST_EndPoint(''' || rec.the_geom::text || ''') ) )'
                INTO heading;

        -- Return record
        seq     := seq + 1;
        gid     := rec.gid;
        name    := rec.name;
        cost    := rec.cost;
        geom    := rec.the_geom;
        RETURN NEXT;
    END LOOP;
    RETURN;
END;
$BODY$
LANGUAGE 'plpgsql' VOLATILE STRICT;
```

この関数がしないことは:

- BBOX (bounding box - 境界ボックス) によって選択された道路ネットワークの制限をしません (大きなネットワークにとって必要なことです)
- 道路のクラスやその他の属性を返しません
- 一方通行の通りを考慮しません
- エラーハンドリングがありません

クエリの例

```
SELECT * FROM pgr_fromAtoB('ways',-122.662,45.528,-122.684,45.514);
```

クエリの結果をテーブルに保存するには次のようにします

```
CREATE TABLE temp_route AS
    SELECT * FROM pgr_fromAtoB('ways',-122.662,45.528,-122.684,45.514);
--DROP TABLE temp_route;
```

データベースにこのファンクションをインストールできます:

```
psql -U user -d pgrouting-workshop -f ~/Desktop/pgrouting-workshop/data/fromAtoB.sql
```

Chapter 9

GeoServer による WMS サーバ

pl/pgsql のラッパーができましたので、 GeoServer を使用して、WMS レイヤとして利用できるようにしていきます。

GeoServer のインストールは、このワークショップの範囲外ですが、もし OSGeo Live を使用している場合は、既にインストール済みとなります。

9.1 管理者ページへの接続

WMS レイヤを作成するにあたり、GeoServer の管理者ページに接続する必要があります。 OSGeo LiveDVD のデスクトップから *Applications* メニューを開き、 *Geospatial* > *Web Services* > *GeoServer* > *Start GeoServer* を選択します。

一度サーバが起動されると、管理者ページ が表示されます。*Login* ボタンをクリックし、 GeoServer の管理者の認証情報を入力します:

Username `admin`

Password `geoserver`

9.2 レイヤの作成

管理者としてログインしたので、 WMS レイヤの作成が可能となります。GeoServer 用語でいう SQL ビュー (詳細については 公式ドキュメント を参照してください) を作成していきます。

最初にすることは、pgrouting のための新しい **ワークスペース** を作成することです。ページ左側のメニューの *Data* セクションから、*Workspaces* をクリックして *Add new workspace* を選択します。

フォームを以下の形式で埋めます。

Name `pgrouting`

Namespace URI `http://pgrouting.org`

そして、submit ボタンを押下します。

次のステップ: ワークスペースにリンクする新しい **ストア** を作成します。まだ左側メニューがあるので、*Stores* をクリックし、*Add new Store* を選択します。これで設定のためのデータソース種別を選択できるようになりました。*PostGIS* を選択します。

フォームを以下の形式で埋めます。

- Basic Store Info:

  Workspace `pgrouting`

Data Source `pgrouting`

- Connection Parameters:

  host `localhost`

  port `5432`

  database `pgrouting-workshop`

  schema `public`

  user `user`

  password `user`

設定値の残りの箇所については、そのままにしておいて構いません。

最後に、**レイヤ** の作成を行います。*Layers* メニューをクリックし、*Add a new resource* を選択します。新しく作成されたワークスペースとストアのペア `pgrouting:pgrouting` を選択します。

テキストの中から、*Configure new SQL view* をクリックします。

ビューに `pgrouting` と名前を入力し、*SQL statement* を以下で埋めます。

```
SELECT ST_MakeLine(route.geom) FROM (
    SELECT geom FROM pgr_fromAtoB('ways', %x1%, %y1%, %x2%, %y2%
  ) ORDER BY seq) AS route
```

SQL view parameters 内で、*Guess parameters from SQL* をクリックすると、上記の SQL から抽出されたパラメータが表示されます。

各パラメータを以下のように設定します:

- Default value を *0* に設定
- Validation regular expression を `^-?[\d.]+$` に変更

この正規表現は、数値のみにマッチします。

Attributes リスト内:

- *Refresh* を叩くと、一つの属性が現れます (*st_makeline* と記載されています)
- Type を `LineString` に変更し、ジオメトリ列の SRID を `-1` から `4326` に変更します。

Save をクリックしてフォームを保存します。

最後に、レイヤの残りの箇所を設定する必要があります。

この画面で行うただ一つのことは、座標参照系が正しいものであることを確認することです。データベースのジオメトリは *EPSG:4326* で格納されていますが、レイヤが表示される OpenLayers の地図が *EPSG:3857* の形式なので、座標参照系を *EPSG:3857* に変更する必要があります。

coordinate reference system セクションにスクロールダウンし、**Declared SRS** を `EPSG:3857` に、また、**SRS handling** を `Reproject native to declared` に変更します。

Compute from data と *Compute from native bounds* リンクをクリックし、自動的にレイヤの範囲を設定します。ページ末尾の *Save* をクリックし、レイヤを作成します。

Chapter 10

OpenLayers 3 ベースのルート検索インターフェース

この章の目的は、OpenLayers 3 により pgRouting への簡単なウェフヘースのユーサインターフェースを作成することてす。経路探索の出発地と目的地の場所を選択し、出発地点と目的地点からルートを取得することか可能となります。

出発地点と目的地点はユーザにより、地図上でクリックすることで作成されます。その後、出発地点と目的地点の座標は WMS サーバ (GeoServer) に WMS の `GetMap` リクエストとして送信されます。結果画像は地図に **画像** レイヤとして追加されます。

10.1 OpenLayers 3 入門

OpenLayers 3 は OpenLayers 2 を完全に書き直したものてす。 Canvas や WebGL などの最新の JavaScript と HTML5 技術を使用しています。

ウェブページ上で OpenLayers 3 地図を作成するには、 `ol.Map` クラスのインスタンスとなる **地図** オブジェクトを作成します。そうすることで、データレイヤやコントロールをその地図オブジェクトに追加することが可能となります。

地図の中心や解像度 (ズームレベル) は **ビュー** オブジェクトを介して制御します。他の地図ライブラリと異なり、ビューは地図から切り離されています; 優位性の一つとして、複数の地図で同一のビューを共有することが可能になることが上げられます。 こちらのサンプル を参照してください。

OpenLayers 3 は 3 つのレンダラーを持っています: *Canvas* レンダラー、 *WebGL* レンダラー、 *DOM* レンダラー。今のところ、最も機能の多いレンダラーは Canvas となります。とりわけ、 Canvas レンダラーは、他の 2 つがサポートしていない、ベクタレイヤをサポートしています。 Canvas がデフォルトのレンダラーで、このワークショップでも、このレンダラーを使用します。

ノート: 将来的には、 WebGL レンダラーが大量のベクタデータや 3D オブジェクトを描画するために使用されるようになります。

10.2 最小限の地図の作成

では、最初の OpenLayers 3 地図を作成しましょう: テキストエディタを開き、以下のコードをコピーして `ol3.html` という名前のファイルを作成します。このファイルは、 `Desktop` に保存して、ウェブブラウザで開くことができます。

```
1  <!DOCTYPE html>
2  <html>
3    <head>
```

```
4     <title>ol3 pgRouting client</title>
5     <meta charset="utf-8">
6     <link href="ol3/ol.css" rel="stylesheet">
7     <style>
8     #map {
9       width: 100%;
10      height: 400px;
11    }
12    </style>
13    </head>
14    <body>
15    <div id="map"></div>
16    <script src="ol3/ol.js"></script>
17    <script type="text/javascript">
18    var map = new ol.Map({
19      target: 'map',
20      layers: [
21        new ol.layer.Tile({
22          source: new ol.source.OSM()
23        })
24      ],
25      view: new ol.View({
26        center: [-13657275.569447909, 5699392.057118396],
27        zoom: 10
28      }),
29      controls: ol.control.defaults({
30        attributionOptions: {
31          collapsible: false
32        }
33      })
34    });
35    </script>
36    </body>
37  </html>
```

このウェブページは OpenStreetMap レイヤの簡単な地図を含み、既定の位置を中心としています。今のところ、ルート検索に関連したコードはありません; 単に標準のナビゲーションツールを備えた簡単な地図となります。

各行では以下を行っています:

- 6 行目: デフォルトの OpenLayers CSS ファイルを読み込みます。
- 7 〜 12 行目: 地図の DOM 要素にサイズを割り当てるため、 CSS ルールを記述します。
- 15 行目: 地図のために div 要素を追加します。要素の識別子は `map` とします。
- 16 行目: OpenLayers 3 のライブラリコードを読み込みます。 `ol` 名前空間に含まれる関数とクラスは、ここで使用可能となります。
- 18 〜 29 行目: このサンプル特有の JavaScript コードとなります。

では、 OpenLayers 3 の地図を作成する箇所について、さらに深く見ていきましょう:

```
var map = new ol.Map({
  target: 'map',
  layers: [
    new ol.layer.Tile({
      source: new ol.source.OSM()
    })
  ],
  view: new ol.View({
    center: [-13657275.569447909, 5699392.057118396],
    zoom: 10
  }),
```

```
  controls: ol.control.defaults({
    attributionOptions: {
      collapsible: false
    }
  })
});
```

このコードは target が HTML ページ内の map DOM 要素となる ol.Map インスタンスを作成します。地図は **タイルレイヤ** の設定を含み、レイヤは OpenStreetMap の *source* 設定を含みます。地図はさらに、既定の *center* と *zoom* レベルの既定値を定義した *view* インスタンス (ol.View クラス) の設定を含みます。

コード内の center と zoom レベルを変更し、ブラウザでページを再読み込みすることで、その効果を確認することができます。さらに、ブラウザの JavaScript コンソールで動的にビューを変更するすることも可能です。例:

```
map.getView().getCenter();
map.getView().setCenter([-29686, 6700403]);
map.getView().setRotation(Math.PI);
```

10.3 WMS GET パラメータ

以下のコードを地図の作成の後に追加します。

```
var params = {
  LAYERS: 'pgrouting:pgrouting',
  FORMAT: 'image/png'
};
```

params オブジェクトは GeoServer に送信される WMS GET パラメータを保持します。ここでは、変更されることのない値: レイヤ名称と出力フォーマット を設定します。

10.4 “出発地” と “目的地” の選択

地図上でのクリックにより、ユーザが出発地と目的地を追加できるようにすることが必要です。以下のコードを追加します:

```
// The "start" and "destination" features.
var startPoint = new ol.Feature();
var destPoint = new ol.Feature();

// The vector layer used to display the "start" and "destination" features.
var vectorLayer = new ol.layer.Vector({
  source: new ol.source.Vector({
    features: [startPoint, destPoint]
  })
});
map.addLayer(vectorLayer);
```

上記コードは 2 つのベクタフィーチャを作成し、 1 つは “出発地” 、もう 1 つは “目的地” となります。 これらのフィーチャは、今のところ空 - ジオメトリを含まない - 状態となります。

上記コードは、さらにベクタレイヤを作成し、 “出発地” と “目的地” のフィーチャを追加しています。さらに、 map の addLayer メソッドにより、地図にベクタレイヤを追加しています。

次に、以下のコードを追加します:

```
// A transform function to convert coordinates from EPSG:3857
// to EPSG:4326.
var transform = ol.proj.getTransform('EPSG:3857', 'EPSG:4326');
```

```
// Register a map click listener.
map.on('click', function(event) {
  if (startPoint.getGeometry() == null) {
    // First click.
    startPoint.setGeometry(new ol.geom.Point(event.coordinate));
  } else if (destPoint.getGeometry() == null) {
    // Second click.
    destPoint.setGeometry(new ol.geom.Point(event.coordinate));
    // Transform the coordinates from the map projection (EPSG:3857)
    // to the server projection (EPSG:4326).
    var startCoord = transform(startPoint.getGeometry().getCoordinates());
    var destCoord = transform(destPoint.getGeometry().getCoordinates());
    var viewparams = [
      'x1:' + startCoord[0], 'y1:' + startCoord[1],
      'x2:' + destCoord[0], 'y2:' + destCoord[1]
    ];
    params.viewparams = viewparams.join(';');
    result = new ol.layer.Image({
      source: new ol.source.ImageWMS({
        url: 'http://localhost:8082/geoserver/pgrouting/wms',
        params: params
      })
    });
    map.addLayer(result);
  }
});
```

上記のコードは、地図の `click` イベントのためのリスナー関数を登録します。このことは、 OpenLayers 3 は地図上でのクリックイベントの検知の度に、この関数を呼び出すことを意味しています。

リスナー関数に渡されるイベントオブジェクトは、 クリックの地理的な位置を表す `coordinate` プロパティを含みます。 coordinate は “出発地” と “目的地” フィーチャの **ポイント** ジオメトリを作成するのに使用されます。

一度、始点と目的地のポイントが (2 度のクリックにより) 確定すると、2 つの座標値のペアは、地図の投影法 (`EPSG:3857`) からサーバの投影法 (`EPSG:4326`) に `transform` 関数を使用して変換されます。

`viewparams` プロパティは WMS GET パラメータオブジェクトに設定されます。このプロパティの値は特別な意味を持ちます: GeoServer はこのレイヤのための SQL クエリを実行する前に、この値を代用します。例えば、もし:

```
SELECT * FROM ways WHERE maxspeed_forward BETWEEN %min% AND %max%
```

の場合、 `viewparams` は `viewparams=min:20;max:120` となり、 PostGIS に送信される SQL クエリは:

```
SELECT * FROM ways WHERE maxspeed_forward BETWEEN 20 AND 120
```

となります。最後に、param オブジェクトが渡された、新しい OpenLayers WMS レイヤが作成されて、地図に追加されます。

10.5 結果のクリア

以下のコードを HTML ページの地図の `div` の後に追加します:

```
<button id="clear">clear</button>
```

上記により、ルート検索ポイントをクリアし、新しいルート検索クエリを開始することを可能にするボタンが作成されます。

次に、以下のコードを JavaScript コードに追加します:

```
var clearButton = document.getElementById('clear');
clearButton.addEventListener('click', function(event) {
  // Reset the "start" and "destination" features.
  startPoint.setGeometry(null);
  destPoint.setGeometry(null);
  // Remove the result layer.
  map.removeLayer(result);
});
```

このボタンがクリックされた際は、 `addEventListener` に渡されたこの関数が実行されます。この関数は “出発地” と “目的地” のフィーチャをリセットし、ルート検索結果レイヤを地図から削除します。