Vol.2013-DBS-157 No.3
Vol.2013-IFAT-111 No.3
2013/7/22

グラフ構造に基礎をおく地図データベース構造の試作と福岡市街道路網での検証

本田真也[†1]　　金子邦彦[†2]

道路網や公共交通網を記述した地図データベースに重ね合わせるような形で、渋滞情報や最短経路案内などの交通情報を表示したいとき、交差点、信号機のような地点データだけでなく、道路の区間も精密に記述する必要がある。そのため、地図データベースでは、交差点、信号機を記述する点オブジェクトに加え、区間オブジェクトを扱える必要がある。これらの 2 種のオブジェクトは全体でグラフ構造をなす。グラフ構造は，R 言語のような外部言語のオブジェクトとして簡単に取り扱える必要もある。本稿では、R 言語のグラフ構造のオブジェクトを，レコード構造にマッピングする．このことで，グラフは，SQL 言語や R 言語などで簡単に扱えるようになる。本稿では，福岡市内の道路網の地図データベースを試作した報告を行う。福岡市街地の道路網に対して R 言語の最短経路検索機能を簡単に扱えることの確認までを終えている。

A Map Database Structure based on Graph Structure and its evaluation using a Fukuoka City Road Map

SHINYA HONDA[†1]　KUNIHIKO KANEKO[†2]

When overlay displaying traffic information onto a network data such as road-network or traffic-network, it must be described as point data and interval data. A point represents an intersection of roads, or a traffic signal, etc. Then, a map database constists of two types of data. They are point type and interval type. These objects form a graph structure. The problem is the easiness of use of graph data on a host language such as R language. In this report, a graph is mapped onto sets of records. It enables to handle graph data easily using programming languages such as R language, SQL language, etc. In this report, a Fukuoka City map database is also presented. Using a graph object describing roads in the Fukuoka City, graph handlings such as finding shortest paths becomes easy.

1. はじめに

近年、地図データベースの普及とともに、国土地理院による地図データの公開、OpenStreetMap プロジェクトによる世界規模の地図データの公開などオープンで無償使用可能な地図データベースが整備されつつある。OpenStreetMap プロジェクトの地図データベースは、道路網や鉄道網などが精密に記述されている。その内部のデータ構造は、地図の編集がしやすいような種々の工夫がなされている。OpenStreetMap の地図ファイルを容易に扱えるライブラリやソフトウエアも種々、公開されており、多くは無料で利用可能である。本研究では R 言語の osmar パッケージを用いて OpenStreepMap の地図ファイルを扱う。以下、本稿では、OpenStreetMap の地図ファイルのことを osm ファイルと記す。

本来、OpenStreetMap のデータファイル形式は、編集容易なように種々の工夫がなされている。一方で、グラフ G=(E, N) （E は枝集合，N は節集合）を扱うとき，各枝と各節を 1 つのレコードとして扱うことが多い。例えば、各節は緯度と経度という 2 つの属性を含むレコード、各節は両端の節 ID を属性として含むレコードとして扱う。R 言語で整備されているグラフのパッケージ、例えば graphNEL パッケージや igraph パッケージでも、レコード形式としてグラフを取り扱う。

本稿の構成は以下の通り．2 章では OpenStreetMap が配布している地図ファイルの概要と、osm ファイル内のグラフデータを、レコード形式のデータに変換する手続きとについて説明する。3 章は、osm ファイルを用いた最短経路探索の試行例について報告する。4 章は、osm ファイルの図化（プロット）について説明する。

2. osm ファイル

2.1 osm ファイルのデータ構造

osm ファイルは node,way,relation の 3 つの要素からなるリストであり、各要素はいくつかのデータフレームから構成されている。まず一つ目の node の要素は attrs,tags という二つのデータフレームを持っており、attrs は node の id やその node の緯度経度の情報などが格納されている。tags には node の id,その node の内容を大まかに説明する情報である k、細かく説明する情報である v が格納されている。つぎに二つ目の way の要素は attrs,tags,refs という 3 つのデータフレームを持っている。attrs には way の id や timestamp などの情報が格納されている。tags には way の id,その way

†1　九州大学
Kyushu University
†2　九州大学
Kyushu University

Vol.2013-DBS-157 No.3
Vol.2013-IFAT-111 No.3
2013/7/22

の内容を大まかに説明する情報である k,細かく説明する情報である v が格納されている。refs は way の id とその way を構成する node の id が格納されている。最後に 3 つ目の要素である relation の要素も way と同様に attrs,tagas,refs の 3 つのデータフレームを持っている。attrs は relation の id や、timestamp などの情報が格納されている。tags には relation の id、その relation を大まかに説明する情報である k,細かく説明する情報である v が格納されている。refs には relation の id,relation を構成するオブジェクトのタイプを示す type,メンバーの id を示す ref,メンバーの役割をあらわす文字列 role から構成されている。

2.2 博多駅周辺の osm ファイル取得

博多駅周辺 1km 四方などの少ないデータ数の場合 OpenStreetMap API を使用してデータをダウンロードすることができるが、OpenStreetMap API はダウンロードできるサイズが制限されているので広大な範囲のデータをダウンロードして使用することはできない。R を用いて博多駅周辺 1km 四方のデータを取得するにはまず範囲の中心となる博多駅の緯度経度を調べる必要がある。Osmar パッケージをダウンロードし読み込んだ後、博多駅の緯度経度の情報と取得したいデータの範囲の値とデータがどこにあるのかを関数 get_osm()に入力するとその範囲のデータを取得することができる。データの取得が完了すると R でさまざまな操作が可能となる。以下に示した図は取得した osm データから道路と建物の情報を取りだし、グラフィック処理を施してからプロットしたものである。また node 要素内の attrs データフレームを csv ファイルに書き出したもののファイルサイズは 195.3kB であり、node の個数は 2247 個であった。

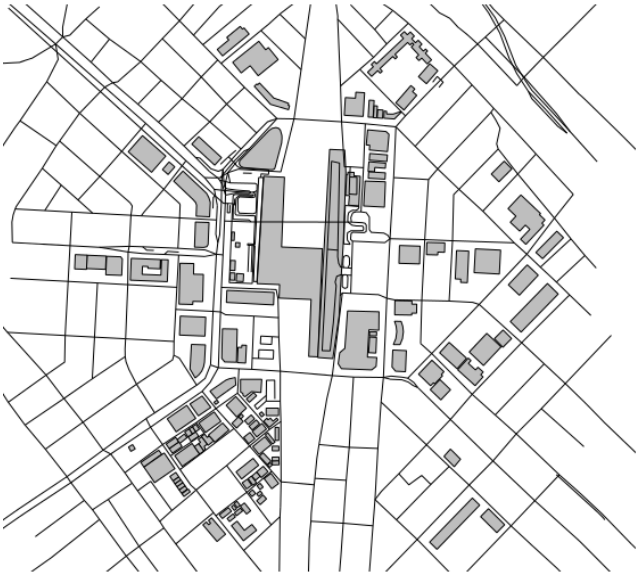

図 1　博多駅周辺 1km 四方をプロットした図

2.3 福岡市街地の osm ファイルの取得

福岡市街地の osm ファイルをダウンロードする場合ファイルサイズが大きく、OpenStreetMap API を使用してデータをダウンロードすることは不可能である。そのため GEOFABRIK が配布している日本全域の OpenStreetMap データをダウンロードし、その後必要となる範囲のデータを切り取る。まず GEOFABRIK のライセンス条項の確認を行ってから japan-latest.osm.bz2 というファイルをダウンロードする。japan-latest.osm.bz2 のファイルサイズは 1.3GB であった。次に福岡市はおおよそ緯度が 33.4308～33.7147、経度が 130.2037～130.5053 という範囲なので、その範囲を切り取るようにコマンドを端末で入力することで福岡市の osm ファイルを作成することができる。この範囲で切り取ったファイル fukuoka-city.osm のファイルサイズは 100.3MB であった。R を用いてこのデータを操作する際には一度 R にデータを読み込む必要がある。今回は経度 130.3545,緯度 33.5725 を中心として東西方向に 27km、南北方向に 30km の範囲を読み込んで操作を行った。またこの範囲のデータを読み込むのに約 4 時間かかった。

3. 最短経路探索

3.1 新たなデータフレームの作成

R を用いて OpenStreetMap のデータを扱う際に、igraph を用いて最短経路探索などを行おうと思ったが、R に実装されている osmar オブジェクトを igraph オブジェクトに変換する関数である as.igraph()はエラーが出てしまい使用することができなかった。そこでこれから他の作業で OpenStreetMap のデータを扱う際にもデータを扱いやすいようにすべてのデータフレームを csv ファイルとして書き出し、さらに igraph オブジェクトに変換しやすいような新たなデータフレームを作成する関数を作成し、新たなデータフレームを作成した。このデータフレームは way の id,way の始点である node の id,way の終点である node の id,それら 2 地点の緯度経度の情報、2 地点間の距離の 8 つの要素からなるデータフレームとした。

3.2 関数の説明

はじめに osmar オブジェクトから subset 関数を用いて道路の node,way 要素のみを取り出し、csv ファイルに書き出しておく。道路の way 要素の refs のデータフレームから way の id,始点となる node の id,終点となる node の id の 3 つの列を作成する。次にこの 3 列と道路の node 要素の attrs データフレームから始点と終点との各 node の緯度、経度の情報を取り出し緯度経度の情報の 4 列を作成する。最後に始点と終点の node の緯度経度からその 2 地点間の距離を計算する。緯度経度から距離を計算する方法はいくつかあるが、今回の関数ではヒュベニの公式を用いて計算した。操

作を行った道路の way 要素の refs は csv ファイルで 254734 行あり、ファイルサイズは 7.0MB であった。作成した新たなデータフレームは csv ファイルのファイルサイズは 21.3MB であり、213611 本の線の情報を含んであった。また変換にかかった時間は 6 時間 47 分であった。図 2 は新たに作成したデータフレームの一部であり、列名は左から順に id,node1,node2,node1lat,node1lon,node2lat,node2lon,weight となっている。R に実装されているデータ型を調べる関数 mode()で各列のデータを調べた結果、データ型は全て実数である numeric であった。

id	node1	node2	node1lat	node1lon	node2lat	node2lon	weight
23341156	1253326223	1253326221	33.6353476	130.4220216	33.6352338	130.4224027	37.5430409401
23341318	716676278	1844820369	33.6321826	130.4905336	33.6324445	130.4906068	29.8322030491
23341318	1844820369	1844820371	33.6324445	130.4906068	33.63311	130.4907291	74.6815092297
23341318	1844820371	1844820373	33.63311	130.4907291	33.6336067	130.4907999	55.4820984298

図 2　新たに作成したデータフレームの一部

3.3 ヒュベニの公式の説明

2 地点の緯度経度情報から距離を算出する公式には地球を球体としてみる簡易的な公式や楕円体であることを考慮したヒュベニの公式、Lambert-Andoyer の公式などがある。精度がもっとも良いのはこの中では Lambert-Andoyer の公式であるが、実装が簡単であり、また距離が 50km 以下ではほとんど誤差が同じであるという理由から今回の関数ではヒュベニの公式を利用した。ヒュベニの公式は以下で示す。ただし地点 1 の緯度経度をそれぞれ lat1,lon1,地点 2 の緯度経度をそれぞれ lat2,lon2、求める距離を D とする。また M.N はそれぞれ子午線曲率半径、卯酉線曲率半径をしめしている。

D = ((M * latDiff)^2 + (N * cos(latAveRad) * lonDiff)^2)^0.5

M=6334834 / ((1 - 0.006674 * sin(latAveRad)^2)^3)^0.5
N = = 6377397 / (1 - 0.006674 * sin(latAveRad)^2)0.5
latDiff = latRad1 – latRad2
lonDiff = lonRad1-lonRad2
latAveRad = (latRad1 + latRad2) / 2
lonRad1 = lon1 * PI / 180
latRad1 = lat1 * PI / 180
lonRad2 = lon2 * PI / 180
latRad2 = lat2 * PI / 18

3.4 グラフの作成

最短経路探索を行うためにはまず道路の情報を重みを持ったグラフとして表現する必要がある。グラフとして表現することで igraph パッケージ内の最短経路を探す関数である get.shortest.paths()を使用することで簡単に最短経路を探すことができる。グラフを作成するためにはどことどことの node がつながっているのかというエッジリストの情報とその辺の重みの情報とそれぞれの node の情報とをベクトルとして準備する必要がある。それらのベクトルを使うことで関数を使用してグラフを作成することができる。

3.5 実行結果

実際に福岡市街の道路のグラフを作成し、乱数を発生させてそれと対応させた node の id を始点と終点に設定した後に igraph パッケージ内の最短経路を探索する関数である get.shortest.path()を実行した。get.shortest.path()は最短経路の node を list として返す関数である。1 度目の実行で作成された list は 429 個の要素を持ち、これを csv ファイルに書き出したもののファイルサイズは 6.2KB であった。また乱数を発生させて始点と終点を設定し関数 get.shortest.paths()を呼び出した際しばしばエラーが発生して最短経路が求まらない場合が観測できた。これは乱数によって設定された始点と終点を結ぶ経路が見つからない場合に観測されると思われるので、無視して関数を呼び出しなおした。図 4 はランダムに選んだ 2 地点で 50 回最短経路探索を行ったときの計算時間の分布である。横軸は経路上の node の数を示しており、縦軸はかかった計算時間を示している。この表から実行時間は経路の node 数にほとんど比例していることが観測できる。

	A	B
1	1	2014965143
2	2	2014965194
3	3	2014965207
4	4	2014965373
5	5	2014965399
6	6	2014965390
7	7	2014965386
8	8	2014965315
9	9	2014965294
10	10	2014965306
11	11	2014965369
12	12	2014965365
13	13	2014965230
14	14	2014965228
15	15	2014965232
16	16	2014965300
17	17	2150451897
18	18	2150451898
19	19	2150451902
20	20	2150451903

図 3　得られた経路データの一部

Vol.2013-DBS-157 No.3
Vol.2013-IFAT-111 No.3
2013/7/22

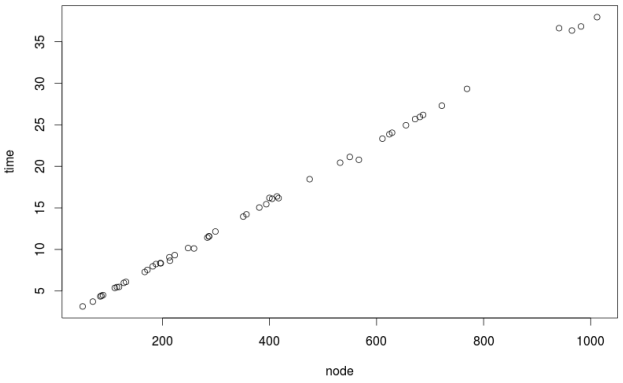

図 4　経路の node 数と実行時間の関係

4. プロット

4.1 R を用いた node の 3D プロット

3D プロットを行うためには R の rgl というパッケージをダウンロードする必要がある。このパッケージを読み込むことで関数 plot3d()を使用することができる。3 次元でプロットを行うことで各 node は 2 次元上にプロットされているが、斜め上から見た様子を確認することなどが可能となる。3D プロットを行うためには軸となる 3 つの値を持つデータフレームを作成する必要がある。今回は node の緯度経度の情報を用いてプロットを行いたいので node 要素の attrs データフレームを csv ファイルで書き出したものから緯度経度の情報を持っている 2 列を抜き出し、それに高さの列を加えた csv ファイルを新たに作成した。ただし高さの列はすべて値を 0 とした。プロットした node の数は 396536 であり、そのうち traffic_signal の tag がつけられているものは 453 個であった。以下にプロットした結果を示す。図 5 の各黒点は node であり、大きな黒点は traffic_signals の tag が付けられている node を示している。

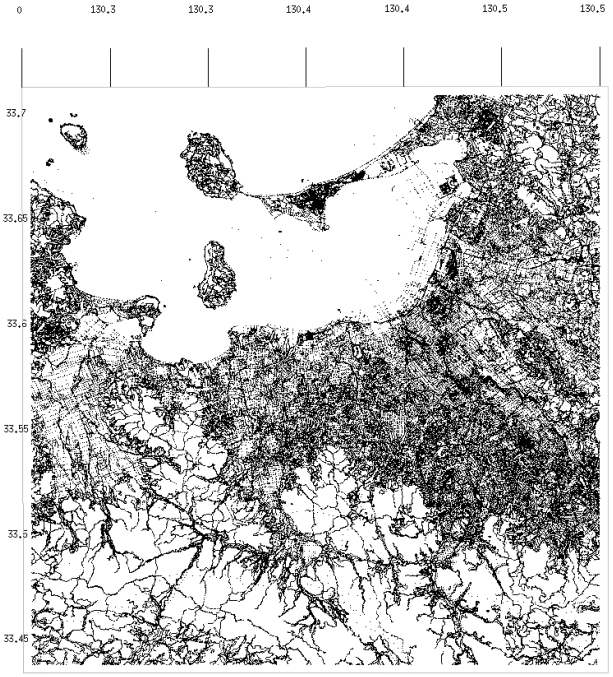

図 5　plot3d の結果(1)

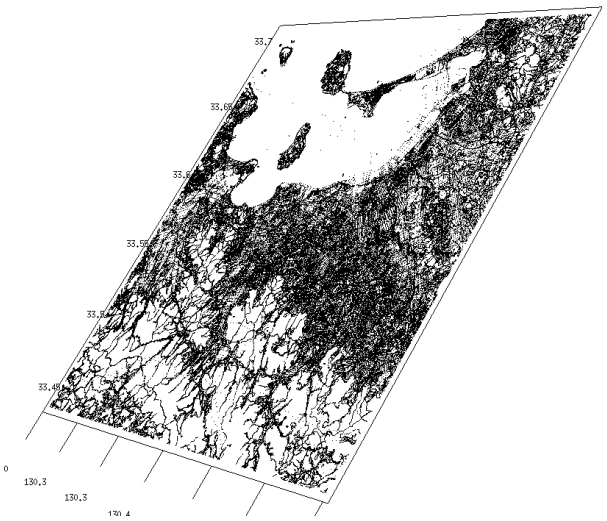

図 6　plot3d の結果(2)

Vol.2013-DBS-157 No.3
Vol.2013-IFAT-111 No.3
2013/7/22

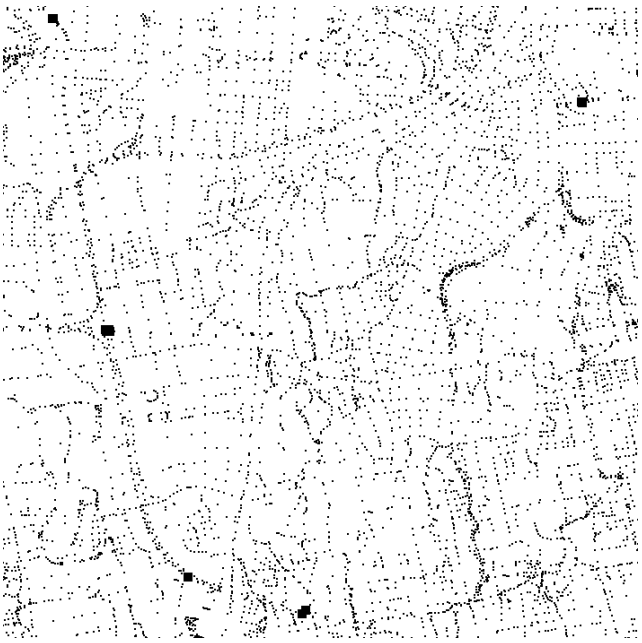

図 7　plot3d の結果の拡大図

4.2 R を用いた way のプロット

福岡市街地の道路の情報のみを取り出したものに osmar パッケージ内のグラフィック処理を行う関数 as_sp()を使用した。関数 as_sp()はグラフィック処理の方法を points,lines,polygons の 3 種類から指定して行うことが可能である。今回は道路の情報なので lines を指定して関数を実行した。以下にその結果を示す。

図 8　道路の情報のグラフィック処理後のプロット

5. おわりに

本稿ではグラフオブジェクトを、レコード集合に変換したのち、最短経路探索とプロットを行った。レコード集合への変換は容易であった。本稿では報告しなかったが、リレーショナルデータベースへの格納も容易である。今後、他種データとの統合（例えば、ある緯度、経度から撮影した画像データとの連携や、「POI」と呼ばれる地点データセットとの連携など）を進める。これで、OpenStreetMap の地図データが他種のデータと連携できる。日本全国規模の大規模なデータベース試作も今後の課題である。

謝辞

本研究は（独） 新エネルギー・産業技術総合開発機構，IT 融合による新社会システムの開発・実証プロジェクト（テーマ名：移動体データ銀行で実現する次世代交通情報共通基盤アジアモデルの構築）の助成を受けたものです。

付録

```
newdataframe <- function(){
  df_x <- NULL
  ways_refs <- read.table("/home/sh /
      Rproject/No.1/dataframe/hw_ways_refs.csv"
)
  id_ref                                        <-
cbind(ways_refs["V2"],ways_refs["V3"])
  i <- 1
  while(i <= nrow(id_ref)){
    wayid <- id_ref$V2[i]
    df_a <- subset(id_ref, id_ref["V2"] == wayid)
    n <- nrow(df_a)
    df_b <- df_a["V3"]
    df_a$V2[n] <- NA
    df_a <- na.omit(df_a)
    df_b$V3[1] <- NA
    df_b <- na.omit(df_b)
    df_c <- cbind(df_a,df_b)
    df_x <- rbind(df_x,df_c)
    i <- i+n
  }
  colnames(df_x) <- c("id", "node1", "node2")

  nodes_attrs                                   <-
read.table("/home/sh/Rproject/No.1

      /dataframe/nodes_attrs.csv")
  df_NA <- data.frame(V3=NA,V4=NA)
  df_y <- NULL
  for(i in 1:nrow(df_x)){
    node1id <- df_x$node1[i]
    df_a                                        <-
subset(nodes_attrs,nodes_attrs["V2"]==node1id)
    {if(nrow(df_a)==0){
      df_a <- df_NA}
      else df_a <- cbind(df_a["V3"],df_a["V4"])}
{if(nrow(df_b)==0){
      df_b <- df_NA}
      else df_b <- cbind(df_b["V3"],df_b["V4"])}
    df_c <- cbind(df_a,df_b)
    df_y <- rbind(df_y,df_c)
  }
  colnames(df_y) <-

c("node1lat","node1lon","node2lat","node2lon")
  df_xy <- cbind(df_x,df_y)
  df_xy <- na.omit(df_xy)

##ヒュベニの公式
  a = 6378137.0
  b = 6356752.314140
  e2 <- (a^2 - b^2) / a^2
  Mnum <- a * (1 - e2)
  df_z <- NULL
  df_NA <- data.frame(d=NA)
  for(i in 1:nrow(df_xy)){
    x1 <- df_xy$node1lat[i]
    y1 <- df_xy$node1lon[i]
    x2 <- df_xy$node2lat[i]
    y2 <- df_xy$node2lon[i]
    dy <- (x1 - x2)*pi/180
    dx <- (y1 - y2)*pi/180
    my <- ((x1 + x2) / 2)*pi/180
    W <- sqrt(1 - e2 * sin(my)^2)
    M <- Mnum / W^3
    N <- a / W
    d <- sqrt((dy * M)^2 + (dx * N * cos(my))^2)
    d <- as.data.frame(d)
    df_z <- rbind(df_z,d)
  }
  colnames(df_z) <- c("weight")
  newdf <- cbind(df_xy,df_z)
  return(newdf)
}
```

図 7 igraph 作成用データフレーム作成関数のソースコード