pgRouting 翻訳文書

平成 20 年 4 月

日本ユニシス株式会社

目次

はじめに

本資料は、株式会社オークニーの PostLBS/pgRouting サイトに記載の pgRouting の機能説明原文を翻訳していますが、下記のページは翻訳の対象外としています。

- Installation on Xubuntu 7.04
- Installation on Ubuntu 7.10 Beta
- Tutorials ： SQL tips & tricks
- Print Media
- Workshops

また、翻訳文書の 2.5 pgRouting 機能には原文にはないグラフ図などを挿入しています。
なお、原文と翻訳文書のページ構成が必ずしも一致していません。ご容赦ください。

最後に、本資料が皆様の pgRouting 機能理解の一助になれば幸いでございます。

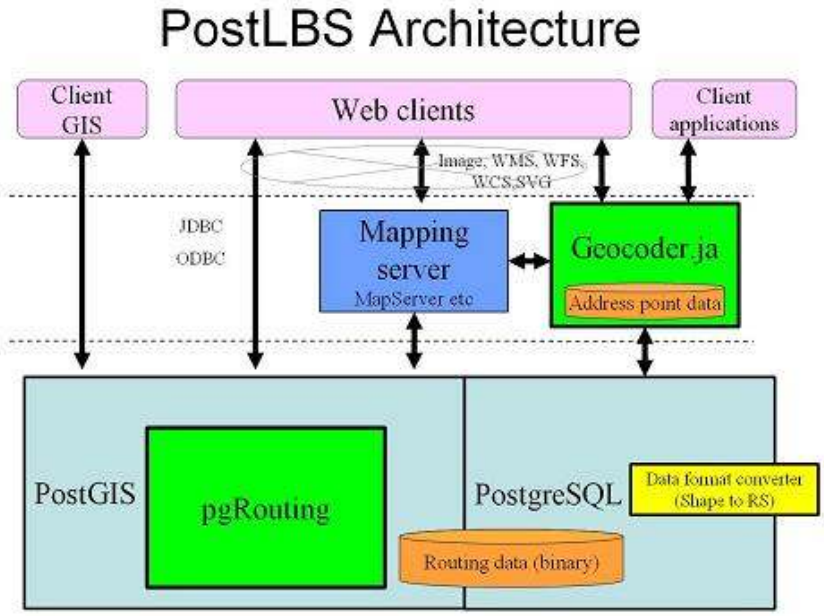
訳者 ： 日本ユニシス株式会社 共通利用技術部 齊藤義雄

1. PostLBS サービス

1.1. PostLBS とは

PostLBS は、商用ソフトウェアと同様の機能性を持つ経路検索(pgRouting) とジオコーディング(geocoder.ja、現在は日本のみ提供)をオープンソース・ソフトウェア(OSS)の位置情報サービス(LBS)のコアツールとして提供することを目指して開発しています。

OSS 及びベンダーGIS クライアントからは、pgRouting と geocoder.ja に容易にアクセス可能です。また、異なるデータ属性構成の取り扱い、他のフォーマットとのデータ交換が容易といった柔軟性を備えています。



1.2. PostLBS の特徴

現在、PostLBS は、*pgRouting* という経路検索モジュールと、*geocoder.ja* という日本の住所を対象としたアドレスマッチングの 2 つのモジュールから成ります。

pgRouting

- PostgreSQL に統合された経路検索エンジン。
- 複数クライアントによる JDBC、ODBC 経由または PostgreSQL API を使用した直接アクセスが可能。 クライアントは、PC およびモバイル機器を対象。
- Well Known Text(WKT)と Well Known Binary(WKB)を OGC データフォーマットとして使用するのと同様に、空間データフォーマットについて PostGIS を使用。 これは既存のオープンなデータ変換の使用を可能にします。
- qGIS と uDIG のようなオープンソースソフトウェアはデータ/属性を変更することができます。そして、即座に経路検索エンジンを介して、これらの変更を反映することができます。

・SQL を通して動的に”cost”パラメータを準備することができますので、値は複数の項目かテーブルから与えることができます。 また、PostgreSQL の他の機能分析およびカレンダー機能のような SQL を使用することでパラメータを準備することができます。
・距離を基準とした到達圏ポリゴンは、データベースの中で異なったテーブルを使用するマーケティング分析において、検索パラメータとして使用することができます。
・pgRouting は GNU General Public License (GPL)の下でリリースされます。
・pgRouting は Camptocamp 社の CartoWeb(http://www.cartoweb.org/)の経路検索モジュールである pgDijkstra(http://www.cartoweb.org/contribs.html)に基づき、Web ベースの地理情報システムで使用することができます。 pgDijkstra は元々、最短経路問題の小規模ネットワーク向けに設計されましたが、簡単に追加機能を統合することができるリッチなフレームワークを提供します。

geocoder.ja

・geocoder.ja は、全てのデータベースから独立しています。
・ジオコーディング･モジュールと住所データをメモリ上で検索しますので、1 秒間あたり約 10 万件の検索を処理することができます。
・Shift-JIS、UTF-8、EUC などの複数のキャラクタコードを扱うことができます。
・数種類の住所フォーマットを扱うことができるテキスト校正機能を提供します。
・geocoder.ja は GNU Lesser Public License (LGPL)の下でリリースされます。

2. pgRouting

2.1. pgRouting プロジェクトの目的

このプロジェクトの主な目標は、経路検索の機能を PostGIS / PostgreSQL に提供することです。pgRouting は PostLBS の一部であり、オープンソース・ソフトウェア(OSS)のコアツールとして位置情報サービス(LBS)を提供します。 ツールは、商用ソフトウェアに見られるものと同類です。

詳細に関しては、プロジェクトのメイン・サイトを見てください。 :http://pgrouting.postlbs.org/

2.2. pgRouting のダウンロード

リリース状況一覧を示します。

pgRouting バージョン	最新バージョンとの相違	リリース形態
1.01	最新版、2007/12/10 リリース	source.tgz
1.0	バグフィックス	source.tgz win32.zip
1.0.0b	コアと拡張機能の分離	source.tgz
	Cmake ビルド工程	
	バグフィックス	
1.0.0a	ラッパー機能に関して、より簡単な名前を持つ付加的な SQL ファイルを作成	source.tgz win32.zip win32 installer
	バグフィックス	
0.9.9	現実の道路ネットワークのための shooting* 最短経路アルゴリズムの追加	
0.9.8	PostgreSQL8.2 をサポート	
	最短経路機能が経路を見つけることが出来なかった場合、結果を空に戻す	
0.9.7	番号ふり直しスキーマが最短経路機能に追加された	
	Directed 最短経路機能が追加された	
	routing_posgis.sql は巡回経路でダイクストラ法を使用するよう修正された	

2.3. pgRouting のインストール

2.3.1. pgRouting バージョンが 1.0 RC1 以上の場合 (CMake を使用)

2.3.1.1. 必要条件

pgRouting のバージョンが 1.0RC 以上

（以前のバージョンは、TSP と Driving Distance 機能無しではコンパイルできません。）

a. pgRouting コア – 最短経路アルゴリズム

- CMake : バージョン 2.3 以上
- C 及び C++コンパイラ
- PostgreSQL (開発者パッケージを含む) : バージョン 8.0 以上
- PostGIS : バージョン 1.0 以上
- Boost Graph Library (BGL) : astar.hpp を含むバージョン 1.33 以上
  http://www.boost.org/libs/graph/doc/index.html 参照。

b. pgRouting 拡張 – 巡回経路検索アルゴリズム(TSP)を含む

- The Genetic Algorithm Utility Library (GAUL)
  http://gaul.sourceforge.net 参照。

c. pgRouting 拡張 – 到達圏検索アルゴリズム(Driving Distance)を含む

- Computational Geometry Algorithms Library (CGAL) : バージョン 3.2 以上
  http://www.cgal.org 参照。

2.3.1.2. Linux のインストール （一般）

ステップ 1: ライブラリのインストール

- PostgreSQL, PostGIS, Proj, GEOS, BGL, GAUL.をコンパイルし、インストールします。
  対応するディレクトリで実行してください:

```
./configure
# make
# make install
```

- BGL をインストール済みでバージョンが 1.33.0 未満の場合には、
  http://www.boost.org/boost/graph/astar_search.hpp から aster.hpp をダウンロードし、
  BOOST_PATH/graph ディレクトリにコピーしてください。

・巡回経路検索アルゴリズム(Traveling Salesperson Problem) の場合 (オプション)

GAUL ライブラリは、enable-slang=no オプションでコンパイルしてください。さもなければ、必ず、slang.h を*/usr/include* にインストールしてください。その他の詳細については、添付の *README* ファイルまたは *INSTALL* ファイルを参照してください。

```
# ./configure --disable-slang
```

・到達圏検索アルゴリズム(Driving Distance) の場合 (オプション)

CGAL はライブラリを生成して使用することができるように、下記のオプションでコンパイルしてください:

```
# ./install_cgal --prefix=/usr --with-boost=n --without-autofind -ni /usr/bin/g++
```

CGAL は特別な処理を必要とします!

CGAL をインストールした後に、CGAL ヘッダーファイルがあるディレクトリの場所を見つける必要があります。 そのディレクトリ内で、config ディレクトリを見つける必要があります。このディレクトリは、1 つ以上のサブディレクトリを含み、プラットホームに依存する情報を持っています。例えば:

```
/
|
+- usr
  |
  +- include
     |
     +- CGAL
        |
        +- config
           |
           +- i686_Linux-2.6_g++4.1.1
           |  |
           |  +- CGAL
           |     |
           |     +- compiler_config.h     <<< これが必要なファイルです。
           |
           +- msvc7
           |  |
           |  +- CGAL
           |     |
           |     +- compiler_config.h
           |
          etc...
```

プラットホームに対応する compiler_config.h を選択し、symlink するか、またはそれをコピーしてください。例えば:

```
# ln -s /usr/include/CGAL/config/i686_Linux-2.6_g++4.1.1/compiler_config.h /
usr/include/CGAL/compiler_config.h
```

ステップ 2: pgRouting ライブラリのコンパイル

1. pgRouting ソースの入手
2. pgRouting ディレクトリに移動
3. pgRouting のコンパイルおよびインストール

・pgRouting コア の場合 (必須)

```
cmake .
make
make install
```

▪pgRouting 拡張の場合 （オプション）

```
# Add Traveling Salesperson functionality:   -DWITH_TSP=ON
# Add Driving Distance functionality        :   -DWITH_DD=ON
cmake -DWITH_TSP=ON -DWITH_DD=ON .
make
make install
```

ステップ 3: 経路検索データベースの作成、PostGIS および pgRouting 機能のロード

1. 経路探索データベースの作成と PostGIS のロード

```
createdb -U postgres -E UNICODE routing
createlang -U postgres plpgsql routing

psql -U postgres -f /path/to/postgis/lwpostgis.sql routing
psql -U postgres -f /path/to/postgis/spatial_ref_sys.sql routing
```

2. pgRouting コア機能の追加 （必須）

```
psql -U postgres -f /usr/share/postlbs/routing_core.sql routing
psql -U postgres -f /usr/share/postlbs/routing_core_wrappers.sql routing
```

3. pgRouting 拡張機能の追加 （オプション）

```
# With TSP
psql -U postgres -f /usr/share/postlbs/routing_tsp.sql routing
psql -U postgres -f /usr/share/postlbs/routing_tsp_wrappers.sql routing

# With Driving Distance
psql -U postgres -f /usr/share/postlbs/routing_dd.sql routing
psql -U postgres -f /usr/share/postlbs/routing_dd_wrappers.sql routing
```

CentOS 5

（訳者注：原文にリンク先ページなし）

Ubuntu7.04

（訳者注：翻訳の対象外）

Ubuntu 7.10 Beta

（訳者注：翻訳の対象外）

2.3.1.3. Windows のインストール

環境

・MinGW 5.0.3 (E:/Build/mingw にインストールしてください。)

・MSYS-1.0.11 (install to E:/Build/msys にインストールしてください。)

http://www.baldanders.info/spiegel/remark/archives/000209.shtml (日本語)

・pthread-win32-2.7.0

ftp://sources.redhat.com/pub/pthreads-win32/から prebuild-dll-2-7-0-release をダウンロードしてください。

```
cp /include/*.h /mingw/include
cp /lib/libpthreadGC2.a /mingw/lib/libpthread.a
```

CMake

最新のバージョン(2.4.7)を使用してください。

```
#./configure --prefix=E:/Build/msys/1.0/local
#make
#make install
```

または、Windows インストーラを使用できます。

Boost

v2 の問題のため前のバージョン(1.33.1)を使用してください。

```
#bjam -sTOOLS=mingw "-sBUILD=release <runtime-link>static <threading>multi
  <native-wchar_t>on" --prefix=/e/Build/msys/1.0/local install
#mv /usr/local/include/boost-1_33_1/boost /usr/local/include/boost
#rmdir /usr/local/include/boost-1_33_1
```

もし、CGAL ライブラリ(到達圏検索アルゴリズム機能)を必要としない場合は、pgRouting は Boost ヘッダーのみ使用します。その場合、Boost ヘッダー(バージョン 1.33.1 またはそれ以上)をダウンロードして抜き出してください。

Gaul

最新のバージョン(0.1849-0)を使用してください。

```
#./configure --enable-slang=no
edit /util/Makefile
        DEFS = -DHAVE_CONFIG_H
         -> DEFS = -DHAVE_CONFIG_H -DBUILDING_DLL
edit /src/Makefile
        DEFS = -DHAVE_CONFIG_H
         -> DEFS = -DHAVE_CONFIG_H -DBUILDING_DLL
        INCLUDES = -I../util/ -I../../util/
         -> INCLUDES = -I../util -I../../util
edit /tests/Makefile
        DEFS = -DHAVE_CONFIG_H
         -> DEFS = -DHAVE_CONFIG_H -DBUILDING_DLL
        INCLUDES = -I../util/ -I../../util/ -I../src/ -I../../src/ -I/usr/include/slang/
         -> INCLUDES = -I../util -I../../util -I../src -I../../src -I/usr/include/slang/
#make
#make install
```

CGAL

最新のバージョン(3.2.1)を使用して下さい。

```
#./install_cgal --prefix=/usr/local/cgal --with-BOOST
--BOOST_INCL_DIR=/usr/local/include --BOOST_LIB_DIR=/usr/local/lib
--without-autofind -ni /mingw/bin/g++
#cp
/usr/local/cgal/include/CGAL/config/i686_MINGW32NT-5.1_g++-3.4.2/CGAL/c
ompiler_config.h /usr/local/cgal/include/CGAL/compiler_config.h
#cp /usr/local/cgal/lib/i686_MINGW32NT-5.1_g++-3.4.2/libCGAL.a
/usr/local/cgal/lib/libCGAL.a
#cp /usr/local/cgal/lib/i686_MINGW32NT-5.1_g++-3.4.2/libCGAL.so
/usr/local/cgal/lib/libCGAL.so
```

pgRouting

```
#cmake -G"MSYS Makefiles" -DWITH_TSP=ON -DWITH_DD=ON .
```

CMakeCache.txt の編集

Boost_INCLUDE_DIR:PATH=Boost_INCLUDE_DIR-NOTFOUND
-> Boost_INCLUDE_DIR:PATH=E:/Build/msys/1.0/local/include

CGAL_INCLUDE_DIR:PATH=CGAL_INCLUDE_DIR-NOTFOUND
-> CGAL_INCLUDE_DIR:PATH=E:/Build/msys/1.0/local/cgal/include

CGAL_LIBRARIES:FILEPATH=CGAL_LIBRARIES-NOTFOUND
-> CGAL_LIBRARIES:FILEPATH=E:/Build/msys/1.0/local/cgal/lib

GAUL_LIBRARIES:FILEPATH=GAUL_LIBRARIES-NOTFOUND
-> GAUL_LIBRARIES:FILEPATH=E:/Build/msys/1.0/local/lib

core¥src¥CMakeFiles¥routing.dir¥flags.make の編集

<drive>（例. "C"）および<my path> をシステム環境に合わせてください!

```
C_FLAGS = -Drouting_EXPORTS -O2 -g  ¥
        -I/<drive>/<my path>/msys/1.0/local/src/pgrouting/pgrouting/. ¥
        -I/<drive>/<my path>/msys/1.0/local/src/pgrouting/pgrouting/core ¥
        -I/<drive>/<my path>/msys/1.0/local/src/pgrouting/pgrouting/core/src ¥
        -I/<drive>/<my path>/msys/1.0/local/src/pgrouting/pgrouting/extra ¥
        -I/<drive>/<my path>/msys/1.0/local/src/pgrouting/pgrouting/extra/tsp ¥
        -I/<drive>/<my path>/msys/1.0/local/src/pgrouting/pgrouting/extra/tsp/src ¥
        -I/<drive>/<my
path>/msys/1.0/local/src/pgrouting/pgrouting/extra/driving_distance ¥
        -I/<drive>/<my
path>/msys/1.0/local/src/pgrouting/pgrouting/extra/driving_distance/src ¥
        -I/boost ¥
        -I/<drive>/<my path>/msys/1.0/local/include ¥
        -IC:/PROGRA~1/PostgreSQL/8.2/include/server ¥
        -IC:/PROGRA~1/PostgreSQL/8.2/include/server/port/win32
CXX_FLAGS = -Drouting_EXPORTS -O2 -g
          -I/<drive>/<my path>/msys/1.0/local/src/pgrouting/pgrouting/. ¥
          -I/<drive>/<my path>/msys/1.0/local/src/pgrouting/pgrouting/core ¥
          -I/<drive>/<my path>/msys/1.0/local/src/pgrouting/pgrouting/core/src ¥
          -I/<drive>/<my path>/msys/1.0/local/src/pgrouting/pgrouting/extra ¥
          -I/<drive>/<my path>/msys/1.0/local/src/pgrouting/pgrouting/extra/tsp ¥
          -I/<drive>/<my path>/msys/1.0/local/src/pgrouting/pgrouting/extra/tsp/src ¥
          -I/<drive>/<my
path>/msys/1.0/local/src/pgrouting/pgrouting/extra/driving_distance ¥
          -I/<drive>/<my
path>/msys/1.0/local/src/pgrouting/pgrouting/extra/driving_distance/src ¥
          -I/boost ¥
          -I/<drive>/<my path>/msys/1.0/local/include ¥
          -IC:/PROGRA~1/PostgreSQL/8.2/include/server ¥
          -IC:/PROGRA~1/PostgreSQL/8.2/include/server/port/win32
```

(訳者注:“￥”は、原文では”＼”(半角)で記載されています。次ページ以降、数箇所あります。)

core¥src¥CMakeFiles¥routing.dir¥build.make の編集

<drive> (例. "C") および<my path> をシステム環境に合わせてください!

```
routing_EXTERNAL_OBJECTS = ¥
```

次のように最終行を編集してください。

```
cd /<drive>/<my path>/pgrouting/core/src && /<drive>/<my path>/mingw/bin/g++
.exe $(routing_OBJECTS) $(routing_EXTERNAL_OBJECTS) ¥
                -L/<drive>/<my path>/msys/1.0/local/lib ¥
                -LC:/PROGRA~1/PostgreSQL/8.2/lib ¥
                -lpostgres -shared -o ../../lib/librouting.dll ¥
                -Wl,--out-implib,../../lib/librouting.dll.a ¥
                -Wl,--major-image-version,0,--minor-image-version,0
```

extra¥tsp¥src¥CMakeFiles¥routing_tsp.dir¥flags.make の編集

```
C_FLAGS = -Drouting_tsp_EXPORTS -g -O2 -IE:/Build/msys/1.0/local/include
-I/E/Build/msys/1.0/local/src/pgrouting/pgrouting/core/src
-IC:/PROGRA~1/PostgreSQL/8.2/include/server
-IC:/PROGRA~1/PostgreSQL/8.2/include/server/port/win32
CXX_FLAGS = -Drouting_tsp_EXPORTS -g -O2 -IE:/Build/msys/1.0/local/include
-I/E/Build/msys/1.0/local/src/pgrouting/pgrouting/core/src
-IC:/PROGRA~1/PostgreSQL/8.2/include/server
-IC:/PROGRA~1/PostgreSQL/8.2/include/server/port/win32 -DBUILDING_DLL
```

extra¥tsp¥src¥CMakeFiles¥routing_tsp.dir¥build.make の編集

```
routing_tsp_EXTERNAL_OBJECTS = ¥
```

次のように最終ラインを編集して下さい。

```
cd /E/Build/msys/1.0/local/src/pgrouting/pgrouting/extra/tsp/src &&
/e/Build/mingw/bin/g++.exe $(routing_tsp_OBJECTS)
$(routing_tsp_EXTERNAL_OBJECTS) -LE:/Build/msys/1.0/local/lib
-LC:/PROGRA~1/PostgreSQL/8.2/lib -lpostgres -lgaul -lgaul_util -lm -shared
-o ../../../lib/librouting_tsp.dll -Wl,--out-implib,../../../lib/librouting_tsp.dll.a
-Wl,--major-image-version,0,--minor-image-version,0 -Wl,--export-all-symbols
```

extra¥driving_distance¥src¥CMakeFiles¥routing_dd.dir¥flags.make の編集

```
C_FLAGS = -Drouting_dd_EXPORTS -g -O2 -IE:/Build/msys/1.0/local/include
-I/E/Build/msys/1.0/local/src/pgrouting/pgrouting/core/src
-I/E/Build/msys/1.0/local/cgal/include
-IC:/PROGRA~1/PostgreSQL/8.2/include/server
-IC:/PROGRA~1/PostgreSQL/8.2/include/server/port/win32
CXX_FLAGS = -Drouting_dd_EXPORTS -g -O2 -IE:/Build/msys/1.0/local/include
-I/E/Build/msys/1.0/local/src/pgrouting/pgrouting/core/src
-I/E/Build/msys/1.0/local/cgal/include
-IC:/PROGRA~1/PostgreSQL/8.2/include/server
-IC:/PROGRA~1/PostgreSQL/8.2/include/server/port/win32
```

extra¥driving_distance¥src¥CMakeFiles¥routing_dd.dir¥build.make の編集

```
routing_tsp_EXTERNAL_OBJECTS = ¥
```

次のように最終ラインを編集して下さい。

```
cd /E/Build/msys/1.0/local/src/pgrouting/pgrouting/extra/driving_distance/src
&& /e/Build/mingw/bin/g++.exe $(routing_dd_OBJECTS)
$(routing_dd_EXTERNAL_OBJECTS) -LE:/Build/msys/1.0/local/lib
-LE:/Build/msys/1.0/local/cgal/lib -LC:/PROGRA~1/PostgreSQL/8.2/lib
-lCGAL -lpostgres -shared -o ../../../lib/librouting_dd.dll
-Wl,--out-implib,../../../lib/librouting_dd.dll.a
-Wl,--major-image-version,0,--minor-image-version,0
```

pgRouting のビルド

```
#make
```

2.3.2. pgRouting バージョンが 1.0 RC1 未満の場合

2.3.2.1. 必要条件

- C 及び C++コンパイラ
- PostgreSQL ： バージョン 8.0 以上
- PostGIS ： バージョン 1.0 以上
- Boost Graph Library (BGL) ： astar.hpp を含むバージョン 1.33 以上
  http://www.boost.org/libs/graph/doc/index.html 参照。
- Genetic Algorithm Utility Library (GAUL)

http://gaul.sourceforge.net 参照。

・Computational Geometry Algorithms Library (CGAL) : バージョン 3.2 以上
http://www.cgal.org 参照。

2.3.2.2. Linux のインストール (一般)

ステップ 1: ライブラリのインストール

・PostgreSQL, PostGIS, Proj, GEOS, BGL, GAUL のコンパイルし、インストールします。
対応するディレクトリで実行してください:

```
./configure
# make
# make install
```

BGL インストール済みでバージョンが 1.33.0 未満の場合には、
http://www.boost.org/boost/graph/astar_search.hpp から aster.hpp をダウンロードし、
BOOST_PATH/graph ディレクトリにコピーしてください。

GAUL ライブラリは、enable-slang=no オプションでコンパイルしてください。さもなければ、必ず、slang.h を/usr/include にインストールしてください。その他の詳細については、添付の README ファイルまたは INSTALL ファイルを参照してください。

```
# ./configure --disable-slang
```

CGAL ではライブラリを生成して使用することができるように、下記のオプションでコンパイルしてください:

```
# ./install_cgal --prefix=/usr --with-boost=n --without-autofind -ni /usr/bin/g++
```

ステップ 2 : pgRouting ライブラリのコンパイル

同じコンピュータ上で 2 つのバージョンの PostgreSQL を運用している場合(例えば、7.4 と 8.1 の PostgreSQL)、対象の PostgreSQL の pg_config が PATH 環境にあることを確認してください。
pgRouting をビルドするためには、最初に"configure"を実行すれば通常、充分なはずです。もし、BOOST、GAUL と CGAL ライブラリがデフォルトパスにない場合には、それらの位置を configure に指示する必要があります。"./configure --help"をタイプすることで、configure オプションが表示されます。

```
# ./configure --with-cgal=/usr/local/CGAL --with-gaul=/usr/local
# make install
```

ステップ 3 ： データベースのセットアップ

（例 ： サンプル・アプリケーション）

ネットワーク・データベースを作成するためには、次のようにしてください。

```
# PGSQL_PATH/bin/createdb -E UTF-8 routing
# PGSQL_PATH/bin/createlang plpgsql routing
# PGSQL_PATH/bin/psql -f PGSQL_PATH/share/contrib/lwpostgis.sql routing
# PGSQL_PATH/bin/psql -f PGSQL_PATH/share/contrib/spatial_ref_sys.sql routing
# PGSQL_PATH/bin/shp2pgsql -D kanagawa.shp kanagawa > kanagawa.sql
# PGSQL_PATH/bin/psql -f kanagawa.sql routing
```

実行速度を改善するために、kanagawa テーブルでいくつかのインデックスリストを作成してください。

```
# create index gid_idx on kanagawa(gid);
# create index source_idx on kanagawa(source);
# create index target_idx on kanagawa(target);
# create index geom_idx on kanagawa using GIST (the_geom GIST_GEOMETRY_OPS);
```

pgRouting 機能をインストールするために sql ファイル”routing.sql”を実行してください。

```
# PGSQL_PATH/bin/psql -f routing.sql routing
```

PostGIS の入力と操作機能を作成する”routing_postgis.sql”を実行してください。

```
# PGSQL_PATH/bin/psql -f routing_postgis.sql routing
```

2.3.2.3. Windows のバイナリを作成する方法

注釈 ： この説明は、pgRoutingのWindowsバイナリをビルドするためのチェックリストです。
pgRoutingの初期のバージョンが対象のため、情報が古い可能性があります。

環境

・MinGW 5.0.3

・MSYS-1.0.11

http://www.baldanders.info/spiegel/remark/archives/000209.shtml (日本語)

・pthread-win32-2.7.0

ftp://sources.redhat.com/pub/pthreads-win32/ から prebuild-dll-2-7-0-release をダウンロードしてください。

```
cp /include/*.h /mingw/include
cp /lib/libpthreadGC2.a /mingw/lib/libpthread.a
```

Boost

最新のバージョン(1.33.1)を使用してください。

```
#bjam -sTOOLS=mingw "-sBUILD=release <runtime-link>static <threading>multi
  <native-wchar_t>on" --prefix=/e/Build/msys/1.0/local install
#mv /usr/local/include/boost-1_33_1/boost /usr/local/include/boost
#rmdir /usr/local/include/boost-1_33_1
```

Gaul

最新のバージョン(0.1849－0)を使用してください。

```
#./configure --enable-slang=no
edit /util/Makefile
        DEFS = -DHAVE_CONFIG_H
          -> DEFS = -DHAVE_CONFIG_H -DBUILDING_DLL
edit /src/Makefile
        DEFS = -DHAVE_CONFIG_H
          -> DEFS = -DHAVE_CONFIG_H -DBUILDING_DLL
        INCLUDES = -I../util/ -I../../util/
          -> INCLUDES = -I../util -I../../util
edit /tests/Makefile
        DEFS = -DHAVE_CONFIG_H
          -> DEFS = -DHAVE_CONFIG_H -DBUILDING_DLL
        INCLUDES = -I../util/ -I../../util/ -I../src/ -I../../src/ -I/usr/include/slang/
          -> INCLUDES = -I../util -I../../util -I../src -I../../src -I/usr/include/slang/
#make
#make install
```

CGAL

最新のバージョン(3.2.1)を使用して下さい。

```
#./install_cgal --prefix=/usr/local/cgal --with-BOOST
--BOOST_INCL_DIR=/usr/local/include --BOOST_LIB_DIR=/usr/local/lib
--without-autofind -ni /mingw/bin/g++
#cp
/usr/local/cgal/include/CGAL/config/i686_MINGW32NT-5.1_g++-3.4.2/CGAL/c
ompiler_config.h /usr/local/cgal/include/CGAL/compiler_config.h
#cp /usr/local/cgal/lib/i686_MINGW32NT-5.1_g++-3.4.2/libCGAL.a
/usr/local/cgal/lib/libCGAL.a
#cp /usr/local/cgal/lib/i686_MINGW32NT-5.1_g++-3.4.2/libCGAL.so
/usr/local/cgal/lib/libCGAL.so
```

pgRouting

```
#./configure --with-boost=/usr/local --with-cgal=/usr/local/cgal
--with-gaul=/usr/local
```

sys/time.h の編集

```
comment out timezone struct
comment out gettimeofday function
```

Makefile の編集

```
BOOST_PATH=/usr/local/include/boost/graph
-> BOOST_PATH=/usr/local/include
GAUL_PATH =/usr/local/include/gaul
-> GAUL_PATH =/usr/local/include
TSP_LIBS= -L$(GAUL_LIB) -lgaul_util -lgaul
-> TSP_LIBS= -L$(GAUL_LIB) -lgaul -lgaul_util -lm
$(CXX) $(CPPFLAGS) $(EXTRA_FLAGS) -I$(GAUL_PATH) -c tsp_solver.cpp
-> $(CXX) $(CPPFLAGS) $(EXTRA_FLAGS) -I$(GAUL_PATH) -DBUILDING_DLL
-c tsp_solver.cpp
$(CXX) $(CPPFLAGS) $(EXTRA_FLAGS) -I$(CGAL_PATH) -c alpha_drivedist.cpp
-> $(CXX) $(CPPFLAGS) $(EXTRA_FLAGS) -I$(BOOST_PATH) -I$(CGAL_PATH)
-c alpha_drivedist.cpp
```

```
#make
#make install
```

2.4. サポート

2.4.1. コミュニティ・サポート

・技術資料(http://pgrouting.postlbs.org/wiki/pgRoutingDocs)を参照する。

・開発者フォーラム(http://pgrouting.postlbs.org/discussion)で質問する。

・メーリングリスト ： http://lists.postlbs.org/mailman/listinfo

2.4.2. 有償サポート

専門家のサポート、開発またはコンサルタント業務を必要とするユーザのために、株式会社オークニー (PostLBS プロジェクトの開発元)が様々なサービスを提供しています。 コンタクト先と費用についてはサポート情報ページを参照してください ： http://www.orkney.co.jp

2.4.3. PostLBS の関係リンク先

・PostgreSQL ： http://www.postgresql.org

・PostGIS ： http://www.postgis.org

2.4.4. コンタクト先

株式会社オークニー 本社

横浜市西区みなとみらい3-6-3 MMパークビル13階

〒220-0012 Tel 81-45-228-3320 Fax 81-45-228-3321

http://www.orkney.co.jp/

2.5. pgRouting の機能

pgRouting は、Camptocamp によって開発された pgDijkstra をさらに発展させたもので、オークニーによって開発され、保守されています。

2.5.1. Shortest Path Dijkstra

ダイクストラ法に基づく経路検索アルゴリズムで、経験則なしの経路検索です。

最短経路機能には、次の宣言文が必要です:

```
CREATE OR REPLACE FUNCTION shortest_path(sql text, source_id integer,
                        target_id integer, directed boolean, has_reverse_cost boolean)
        RETURNS SETOF path_result
```

引数 :

sql : SQL 検索すると次の列を持つ行セットを返します:

```
SELECT id, source, target, cost FROM edge_table
```

- id : エッジの識別子 [int4]
- source : 始点ノードの識別子 [int4]
- target : 終点ノードの識別子 [int4]
- cost : 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のを防ぎます)。 [float8]
- reverse_cost(オプション) : エッジの反対方向のためのコスト。
  有向グラフで、has_reverse_cost パラメータが真の時のみ使用される(負のコストについては前述の通りです)。
- edge_table : エッジ、始終点ノード、コストが格納されているテーブル。

source_id : 始点のノード id [int4]

target_id : 終点のノード id [int4]

directed : グラフが方向を指示されている(有向グラフ)場合は真。

has_reverse_cost : 真の場合、SQL で生成している行セットの列の reverse_cost は逆方向へのエッジにかかる重みに使用されます。

path_result : 関数は、1 セットの行を返します。それぞれの交差しているエッジ当たり 1 行、および終点の頂点を含む追加行が 1 行あります。各行の列は:

- vertex_id : 各エッジの始点の識別子。終点経路のノード識別子を含む最後のエッジの後ろにもう 1 行あります。
- edge_id : 交差したエッジの識別子。

・cost ： 現在のエッジに関連づけられたコストで、最終エッジの後の行では 0 です。したがって、経路の合計コストは cost 列を合計したものです。

例:

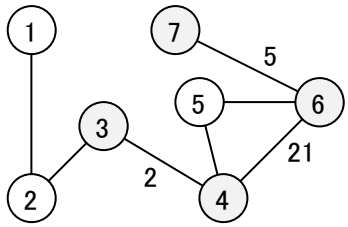
```
SELECT * FROM shortest_path ('SELECT gid AS id, source::int4,
            target::int4, length::double precision AS cost,FROM dourol', 3, 7, false, false);
```

解説 ： ノード 3 からノード 7 への最短経路を求める SQL 文で、エッジ id として douro1 テーブルの gid、コストとして douro1 テーブルの長さ、無向グラフ(directed : false)、逆向きコスト無し(has_reserve_cost : false)が指定されています。下表は、douro1(edge_table)です。

id	source	target	length
	1	2	
	2	3	
2	3	4	0.000763954363701041
	4	5	
21	4	6	0.00150254971056274
	5	6	
5	6	7	0.000417442425988342

下表は、検索結果を示す。合計コストは cost 列を合計したものです。なお、グラフは一例です。

```
vertex_id | edge_id | cost
-------- +------- +-------------------------
        3 |       2 | 0.000763954363701041
        4 |      21 | 0.00150254971056274
        6 |       5 | 0.000417442425988342
        7 |      -1 | 0
(4 rows)
```




```
SELECT * FROM shortest_path('SELECT gid AS id, source::int4,
            target::int4, length::double precision AS cost,length::double precision
      AS reverse_cost FROM dourol', 3, 7, true, true);
```

解説 ： ノード 3 からノード 7 への最短経路を求める SQL 文で、エッジ id として douro1 テーブルの gid、コストは douro1 テーブルの長さ、逆向きコストも douro1 テーブルの長さ、有向グラフ(directed : true)、逆向きコスト有り(has_reserve_cost : true)が指定されています。

2.5.2. Shortest Path A*

A*法に基づく最短経路アルゴリズムで、大規模データセットのための経験則ありの経路検索です。

shortest_path_astar 機能には、次の宣言文が必要です:

```
CREATE OR REPLACE FUNCTION shortest_path_astar(sql text, source_id integer,
                target_id integer, directed boolean, has_reverse_cost boolean)
        RETURNS SETOF path_result
```

引数 :

sql : SQL 検索すると次の列を持つ行セットを返します:

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

- id : エッジの識別子 [int4]
- source : 始点ノードの識別子 [int4]
- target : 終点ノードの識別子 [int4]
- cost : エッジにかかるコスト(負のコストは、エッジがグラフに挿入されるのを防ぎます)。[float8]
- 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ためのコスト。有向グラフで、has_reverse_cost パラメータが真の時のみ使用される(負のコストについては前述の通りです)。
- edge_table : エッジ、始終点ノード、コストが格納されているテーブル。

source_id : 始点のノード id [int4]

target_id : 終点のノード id [int4]

directed : グラフが方向を指示されている(有向グラフ)場合は真。

has_reverse_cost : 真の場合、SQL で生成している行セットの列の reverse_cost は逆方向へのエッジにかかる重みに使用されます。

path_result : 関数は、1 セットの行を返します。それぞれの交差しているエッジ当たり 1 行、および終点の頂点を含む追加行が 1 行あります。各行の列は:

- vertex_id : 各エッジの始点の識別子。終点経路のノード識別子を含む最後のエッジの後ろにもう 1 行あります。

・edge_id ： 交差したエッジの識別子。

・cost ： 現在のエッジに関連づけられたコストで、最終エッジの後の行では 0 です。したがって、経路の合計コストは cost 列を合計したものです。

例:

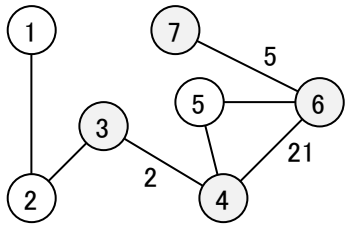
```
SELECT * FROM shortest_path_astar('SELECT gid AS id, source::int4,
        target::int4, length::double precision AS cost, x1, y1, x2, y2
    FROM douro1', 3, 7, false, false);
```

解説 ： ノード 3 からノード 7 への最短経路を求める SQL 文で、エッジ id として douro1 テーブルの gid、コストとして douro1 テーブルの長さ、エッジの始終点の xy 座標、無向グラフ(directed:false)、逆向きコスト無し(has_reserve_cost:false)が指定されています。下表は、douro1(edge_table)です。

id	source	target	x1	y1	x2	y2	length
	1	2					
	2	3					
2	3	4					0.000763954363701041
	4	5					
21	4	6					0.00150254971056274
	5	6					
5	6	7					0.000417442425988342

下表は、検索結果を示す。合計コストは cost 列を合計したものです。なお、グラフは一例です。

```
vertex_id | edge_id | cost
-------- +------- +--------------------------
        3 |       2 | 0.000763954363701041
        4 |      21 | 0.00150254971056274
        6 |       5 | 0.000417442425988342
        7 |      -1 | 0
(4 rows)
```




```
SELECT * FROM shortest_path_astar('SELECT gid AS id, source::int4,
        target::int4, length::double precision AS cost,length::double precision
    AS reverse_cost, x1, y1, x2, y2 FROM douro1', 3, 7, true, true);
```

解説 ： ノード 3 からノード 7 への最短経路を求める SQL 文で、エッジ id として douro1 テーブルの

gid、コストは douro1 テーブルの長さ、逆向きコストも douro1 テーブルの長さ、エッジの始終点の xy 座標、有向グラフ(directed：true)、逆向きコスト有り(has_reserve_cost：true)が指定されています。

2.5.3. Shortest Path Shooting Star

指定方向外進行禁止に対応した経路検索アルゴリズムです。

Shortest Path Shooting Star 機能には、次の宣言文が必要です：

```
CREATE OR REPLACE FUNCTION shortest_path_shooting_star(sql text, source_id integer,
        target_id integer, directed boolean, has_reverse_cost boolean)
        RETURNS SETOF path_result
```

path_result ：

```
CREATE TYPE path_result AS (vertex_id integer, edge_id integer, cost float8);
```

引数 ：

sql ： SQL 検索すると次の列を持つ行セットを返します：

```
SELECT id, source, target, cost, x1, y1, x2, y2, rule, to_cost FROM edges
```

- id ： エッジの識別子 [int4]
- source ： 始点ノードの識別子 [int4]
- target ： 終点ノードの識別子 [int4]
- cost ： エッジにかかるコスト(負のコストは、エッジがグラフに挿入されるのを防ぎます)。 [float8]
- reverse_cost(オプション) ： エッジの反対方向のためのコスト。
  有向グラフで、has_reverse_cost パラメータが真の時のみ使用されます(負のコストについては前述の通りです)。

- x1 ： エッジの始点の x 座標 [倍精度]
- y1 ： エッジの始点の y 座標 [倍精度]
- x2 ： エッジの終点の x 座標 [倍精度]
- y2 ： エッジの終点の y 座標 [倍精度]

- rule ： 指定方向外進行禁止の規則を記述するエッジ id 毎のコンマで区切られたリストを持つストリング (これらのエッジに沿って行くならば、to_cost で決められたコストで通り抜けることができます)。
- to_cost ： 制限のある通路のコスト(指定方向外進行禁止の場合には非常に高くなり

ますが、信号機の場合にはエッジのコストに匹敵する場合があります)。

・edges ： エッジ、始終点ノード、コストが格納されているテーブル。

source_id ： 始点のノード id [int4]

target_id ： 終点のノード id [int4]

directed ： グラフが方向を指示されている(有向グラフ)場合は真。

has_reverse_cost ： 真の場合、SQL で生成している行セットの列の reverse_cost は逆方向へのエッジにかかる重みに使用されます。

path_result ： 関数は、1 セットの行を返します。それぞれの交差しているエッジ当たり 1 行、および終点の頂点を含む追加行が 1 行あります。各行の列は：

・vertex_id ： 各エッジの始点の識別子。終点経路のノード識別子を含む最後のエッジの後ろにもう 1 行あります。

・edge_id ： 交差したエッジの識別子。

・cost ： 現在のエッジに関連づけられたコストで、最終エッジの後の行では 0 です。したがって、経路の合計コストは cost 列を合計したものです。

例：

Shooting*のアルゴリズムは、エッジからエッジに経路を計算します（ノードからノードではありません）。列 vertex_id は、列 edge_id のエッジの始点ノードを含みます。

指定方向外進行禁止の記述：

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

エッジ 14 からエッジ 12 に行くためのコストは 1000 であることを意味します。そして、

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

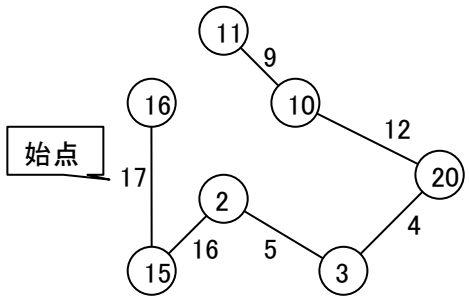
エッジ 14 からエッジ 4 を経由してエッジ 12 に行くためのコストは 1000 であることを意味します。

shooting*アルゴリズムを使用する経路検索：

```
SELECT * FROM shortest_path_shooting_star('SELECT id, source, target, cost,
            x1, y1, x2, y2, rule, to_cost FROM edges', 17, 9, true, false);
```

解説 ： エッジ 17 からエッジ 9 への最短経路を求める SQL 文で、エッジ id として edges テーブルの id、コストは edges テーブルの cost、エッジの始終点の xy 座標、有向グラフ(directed:true)、逆向きコスト無し(has_reserve_cost:false)が指定されています。下表は、検索結果を示す。合計コストは cost 列を合計したものです。なお、グラフは一例です。

```
vertex_id  | edge_id   | cost
-----------+-----------+------
        16 |        17 |    1
        15 |        16 |    1
         2 |         5 |    1
         3 |         4 |    1
        20 |        12 |    2
        10 |         9 |    2
(6 rows)
```



2.5.4. Travelling Sales Person (TSP)

巡回経路検索アルゴリズムです。

TSP 機能には、次の宣言文が必要です:

```
CREATE OR REPLACE FUNCTION tsp(sql text, ids varchar, source_id integer)
        RETURNS SETOF path_result
```

引数 ：

sql ： SQL 検索すると次の列を持つ行セットを返します:

```
SELECT source_id, x, y FROM edge_table
```

- source_id ： ノードの識別子 [int4]
- x ： ノードの x 座標
- y ： ノードの y 座標
- edge_table ： エッジ、始終点ノード、コストが格納されているテーブル。

ids ： カンマで区切られたノード id [int4]の文字列。

source_id ： 始点のノード id [int4]

path_result ： 関数は、1 セットの行を返します。それぞれの交差しているエッジ当たり 1 行、および終点の頂点を含む追加行が 1 行あります。各行の列は:

- vertex_id ： 各エッジの始点の識別子。終点経路のノード識別子を含む最後のエッジ

の後ろにもう 1 行あります。

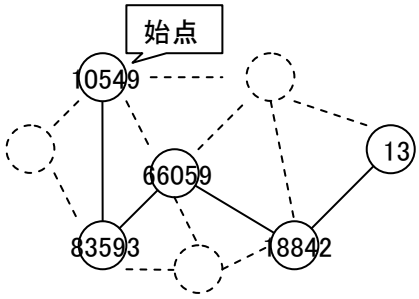
- edge_id ： 未使用、通常はゼロ。
- cost ： 未使用、通常はゼロ。

例:

```
SELECT * FROM tsp('SELECT distinct source AS source_id,
                          x1::double precision AS x,
                          y1::double precision AS y FROM dourol
          WHERE source IN (83593,66059,10549,18842,13)',
                              '83593,66059,10549,18842,13', 10549);
```

解説 ： id:10549 を始点に id:83593,66059,10549,18842,13 を巡回探索します。下表は、その結果を示します。なお、グラフは一例です。

```
vertex_id    | edge_id   | cost
-----------+---------+------
      10549 |         0 |    0
      83593 |         0 |    0
      66059 |         0 |    0
      18842 |         0 |    0
         13 |         0 |    0
(5 rows)
```



あとがき ： ノード id 列は、最短経路計算に使用することができます。

2.5.5. Driving Distance Calculation

走行距離を基準にした到達圏検索アルゴリズムです。

driving_distance 機能には、次の宣言文が必要です:

```
CREATE OR REPLACE FUNCTION driving_distance(sql text, source_id integer, distance float8)
        RETURNS SETOF path_result
```

引数 :

sql ： SQL 検索すると次の列を持つ行セットを返します:

```
SELECT id, source, target, cost FROM edge_table
```

- id ： エッジの識別子 [int4]

・source ： 始点ノードの識別子 [int4]
・target ： 終点ノードの識別子 [int4]
・cost ： 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のを防ぎます)。
[float8]
・edge_table ： エッジ、始終点ノード、コストが格納されているテーブル。

source_id ： 始点のノード id [int4]
distance ： 始点からの距離[float8]

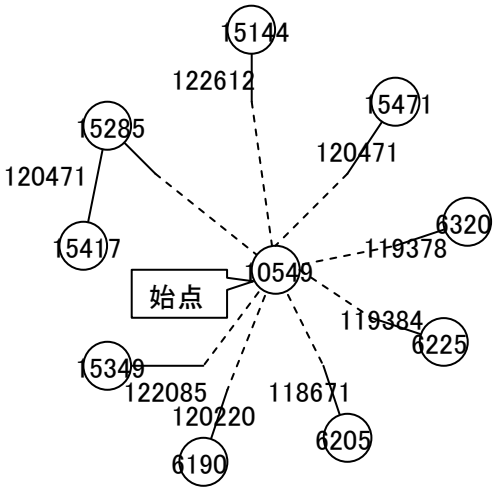
path_result ： 関数は、1 セットの行を返します。それぞれの交差しているエッジ当たり 1 行、および終点の頂点を含む追加行が 1 行あります。各行の列は：
・vertex_id ： 各エッジの始点の識別子。終点経路のノード識別子を含む最後のエッジの後ろにもう 1 行あります。
・edge_id ： 交差したエッジの識別子。
・cost ： 現在のエッジに関連づけられたコストで、最終エッジの後の行では 0 です。したがって、経路の合計コストは cost 列を合計したものです。

例：

```
SELECT * FROM driving_distance('SELECT gid AS id,source,target,
        length::double precision AS cost FROM dourol',10549,0.01);
```

解説 ： 始点 10549 を中心に 100m 範囲の到達圏を求める SQL 文で、エッジ id として douro1 テーブルの gid、コストとして douro1 テーブルの長さが指定されています。下表は、douro1(edge_table)です。なお、グラフは一例です。

```
vertex_id    | edge_id   |       cost
-----------+---------+----------------
        6190 |   120220 | 0.00967666852
        6205 |   118671 | 0.00961557335
        6225 |   119384 | 0.00965668162
        6320 |   119378 | 0.00959826176
        ...
        ...
        ...
       15144 |   122612 | 0.00973386526
       15285 |   120471 | 0.00912965866
       15349 |   122085 | 0.00944814966
       15417 |   120471 | 0.00942316736
       15483 |   121629 | 0.00972957546
(293 rows)
```



2.5.6. 経路検索アプリケーションのためのデータ作成

経路検索エンジンは、通常、最短経路の検索のためにあらゆるラインについて始終点ノードを必要とします。ライン・ネットワークに、このデータを作成するということは、そのネットワーク上にトポロジーを作成することを意味します。 pgDijkstra は、PostgreSQL の中で始終点ノード情報を作成することができますが、性能はそれほど良くはありません。大きなデータセットを完成するには、およそ 1 日かかる場合もあります(DBのチューニングで、時間短縮が可能)。

トポロジーを少しでも速く作成するために利用されているソフトウェアが他にもあります:

PostGIS

この文章を書いている時点で、最新の PostGIS1.1.2 は、トポロジー生成機能を加え始めました。 しかし、アルファ段階で、トポロジーの作成方法に関する資料も非常に少ない状況です。トポロジーの機能性が使用するのに充分安定するようになると、情報は今後増えるでしょう。

ArcInfo

ArcInfo ライセンスがある場合、トポロジーの作成は BUILD コマンドを実行するだけです:

```
build line {Coverage Name}
```

そして、カバレッジを PostGIS でインポート可能な Shape ファイルで出力します。BUILD コマンド

は fnode_、tnode_、length カラムを作成します。それらは、PostgreSQL では、source、target にリネームされます。length は初期値に設定することができます。

GRASS

GRASS は、トポロジーを作成するのにも使用することができますが、shape ファイルに出力(export)したデータセットにトポロジー情報が含まれていないので、トポロジー情報を抜き出して、PostgreSQL に持ってくるのが ArcInfo ほど簡単ではありません。

トポロジー作成コマンド"v.build"は、順番にファイルに送ることができるストリームに情報を出力するダンプオプションを持っています。例えば:

```
v.build map=dourol option=build,dump > dourokukan.txt
```

出力はこのようになります;

```
---------- TOPOLOGY DUMP ----------
N,S,E,W,T,B: 35.897887, 24.281578, 153.985841, 138.943042, 0.000000, 0.000000
Nodes (148304 nodes, alive + dead ):
node = 1, n_lines = 3, xy = 139.756532, 35.67451
line = 1, type = 2, angle = -2.265356
line = -20, type = 2, angle = -0.055499
line = 8, type = 2, angle = 1.281166
node = 2, n_lines = 3, xy = 139.756261, 35.674216
line = -9, type = 2, angle = -2.827622
line = 2, type = 2, angle = -1.878154
...
...
...
Lines (220672 lines, alive + dead ):
line = 1, type = 2, offset = 14 n1 = 1, n2 = 2, left/area = 0, right = 0
N,S,E,W,T,B: 35.674510, 35.674216, 139.756532, 139.756261, 0.000000, 0.000000
line = 2, type = 2, offset = 79 n1 = 2, n2 = 3, left/area = 0, right = 0
N,S,E,W,T,B: 35.674216, 35.672010, 139.756261, 139.755285, 0.000000, 0.000000
line = 3, type = 2, offset = 160 n1 = 3, n2 = 4, left/area = 0, right = 0
N,S,E,W,T,B: 35.672010, 35.671649, 139.755285, 139.755014, 0.000000, 0.000000
```

table_topo.pl のような perl プログラムは、GRASS 出力をトポロジー情報を含むノードとラインテーブルを作成する SQL ファイルに変換するために使用することができます。 そして、これらのテーブルは始終点ノード情報を作成するために PostGIS ネットワークテーブルにリンクすることができます。

プログラム

トポロジーをサポート ： http://pgrouting.postlbs.org/ticket/27

2.6. チュートリアル

2.6.1. MapServer による経路の表示方法

（作成者 ： Camptocamp／ pgDijkstra）

shortest_path_as_geometry()ファンクションは、MapServer 内で最短経路を直接描画する:

```
LAYER
    NAME "europe"
    TYPE LINE

    STATUS DEFAULT
    CONNECTIONTYPE postgis
        CONNECTION "user=postgres host=localhost dbname=geo"
        DATA "the_geom from (SELECT the_geom, gid from
          dijkstra_sp('bahnlinien_europa_polyline', 2629, 10171)) AS
          foo using unique gid using srid=-1"

    TEMPLATE "t"
    CLASS
      NAME "0"
      STYLE
        SYMBOL "circle"
        SIZE 10
        COLOR 50 50 100
      END
    END
END
```

しかしながら、この機能は地図表示画面ごと呼び出されてしまい、その都度最短経路を計算するため、注意が必要です。

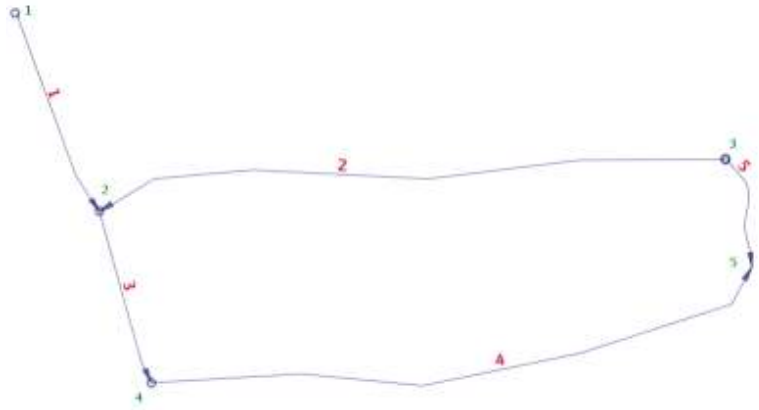
テンポラリなテーブル上で最短経路を生成する形にした方が、よいアプローチといえます。

2.6.2. 一方通行の取り扱い方法

Dijkstra 法と A* 法のアルゴリズムの両方とも、グラフのエッジの両端のコスト(負荷)を計算することができるため、一方通行の通りを含む道路ネットワークの経路探索に利用できます。

この例はその利用方法を説明しています。この例では、データは OpenJump を使用して作成され、pgRouting がインストールされている PostGIS のデータベースに保存されています。

グラフはこのようなものです; ほとんどのエッジが左から右にデジタイズされているにもかかわらず、エッジ#2 は右から左にデジタイズされていることに注意してください。これは、一方通行の通りをシミュレートするためにしました。



エッジの両端の cost(コスト)を計算するとき、経路検索アルゴリズムを通すために、reverse_cost(逆向きのコスト)フィールドが必要です。

初めに、cost と reverse_cost にエッジの長さをセットします。

```
routing=# UPDATE rtest SET cost=length(the_geom), rcost=length(the_geom);
UPDATE 5
```

そして、エッジ#2 の reverse_cost を増やすために、100 万を reverse_cost フィールドの値に加えます。

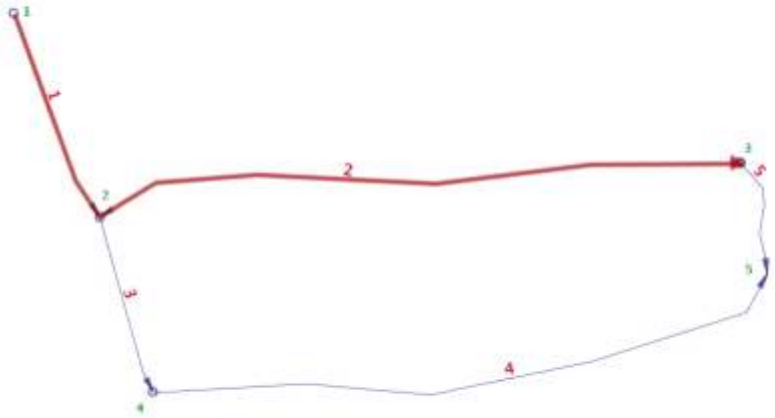
```
routing=# UPDATE rtest SET rcost=rcost + 1000000 WHERE gid = 2;
```

```
routing=# SELECT gid,cost,rcost,source,target FROM rtest ORDER BY gid;

gid  |          cost       |           rcost         | source | target
-----+---------------------+-------------------------+--------+--------
  1  | 90.4777391240398 | 90.4777391240398      |      1 |      2
  2  | 266.663211021467 | 000266.663211021467 |      3 |      2
  3  | 74.7975644284963 | 74.7975644284963      |      2 |      4
  4  | 264.887335603738 | 264.887335603738      |      4 |      5
  5  | 49.0618009646755 | 49.0618009646755      |      3 |      5
(5 rows)
```

Dijkstra および A*の両方のアルゴリズムの最後のパラメータは、グラフで経路を検索する時、reverse_cost も含めて計算するかどうか決定します。

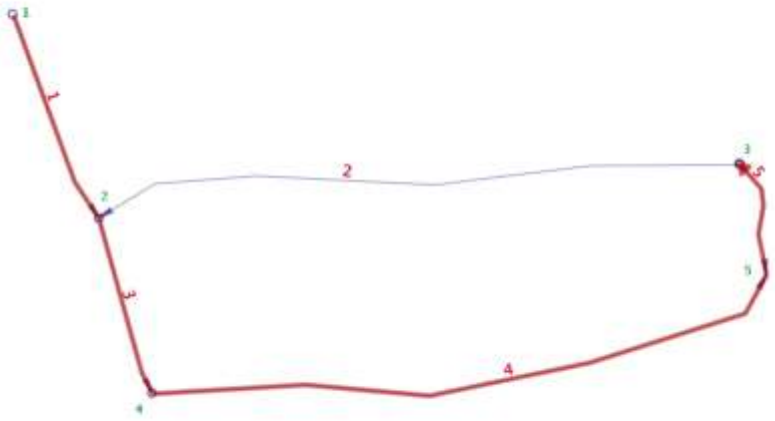
'偽' に設定すると、両方のアルゴリズムは、cost パラメータ(この場合は、それぞれのエッジの長さ)だけを使用して検索します。 この例では、ノード#1 からノード#3 までの経路を探索します。



今、reverse_cost が'真' に設定されると、アルゴリズムは reverse_cost を使用します。エッジ#2 のノード 2 がとても高いコストをもつことになり、別の経路を探します。

```
routing=# SELECT * FROM shortest_path_astar('SELECT gid AS id, source::int4,
        target::int4, cost::double precision, rcost::double precision AS reverse_cost,
        x1,y1,x2,y2 FROM rtest',1,3,false,true);

vertex_id | edge_id |          cost
----------+---------+------------------
         1 |       1 | 90.4777391240398
         2 |       3 | 74.7975644284963
         4 |       4 | 264.887335603738
         5 |       5 | 49.0618009646755
         3 |      -1 |                0
(5 rows)
```



エッジの両端のコストを計算する能力を持つことはすばらしい特徴ですが、パフォーマンスに影響を与えるため、本当に必要であるときにだけ使用してください。

```
routing=# EXPLAIN ANALYZE SELECT * FROM shortest_path_astar('SELECT gid
      AS id,source::int4, target::int4, cost::double precision, rcost::double
      precision AS reverse_cost,x1,y1,x2,y2 FROM rtest',1,3,false,false);

                                                    QUERY PLAN
------------------------------------------------------------------------------
-------------
Function Scan on shortest_path_astar  (cost=0.00..12.50 rows=1000 width=16)
(actual time=0.954..0.958 rows=3 loops=1)  Total runtime: 1.020 ms
(2 rows)
```

```
routing=# EXPLAIN ANALYZE SELECT * FROM shortest_path_astar('SELECT gid
      AS id, source::int4, target::int4, cost::double precision, rcost::double
      precision AS reverse_cost,x1,y1,x2,y2 FROM rtest',1,3,false,true);

                                                    QUERY PLAN
------------------------------------------------------------------------------
-------------
Function Scan on shortest_path_astar  (cost=0.00..12.50 rows=1000 width=16)
(actual time=11.088..11.093 rows=5 loops=1)  Total runtime: 11.155 ms(2 rows)
```

2.6.3. SQL tips & tricks

（訳者注：翻訳の対象外）

2.7. ワークショップ

2.7.1. OpenLayers における pgRouting 入門

（訳者注：翻訳の対象外）

2.8. pgRouting 関連のツール群

2.8.1. WebRoutingService(WRS)

pgRouting のための REST に対応したサービスのプロトタイプです。

概要

WebRouting サービスは、HTTP プロトコルを通して容易に経路検索機能を使用するように設計された Web サービスです。WebRouting サービスは、"リソース指向のアーキテクチャ"(Leonald Richardson & Sam Ruby 著"RESTful Web サービス"参照)である REST (Representational State Transfer)に基づいています。WebRouting サービスは、Java 言語で開発され、Java クラスを REST コンセプトにマッピングしたフレームワークである Restlet フレームワークを使用します。

手軽に使用されるように、WebRouting サービスは多くの空間データフォーマットに対応しています(Geojson, Gml, Kml,...)。

インストール (バージョン 1.0)

SVN リポジトリからプロジェクトをチェックアウトしてください:

```
svn checkout http://pgrouting.postlbs.org/svn/branches/routingservice wrs
```

必要条件:

* pgRouting 1.0
* Java Runtime Environment (JRE) 1.5 か、それ以上

Java 技術により、WebRouting サービスは、Java 実行環境のあるオペレーティングシステム(OS)にインストールすることができます。 現在のところ、私たちは Linux 環境でのみ WebRouting S サービスをテストしています。 そして、クライアント側には、POST/GET リクエストを生成・送信できるアプリケーションのみ必要です。

WebRouting サービスのダウンロード:

WebRouting サービスの最新のリリースをダウンロードしてください。 ダウンロード完了後、tar ファイルを WebRouting サービスをインストールしたいディレクトリに解凍してください。

設定:

WebRouting サービスを開始する前に、サーバの設定が必要です。WebRouting サービス・ディレクトリ直下にある"configuration.xml"ファイルを編集してください。

Linux の開始方法:

以上で WebRouting サービスの準備は終わりましたので、シェルスクリプト:"start.sh"を実行します。 エラーがなければ、インストールは OK です。WebRouting?サービスは、開始されています。

エラーの場合、2 つの可能性があります:

- XML 設定ファイルに不具合があります。この場合は、"設定"ステップに戻ってください。
- JRE が正しくインストールされないか、または JAVA_HOME 環境変数が正しく定義されていません。

```
./start.sh
```

使用方法

インストールが問題ない場合は早速リクエストを生成できますが、その前に"WebRouting サービスの機能"を見てみましょう。

WebRouting Service をコールするには、2 つの方法:GET リクエストおよび POST リクエストがあります。

両方とも、プロバイダ、サービス、入出力データの形式を指定した URL を与えなければなりません。URL の形式は:

```
http://ip_address:port_number/provider_name/format_for_input_data/service_name.format_for_output_data
```

例:

```
http://178.1.1.10:8182/geobase/geojson/shortest_path.kml
```

この例では、geobase プロバイダの geojson 入力データで WebRouting サービスをコールし、最短経路サービスで KML オブジェクト出力を求めています。このように、URL は、"何を持っているか"および"何が欲しいか"を指定します。 次のステップは:指定されたサービスで使用されるパラメータの値を与えることです。1 つのサービスに利用可能なすべてのパラメータは、XML 設定ファイルに指定されています。また、1 つのパラメータが必須あるいはオプションかどうかについても、指定することができます。 このステップは、GET または POST リクエストの間で行います。

リクエストを生成する準備ができましたら、"test"レパートリーの"test.html"をオープンしてください。 このページは、WebRouting サービスをテストする簡単な方法を提供します。 フォームを全て埋めると、GET/POST リクエストが自動的に生成されます。リクエストの結果は、Web ページ下部に表示されます。

もし動かない場合は、"インストール"ステップに戻ってください。

== ビルド方法 ==

近日公開 :)

2.8.2. pgRouting ギャラリー

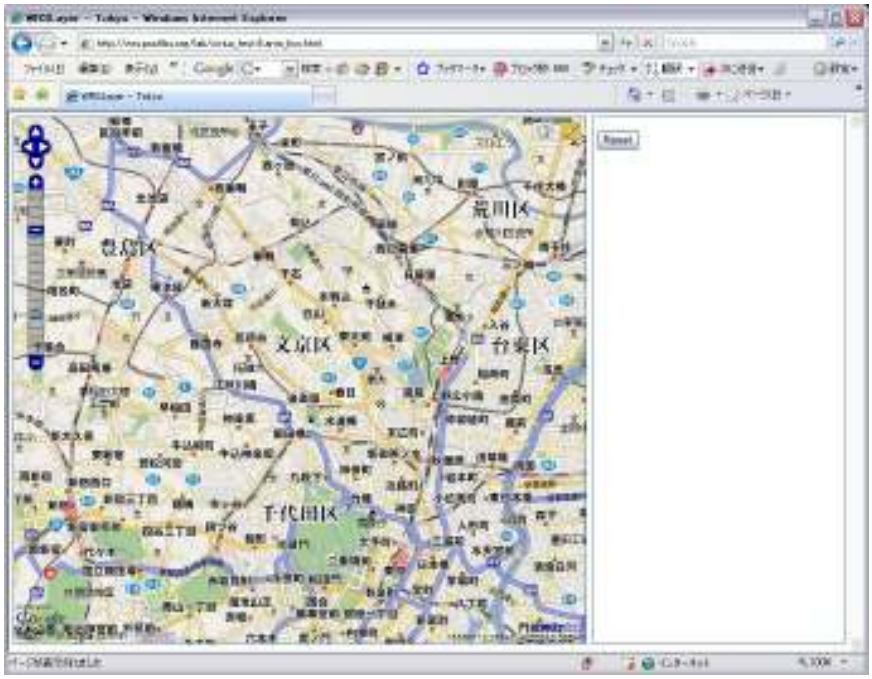
Live projects

・http://maps.mapmobility.com (webmapping application by DM Solutions Group, pgRouting built in by Orkney, Inc.)

WebRoutingService sandbox

各種デモ

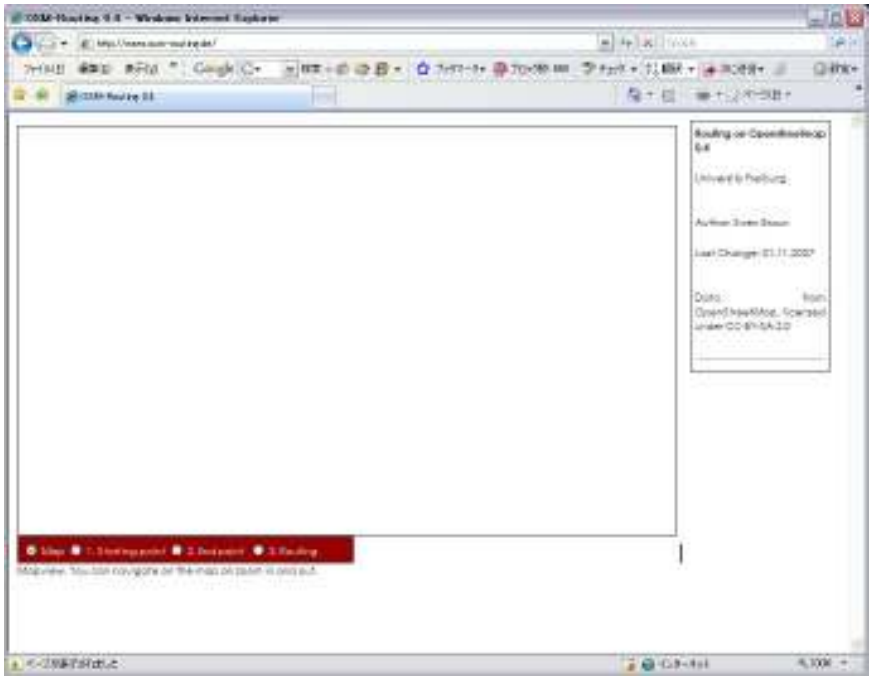
・http://wrs.postlbs.org/lab/cirius_test/kanto_hcc.html



pgRouting with OSM

OSM のデモ

・http://www.osm-routing.de/

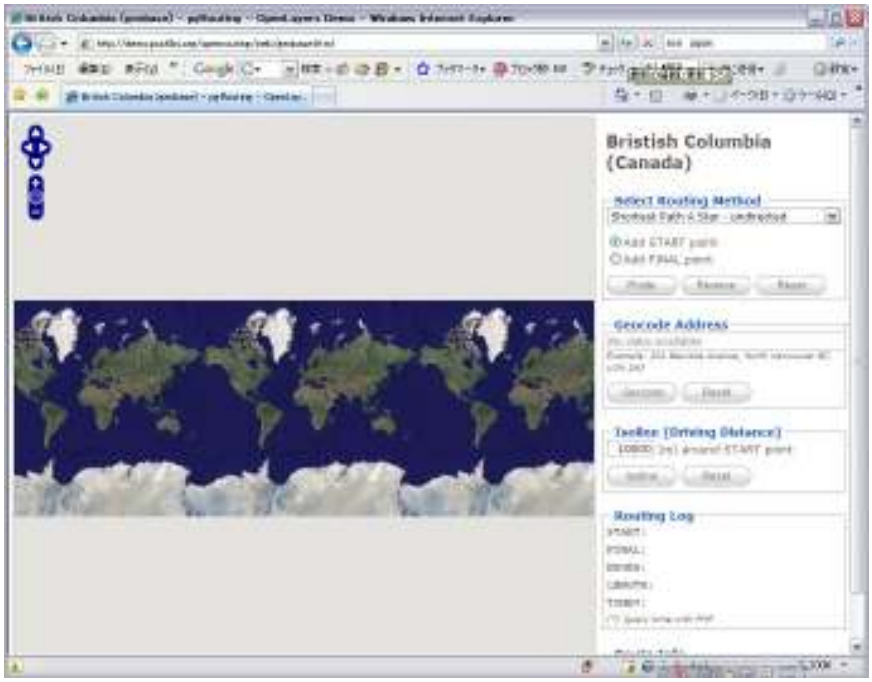


pgRouting with OpenLayers

OpenLayers のデモ

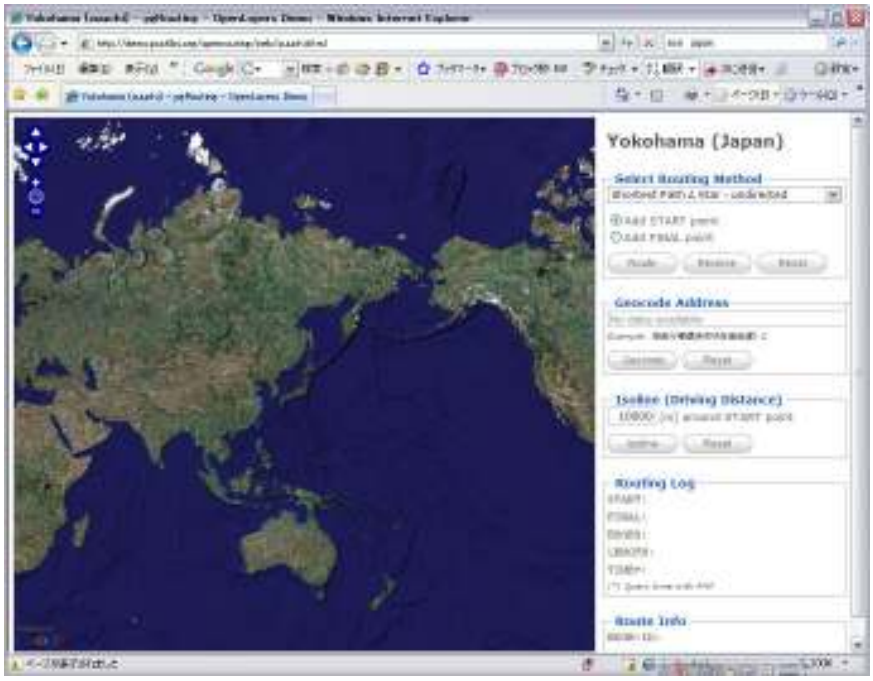
・http://demo.postlbs.org/openrouting/geobase.html

Bristish Columbia の Geobase Canada 道路ネットワークデータ



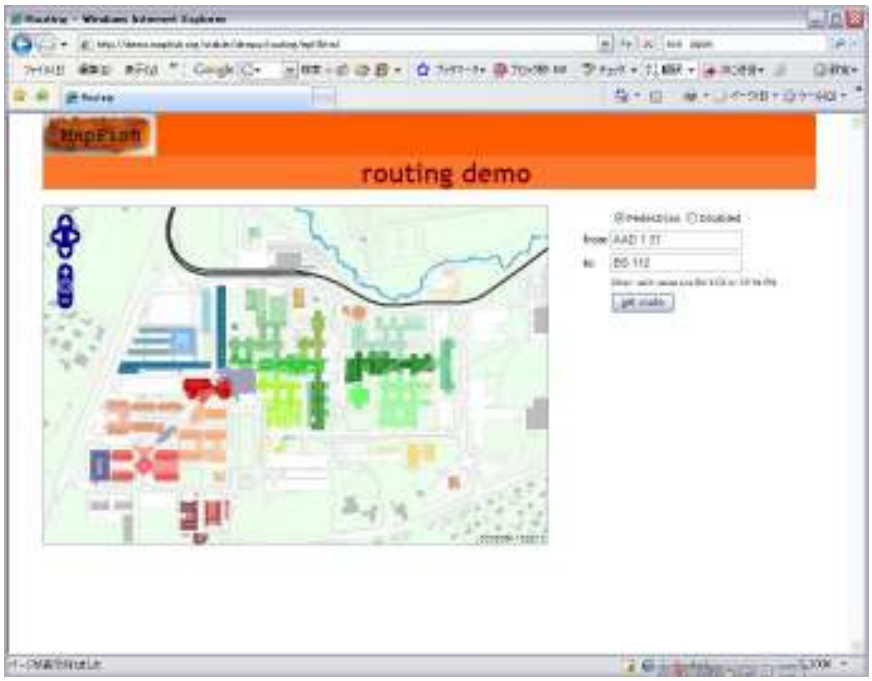
・http://demo.postlbs.org/openrouting/suuchi.html

数値地図の道路ネットワークデータ



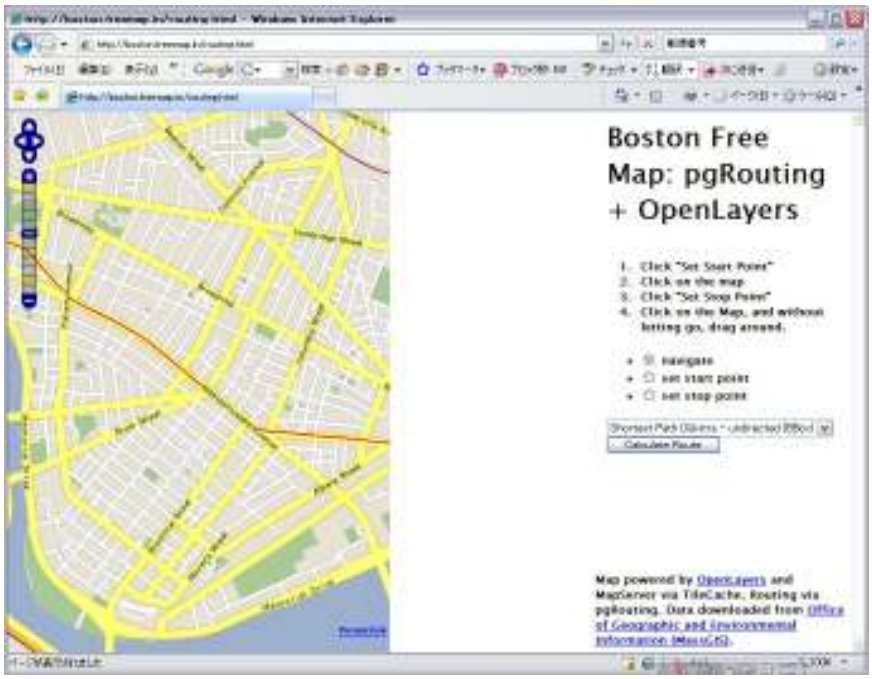
・http://demo.mapfish.org/stable/demos/routing/epfl.html

Camptocamp Mapfish 経路検索デモ



・http://boston.freemap.in/routing.html

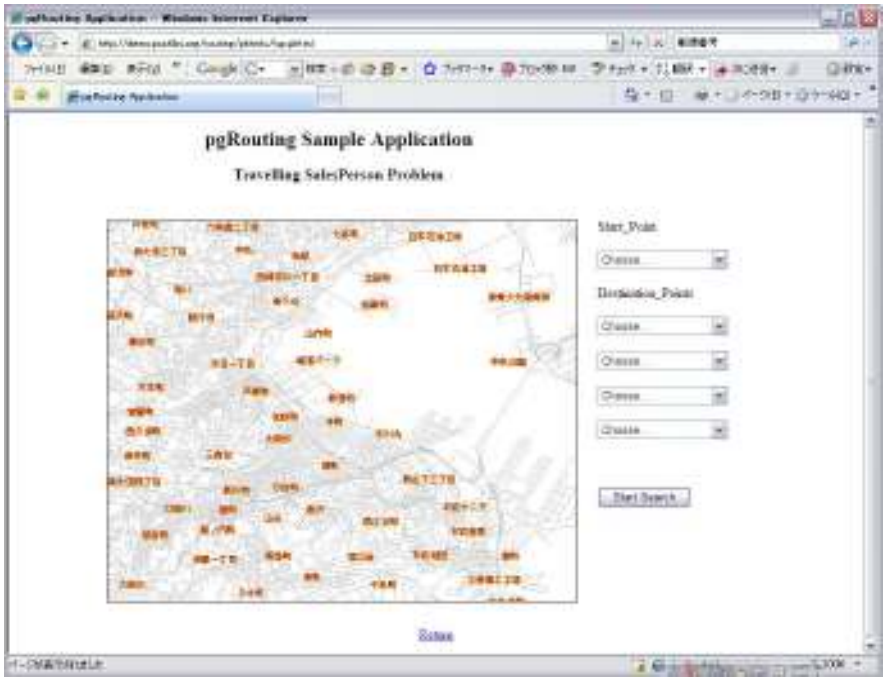
Christopher Schmidt's デモ



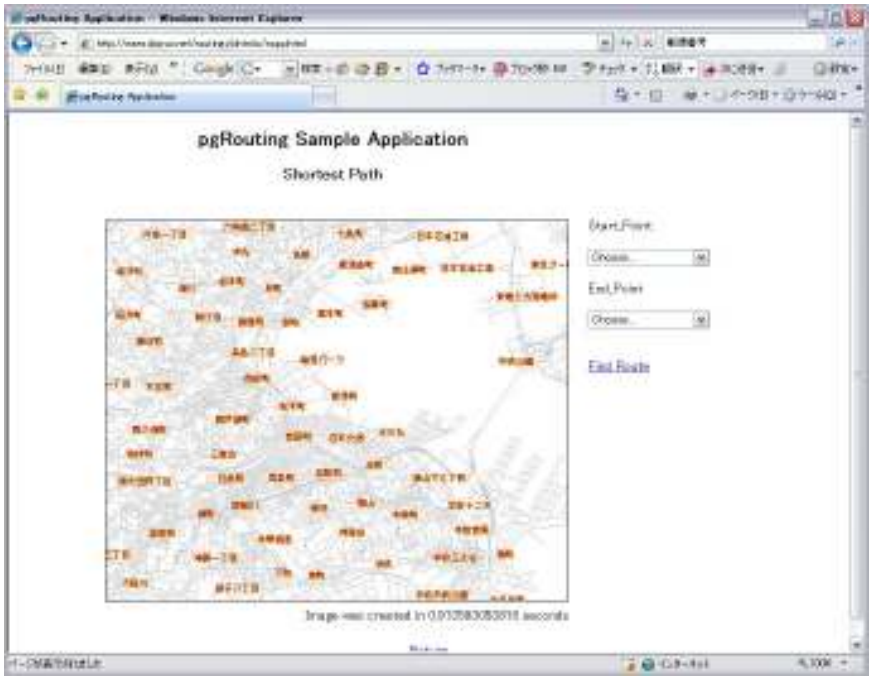
Two version with the Kanagawa sample data

神奈川県のサンプルデータ(2 つのバージョン)

・http://demo.postlbs.org/routing/



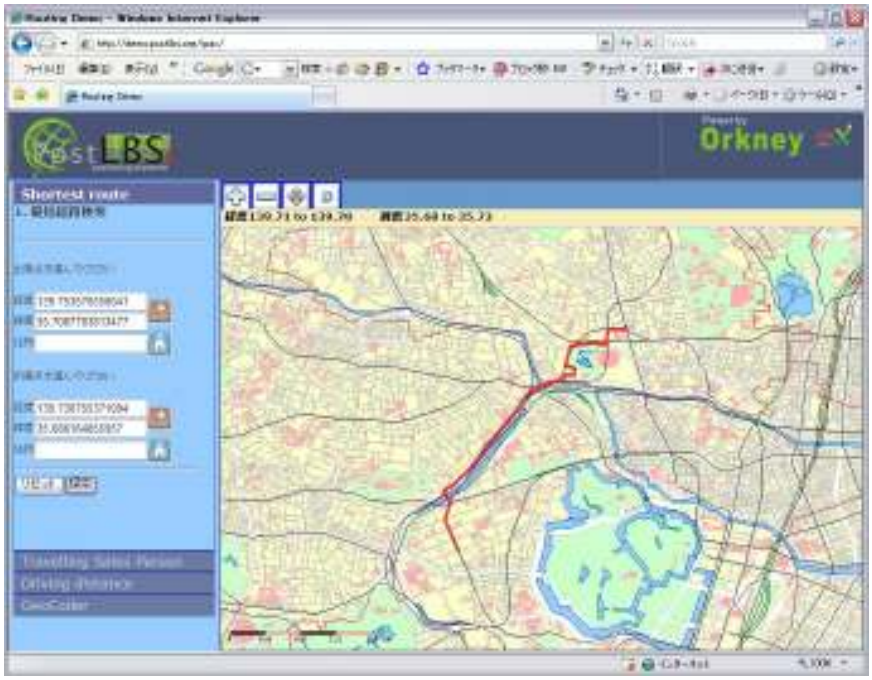
・http://www.djayux.net/routingj/



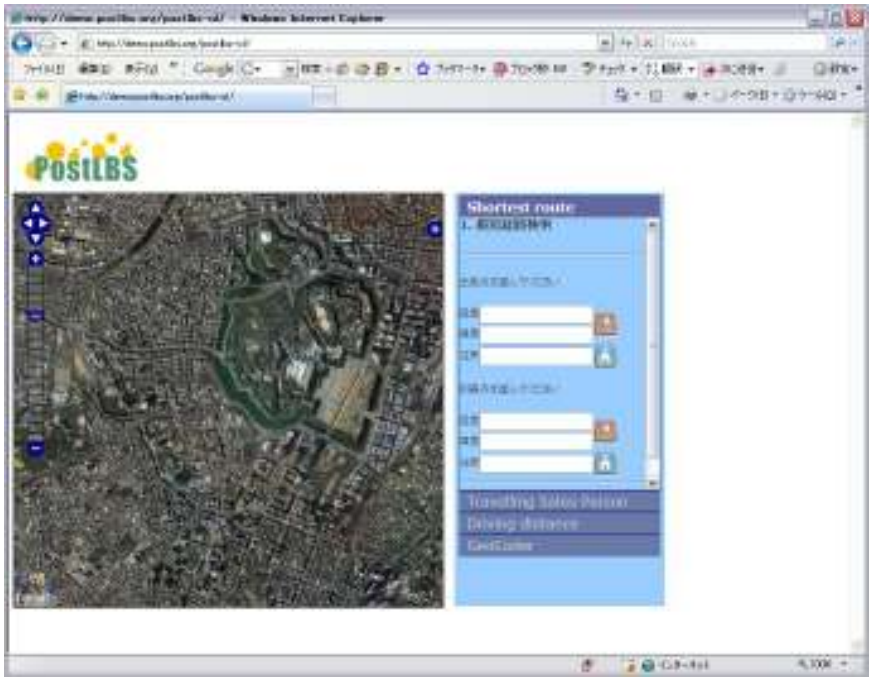
Previous demos

以前のデモ

・http://demo.postlbs.org/ipax/



・http://demo.postlbs.org/postlbs-ol/



著作権および配布ライセンス

1. pgRoutingの機能説明原文の著作権は、株式会社オークニーに帰属します。
   機能説明原文は株式会社オークニーが運営する下記のサイトで公開されています。
   PostLBSサイト － http://www.postlbs.org/
   pgRoutingサイト － http://pgrouting.postlbs.org/

2. 翻訳文書の著作権は、日本ユニシス株式会社に帰属します。
   翻訳文書の配布ライセンスは、GNU Free Doucument License (GFDL) に従います
   (下記英文を参照)。

なお、翻訳文書の記述内容により生じたいかなる損害についても責任を負いかねますので、ご了承ください。

更新履歴

2008 年 4 月　第 1 版を公開（2008 年 4 月時点の原文を翻訳）
　　著作者　日本ユニシス株式会社
2008 年 5 月　誤記を修正

謝辞

翻訳文書の査読を快くお引き受けいただきました、株式会社オークニーの森亮様、株式会社中央ジオマチックスの三瓶司様および安江茂隆様各位に、この場をお借りして心よりお礼を申し上げます。