BENVGSA3

GI Stystems and Science

A tool for the calculation of the compactness of city blocks in R

January 11, 2016

Summary

The R function FUN_SPDF_Cmi_Index.R calculates the index of compactness based on the Second Moment of Inertia of a set of polygons. It has been developed in the context of urban morphology, as a tool to be used in the analysis of street typology of cities. The function takes as an input a set of polygons in the form of a SpatialPolygonDataFrame (SPDF), which is a class of elements of the package "sp". The output is the same SPDF with two added columns containing the area and the index of compactness C_{MI} of each polygon.

Word count : 1458

Contents

1 Context

The street network is an important component of cities: it emerges from the underlying processes at play in the formation and evolution of the city and it is the subject of different types of analyses in different fields. In urban morphology, the road network is used as a schematic view of the city which captures the relationships between locations and between location and the whole. The street network can be approximated fairly well with a planar graph, that is, a network graph that can be embedded in a plane. Different classifications of street networks have been proposed (Lopes Gil, 2014; Marshall, 2004) as well as different analysis tools (Hillier and Hanson, 1984). Louf and Barthélemy (2014) propose a classification based on a "fingerprint" of the cities defined by the conditional distribution of the areas of the cells and their compactness. The underlying rationale is that the road network graph, because it is the representation of a physical phenomenon, contains information not at the topology level only but in its geometry. The geometry of the network is the set of polygons defined by the vertexes or nodes of the network: these are called cells or faces and they correspond to the city blocks.

Roads can be defined in many ways such as the segments between two intersection, as a visual axis (Hillier and Hanson, 1984) or could follow the road names. On the other hand, blocks are unambiguously defined as "the smallest area delimited by roads" (Louf and Barthélemy, 2014). Louf and Barthélemy (2014) performed their analysis and classification across many cities, using the ratio between the area of the polygon and the area of the circumscribed circle as a measure of compactness. For the purpose of this essay, I am interested in the classification at the local level, not as an average of the city, in order to investigate whether local neighbourhoods have their own fingerprints. I will also be using a different index of compactness, as described by Li et al. (2013), that is the measure based on the second moment of inertia of a shape C_{MI}.

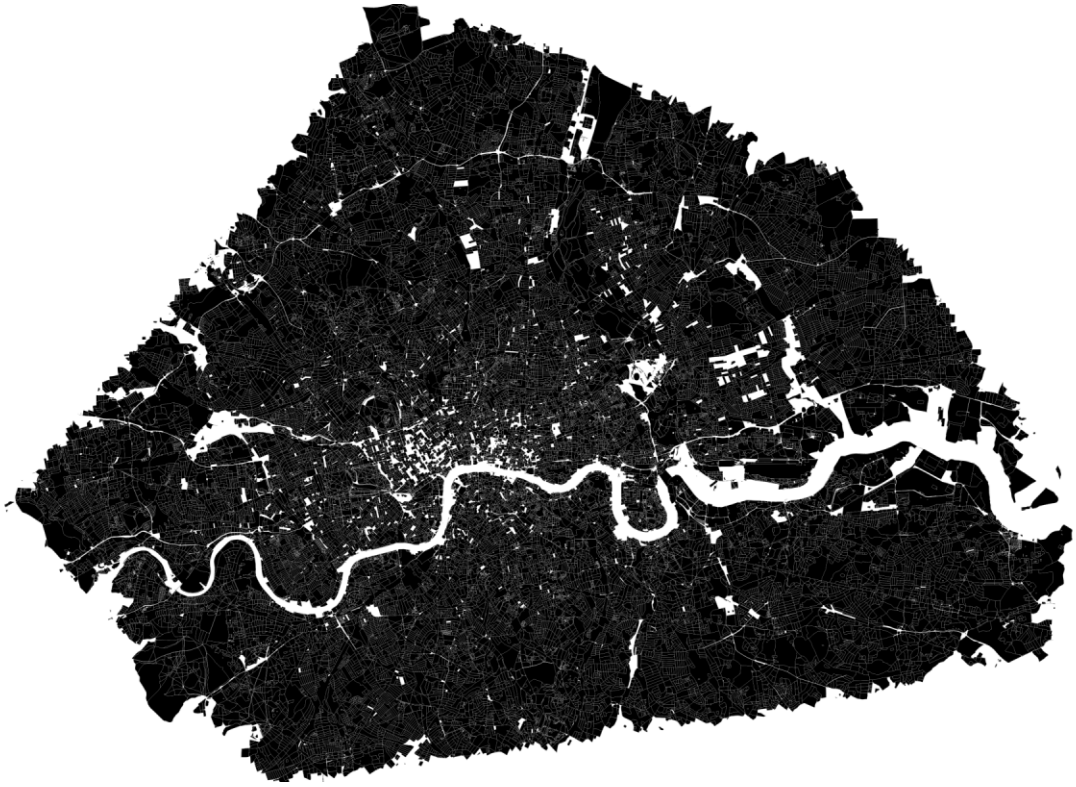


Figure 1: London road network cells
The cells' shape file was kindly provided by Rémi Louf of the Institut de Physique Théorique CEA

2 Data

The data used to perform and optimise the tool has been kindly provided by Rémi Louf of the Institut de Physique Théorique CEA. It is the set of polygons of London blocks obtained by processing the road shape file downloaded from the OpenStreetMap database. It comes in the form of a shape file and it is imported in R with the function readOGR of the package "rgdal".
The method Louf and Barthélemy (2014) have used to extract the polygons is the "always-left-hand" rule: it consists of selecting a random line and move to the next connected line always taking the one most to the left (or right) in case more than one lines are available until the loop is closed.

3 The Tool: FUN_SPDF_Cmi_Index.R

The function FUN_SPDF_Cmi_Index.R has been designed to be used on polygons representing city blocks but it works for any polygon, not necessarily from a city map. It takes as an input a SpatialPolygonDataFrame (a class of geometrical object of the package "sp") and returns the same object with two columns added: the area and the index of compactness C_{MI} for each polygon (FIg. 7 in the Appendix). This first function is the shell that calls all the other functions and puts together the results. It is also used to iterate the calculation in parallel, allowing the possibility of choosing multiple cores, depending on the machine used, in order to speed up the calculation. This passage is necessary if we want to take advantage of multicore processors, otherwise R performs the calculations using a single core only. The function FUN_Polygon_Cmi.R extracts the information to be fed to FUN_calc_Cmi_index.R, which calculates the index C_{MI} for each polygon of the dataset.

4 Calculation of the index of compactness C_MI

The index of compactness C_{MI} is a measure of the dispersion of the components of a shape. It requires the calculation of the second moment of inertia of an area about its centroid g, as follows:

$$I_g = \int z_g^2 \, da \tag{1}$$

The centroid of a shape in strict mathematical terms is the point that minimises the moment of inertia MI for that shape. C_{MI} is calculated as the ratio of the MI of a circle of the same area about its centre, to the MI of the shape about its centroid. Its range is between 0 and 1: a circle has a value of 1 and values closer to 1 mean more compactness. It is calculated as follows:

$$C_{MI} = \frac{A^2}{2\pi I_g} \tag{2}$$

The method proposed by Li et al. (2013) to calculate the MI is a trapezium-based approach (see Fig.2). It consists in taking each edge of the polygon K and building

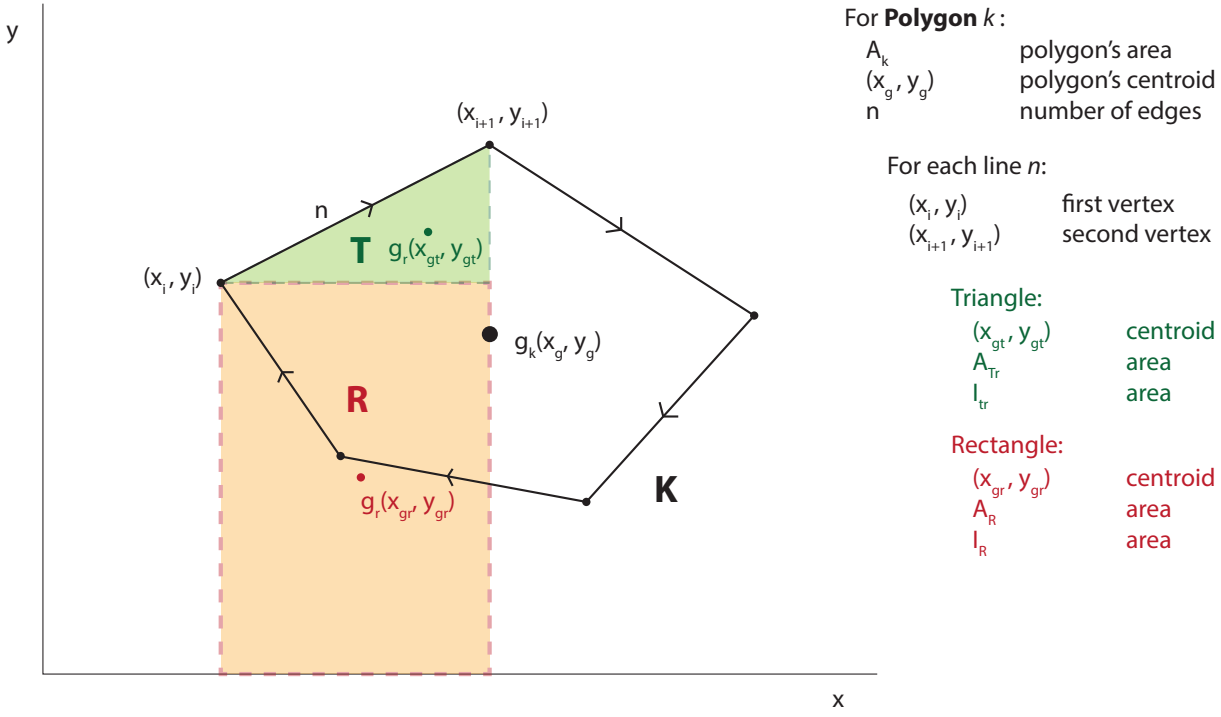


Figure 2: A polygon, showing the trapezia built on one of its edges

The polygon is defined clockwise, in red the area of the rectangle and in green the area of the triangle, both built on one of the polygon's edges.

a trapezium composed by the two vertical projections of the two vertexes on the x-axis, the length of the edge on the x-axis and the edge itself. This trapezium can be decomposed in a triangle and a rectangle: the formulas to calculate the MI of triangles and rectangles are known (5) (8) and these can be recomposed to calculate the MI of the trapezia and the MI of the trapezia to calculate the one of the polygon (12). For the triangle the calculations are as follows:

$$a_{i\_T} = (x_{i+1} - x_i)(y_{i+1} - y_i)/2 \tag{3}$$

$$x^i_{g\_T} = (x_i + 2x_{i+1})/3, \quad y^i_{g\_T} = (y_i + 2y_{i+1})/3 \tag{4}$$

$$I_{i\_T} = a_{i\_T}[(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2]/18 \tag{5}$$

For the rectangle:

$$a_{i\_R} = (x_{i+1} - x_i)y_i \tag{6}$$

$$x^i_{g\_R} = (x_i + x_{i+1})/2, \quad y^i_{g\_R} = y_1/2 \tag{7}$$

$$I_{i\_R} = a_{i\_R}[(x_{i+1} - x_i)^2 + y_i^2]/12 \tag{8}$$

where (x^i_g, y^i_g) is the centroid, a_i is an area and I_i is the MI of each piece. Note that, because areas are calculated using coordinates (3) (6), they are negatives if the piece falls below or completely outside of the polygon. The calculation of the MI for the whole polygon K is obtained by combining all triangles and rectangles built on all of its edges:

$$A_p = \sum_{i=1}^{n}(a_{i\_T} + a_{i\_R}) \tag{9}$$

$$x_g = \sum_{i=1}^{n} \frac{(x^i_{g\_R}a_{i\_R} + x^i_{g\_T}a_{i\_T})}{A_p}, \quad y_g = \sum_{i=1}^{n} \frac{(y^i_{g\_R}a_{i\_R} + y^i_{g\_T}a_{i\_T})}{A_p} \tag{10}$$

According to the parallel axis theorem (Landau et al., 1984), the MI about any point s is related to the MI about the centroid g by:

$$I_s = I_g + d^2da \tag{11}$$

where d is the distance of g from the point s. Adjusting the moments of inertia of all the rectangles and triangles to the polygon's centroid gives I_g:

$$I_g = \sum_{i=1}^{n}(I_{i\_T} + d^2_{i\_T,g}A_{i\_T} + I_{i\_R} + d^2_{i\_R,g}A_{i\_R}) \tag{12}$$

This can than be substituted in (2) to obtain the shape index.

5 Analysis

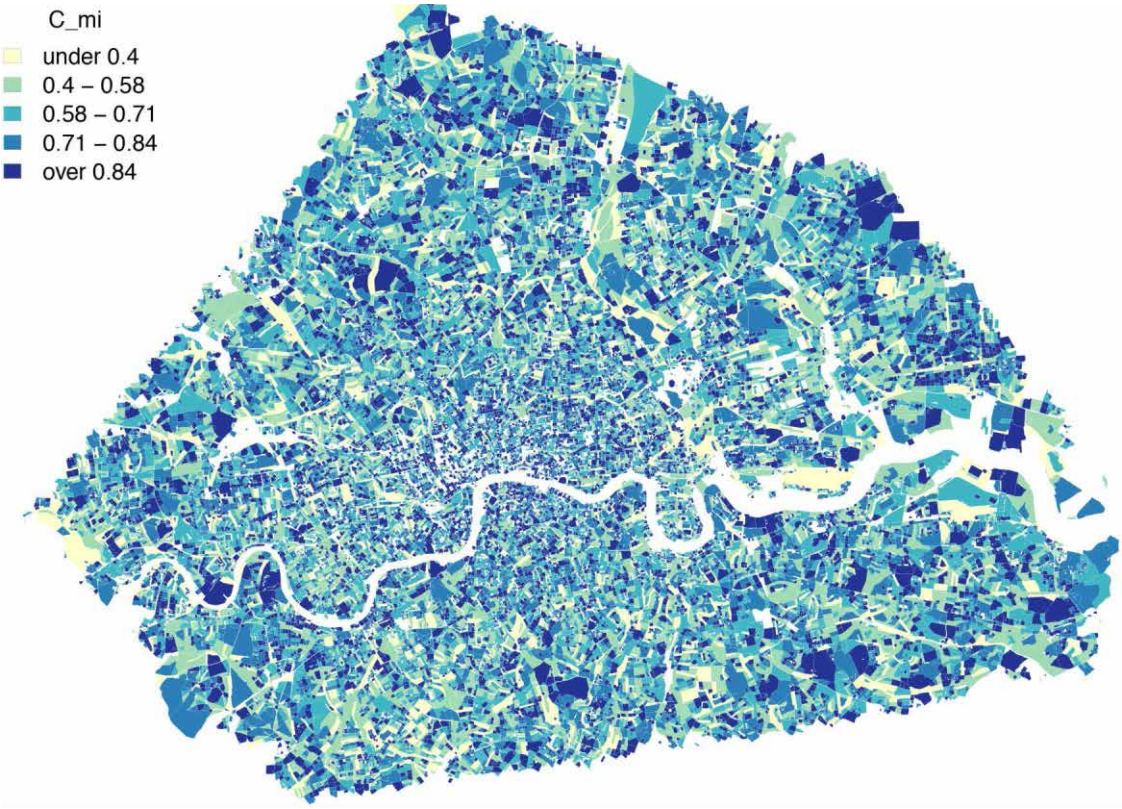


Figure 3: Blocks by C_{MI} index

My hypothesis is that, in historical neighbourhoods that developed incrementally in times when the prevalence of movements would happen by foot, the shape of the blocks would be more compact and the areas would be smaller. The phenomena of urbanisation that peaked during the second half of the 20^{th} century and the increase economic scale incentivised bigger one-off developments with the consequences of lager blocks. The increased reliance on cars as a mean of transportation would also require less efficient connectivity and decrease the demand for compact blocks.

In facts, the figure shows that blocks are larger in the outer parts of the city and smaller in the city centre (Fig.4) but the index of compactness seems to be evenly distributed and to show no sign of spatial pattern (Fig.3).

Figure 5 shows a k-means clustering using the normalised coordinates, the area and the index C_{MI}: there is some sign of clustering of big shapes but they mostly

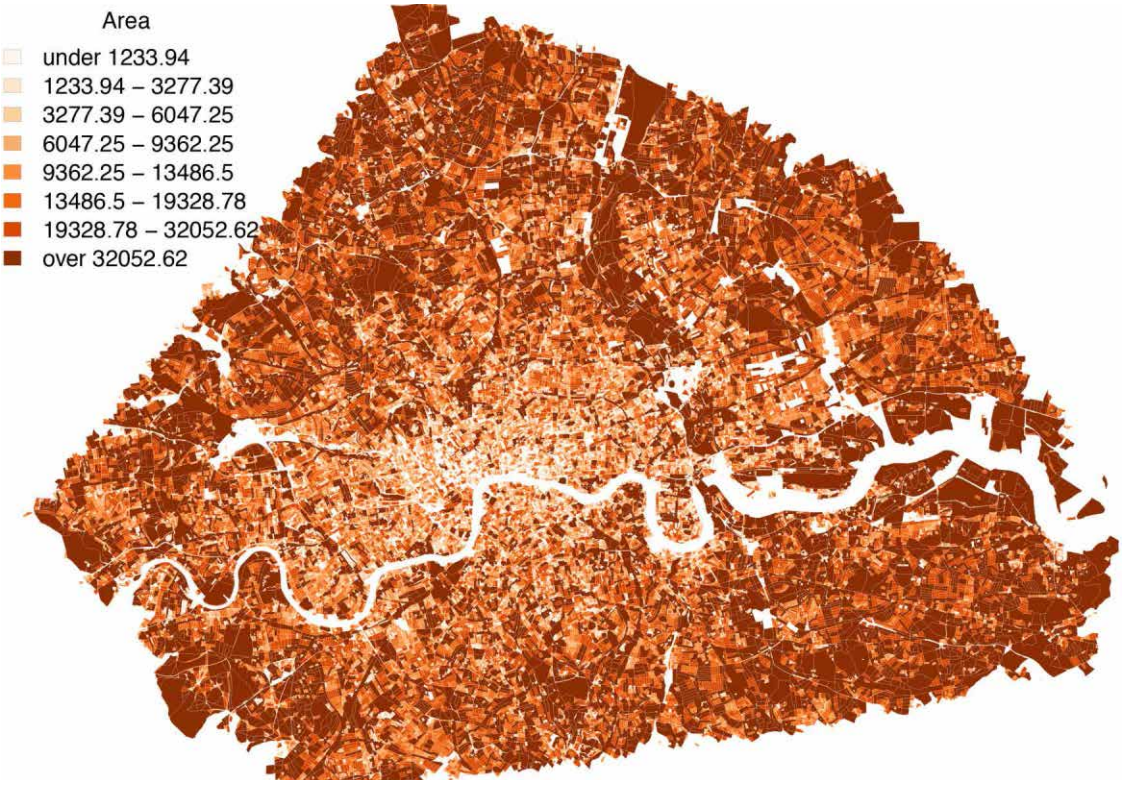


Figure 4: Blocks by area

correspond to parks. The Local Moran I_i index (Fig.6) confirms that there is no pattern and although there is some evidence of both negative and positive I_i values, none of them is statistically relevant.

6 Future research

A quantitative classification of the morphology of different neighbourhoods is no trivial question. I believe the city block per se is not enough information to characterise the road network and that information about the built environment should be included in the analysis. In areas like the centres of Paris or London, where the environment is fully developed, the difference between buildings and city blocks might be minor, but as soon as we move out to the periphery and buildings are more dispersed, this difference might be more substantial. As further research, I would be interested in using the same tool on other cities but including the information about the buildings, possibly adapting the code to calculate a composed density index as suggested by Pont

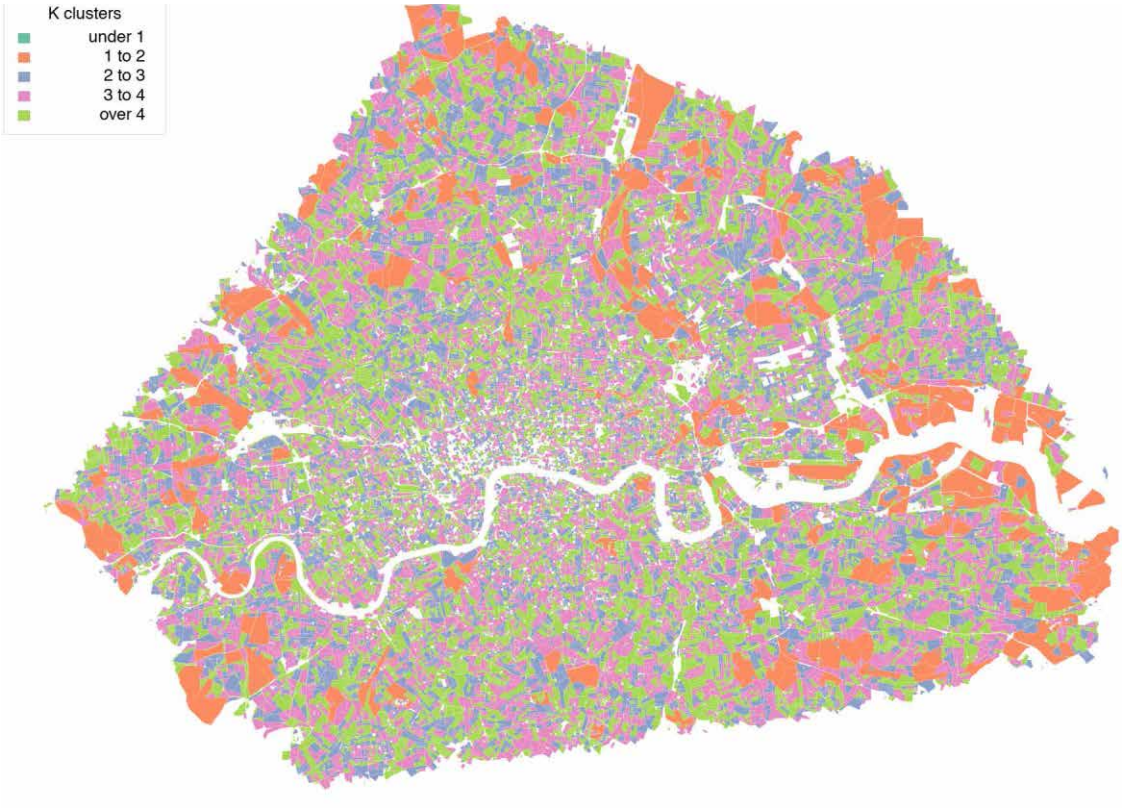


Figure 5: Clustering by coordinates, area and C_{MI} index

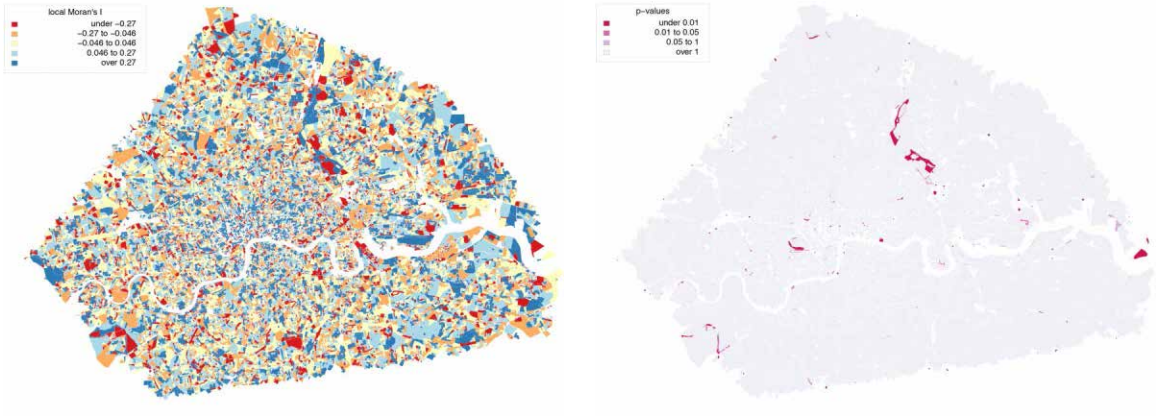


Figure 6: Local Moran's I_i and adjusted p-values

P-values are adjusted according to the Bonferroni method, to account for the large number of observations (more than 40.000 polygons)

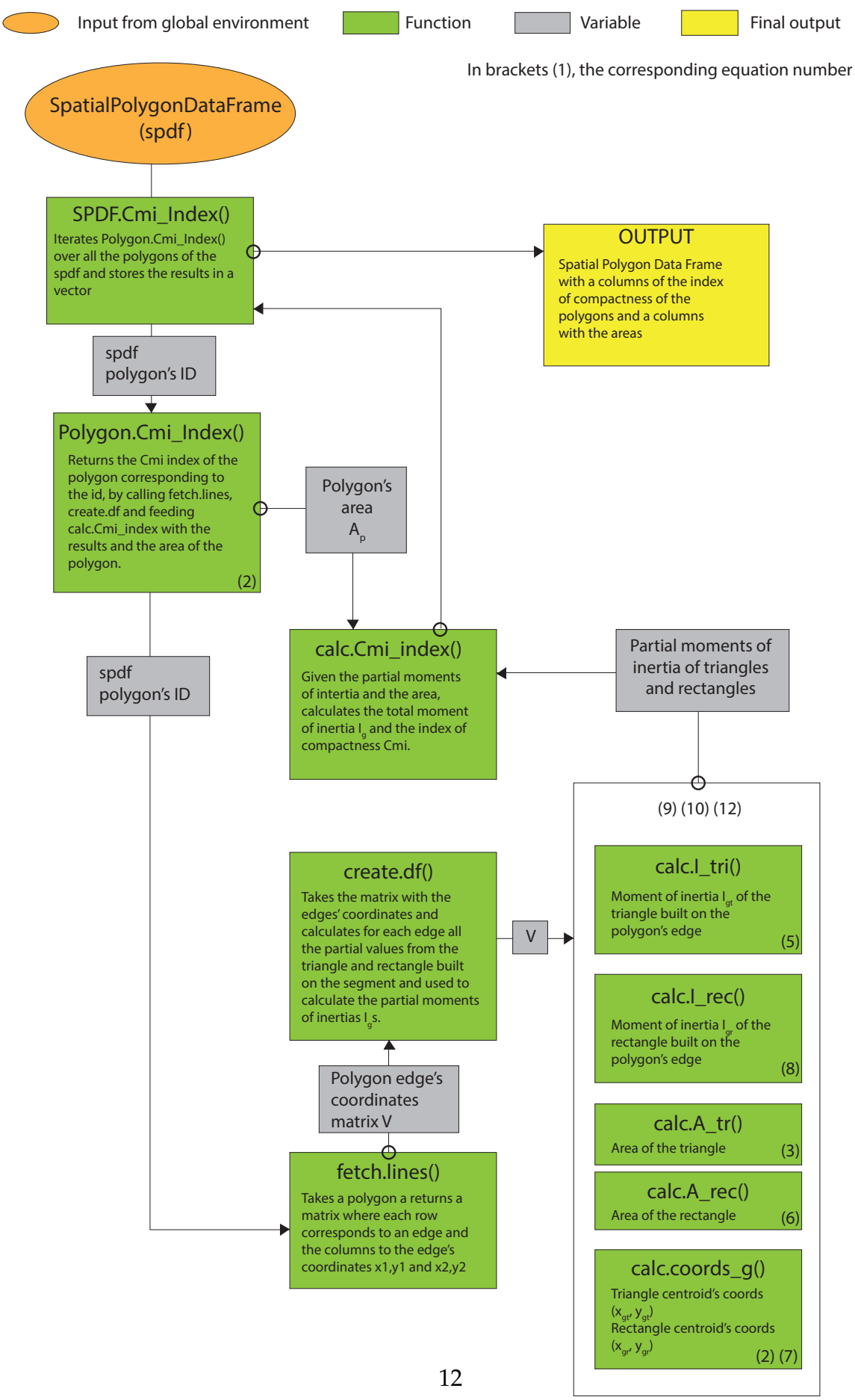
and Haupt (2007). This includes a ground space index, which is the percentage of land covered by buildings; an open-space ratio, which is the amount of non-built space at ground level per floorspace area; and the average number of floors in an area.

7 Appendix

System's Specifications

Please use the "$London\_analysis.R$" to run the code and the analysis. The function and the code has been designed and tested with the following configuration:
sysname: "Darwin"
release: "15.2.0"
version: "$DarwinKernelVersion$15.2.0 : $FriNov$13 19 : 56 : 56PST2015; $root$: $xnu-$ 3248.20.55 2/$RELEASE_X$86_64"
R version 3.2.2 (2015-08-14) "Fire Safety"

Function Diagram


Input from global environment
Function
Variable
Final output
In brackets (1), the corresponding equation number
SpatialPolygonDataFrame (spdf)
SPDF.Cmi_Index()
Iterates Polygon.Cmi_Index() over all the polygons of the spdf and stores the results in a vector
OUTPUT
Spatial Polygon Data Frame with a columns of the index of compactness of the polygons and a columns with the areas
spdf polygon's ID
Polygon.Cmi_Index()
Returns the Cmi index of the polygon corresponding to the id, by calling fetch.lines, create.df and feeding calc.Cmi_index with the results and the area of the polygon.
(2)
Polygon's area A_p
spdf polygon's ID
calc.Cmi_index()
Given the partial moments of intertia and the area, calculates the total moment of inertia I_g and the index of compactness Cmi.
Partial moments of inertia of triangles and rectangles
(9) (10) (12)
create.df()
Takes the matrix with the edges' coordinates and calculates for each edge all the partial values from the triangle and rectangle built on the segment and used to calculate the partial moments of inertias I_gs.
V
calc.I_tri()
Moment of inertia I_{gt} of the triangle built on the polygon's edge
(5)
calc.I_rec()
Moment of inertia I_{gr} of the rectangle built on the polygon's edge
(8)
Polygon edge's coordinates matrix V
calc.A_tri()
Area of the triangle
(3)
calc.A_rec()
Area of the rectangle
(6)
fetch.lines()
Takes a polygon a returns a matrix where each row corresponds to an edge and the columns to the edge's coordinates x1,y1 and x2,y2
calc.coords_g()
Triangle centroid's coords (x_{gt}, y_{gt})
Rectangle centroid's coords (x_{gr}, y_{gr})
(2) (7)

References

Hillier, B. and J. Hanson (1984). *The social logic of space*. Cambridge university press.

Landau, L. D., J. Bell, M. Kearsley, L. Pitaevskii, E. Lifshitz, and J. Sykes (1984). *Electrodynamics of continuous media*, Volume 8. elsevier.

Li, W., M. F. Goodchild, and R. Church (2013). An efficient measure of compactness for two-dimensional shapes and its application in regionalization problems. *International Journal of Geographical Information Science 27*(6), 1227–1250.

Lopes Gil, J. A. (2014). Analyzing the configuration of multimodal urban networks. *Geographical Analysis: an international journal of theoretical geography IF - 1.500 46*(4), 368–391.

Louf, R. and M. Barthélemy (2014). A typology of street patterns. *Journal of the Royal Society Interface 11*(20140924), 1–7.

Marshall, S. (2004). *Streets and patterns*. Routledge.

Pont, M. B. and P. Haupt (2007). The relation between urban form and density. *Urban morphology 11*(1), 62.

A tool for the calculation of the compactness of city blocks in R

BENVGSA3 - GI Stystems and Science

Code in R

Workflow

```
require(rgdal)
require(rgeos)
require(sp)
require(GISTools)
require(RColorBrewer)
require(spdep)



source("./FUN_SPDF_Cmi_Index.R")
source("./FUN_Plot_map_byClassInt.R")
#Read the polygon data
london <- readOGR("../shapes/london/clean", "london")
#Caclulate the C_mi index
london_Cmi <- SPDF.Cmi_Index(london, 6)



attach(london_Cmi@data)
#Check the summary
summary(C_mi)
summary(Area)
pdf("../Figures/hist_C_mi.pdf")
par(lwd = 0.1)
hist(C_mi, breaks = 100, col = "azure4", border = "white")
dev.off()
pdf("../Figures/hist_Area.pdf")
par(lwd = 0.1)
hist(Area[Area < quantile(Area, .99)], breaks = 100,
     col = "gray32", border = "white")
dev.off()
detach(london_Cmi@data)
#####  Plot C_mi values
pdf("../Figures/london_Cmi.pdf")
par(lwd = 0.0001)
plot_spdf_byClassInt(london_Cmi, plot_variable = "C_mi",
                     n = 5, col_pal = "YlGnBu",
                     style_intervals = "quantile",
                     border_col = "white")
dev.off()
#####Plot area values
pdf("../Figures/london_Area.pdf")
par(lwd = 0.0001)
```

```
plot_spdf_byClassInt(london_Cmi, plot_variable = "Area",
                     n = 8, col_pal = "Oranges",
                     style_intervals = "quantile",
                     border_col = "white")
dev.off()

####### Local Moran #####
#Code adapted from the book "An Introduction to R or Spatial Analysis and Mapping"
#by Chris Brunsdon and Lex Comber, 2015
london_lw <- nb2listw(poly2nb(london_Cmi), zero.policy=TRUE)
london_lI <- localmoran(london_Cmi@data$C_mi, london_lw)
london_lI[is.na(london_lI[,1]),1] <- 0
london_lI[is.na(london_lI[,4]),4] <- 0
london_lI[is.na(london_lI[,5]),5] <- 0
#Plot the local moran values
pdf("../Figures/london_LocalMorans.pdf")
par(lwd = 0.0001)
lI_shades <- auto.shading(c(london_lI[,1], -london_lI[,1]),
                          cols = brewer.pal(5, "RdYlBu"))
choropleth(london_Cmi, london_lI[,1],
           shading = lI_shades, lwd = .00001, border = "white")
title("Index of compactness (local Moran's I)", cex.main = 1)
choro.legend(px=681623.5, py=5727840,
             sh = lI_shades, cex = .48, title = "local Moran's I")
dev.off()
##check p-values
pdf("../Figures/london_pvalues.pdf")
par(lwd = 0.0001)
pval_shade <- shading(c(0.01, 0.05, 01),
                      cols = rev(brewer.pal(4, "PuRd")))
choropleth(london_Cmi, london_lI[,5],
           shading = pval_shade, lwd = .00001, border = "white")
title("Index of compactness [Local p-values]", cex.main = 1)
choro.legend(px=681623.5, py=5727840,
             sh = pval_shade, cex = .48, title = "p-values")
dev.off()
##Adjusted p-values Bonferroni
pdf("../Figures/london_adj_pvalues.pdf")
par(lwd = 0.0001)
pval_shade <- shading(c(0.01, 0.05, 01),
                      cols = rev(brewer.pal(4, "PuRd")))
choropleth(london_Cmi, p.adjust(london_lI[,5], method = "bonferroni"),
           shading = pval_shade, lwd = .00001, border = "white")
title("Index of compactness [Adjusted local p-values]", cex.main = 1)
choro.legend(px=681623.5, py=5727840,
             sh = pval_shade, cex = .48, title = "p-values")
dev.off()
#Cluster analysis
coords <- foreach(i = 1:length(london_Cmi), .combine = "rbind") %dopar% london@polygons[i][[1]]@labpt
data <- cbind(london_Cmi@data$C_mi, london_Cmi@data$Area, coords)
data <- scale(data)
source("FUN_wss_plot.R")
wssplot(data)
```

```
k <- 5
clusters <- kmeans(data, centers = k)
london_Cmi@data$Cluster <- clusters$cluster
pdf("../Figures/london_k-means.pdf")
par(lwd = 0.001)
k_shade <- shading(c(1,2,3,4), cols = brewer.pal(5, "Set2"))
choropleth(london_Cmi, london_Cmi@data$Cluster,
           shading = k_shade, lwd = .00001, border = "white")
title("K-mean clusters", cex.main = 1)
choro.legend(px=681623.5, py=5727840,
             sh = k_shade, cex = .48, title = "K clusters")
dev.off()
```

SPDF.Cmi_Index.R

```
##Add a column of C_mi indexes and areas to the polygon dataframe
SPDF.Cmi_Index <- function(spdf, num_cores = 1){
    require(sp)
    require(doParallel)
    source("./FUN_Polygon_Cmi.R", local = TRUE)
    source("./FUN_extract_areas.R", local = TRUE)
    source("./FUN_calc_A_polygon.R", local = TRUE)
    source("./FUN_fetch_lines.R", local = TRUE)
    source("./FUN_calc_A.R", local = TRUE)
    source("./FUN_calc_coords_g.R", local = TRUE)
    source("./FUN_calc_I.R", local = TRUE)
    source("./FUN_create_df.R", local = TRUE)
    source("./FUN_calc_Cmi_index.R", local = TRUE)

    n <- length(spdf@polygons)#fetch total number of polygons
    registerDoParallel(cores = num_cores)#set number of cores
    #Calculate the index of compactness C_mi in parallel
    Cmi_indexes <- foreach(i = 1:n, .combine = "c") %dopar%
        Polygon.Cmi_Index(spdf, i)
    #add a columns with the area of the polygons
    spdf@data$Area <- extract.areas(spdf)
    #add the column with the index to the polygon's
    spdf@data$C_mi <- Cmi_indexes
    spdf <- spdf[spdf@data$C_mi > 0 & spdf@data$C_mi < 1,]
    spdf <- spdf[spdf@data$Area > 0,]
    return(spdf)
}
```

Polygon.Cmi_Index.R

```
#### Given a SpatialPolygonDataFrame and a polygon's ID
#### return the Cmi index of the polygon

Polygon.Cmi_Index <- function(sp, id){
    require(sp)
```

```
    Area_polygon <- sp@polygons[id][[1]]@Polygons[[1]]@area
    V_lines <- fetch.lines(sp, id)
    V_dataframe <- create.df(V_lines)
    C_mi <- calc.Cmi_index(V_dataframe, Area_polygon)
    return(C_mi)
}
```

calc.Cmi_index.R

```
#############################################################################################
calc.Cmi_index <- function(V_df, A_pol){
    Ig <- sum(V_df$Ig_partial)
    C_mi <- (A_pol^2)/(2*pi*Ig) #equation (2)
    return(C_mi)
}
#############################################################################################
```

calc.A_polygon.R

```
#############################################################################################
#Calculate the area of the polygon from its coordinates
calc.A_polygon <- function(sp, id){
    require(sp)
    coords <- sp@polygons[id][[1]]@Polygons[[1]]@coords
    area <- 0
    j <- nrow(coords)-1
    for(i in 1:(nrow(coords)-1)){
        area <- area + (coords[j,1]+coords[i,1])*(coords[j,2]-coords[i,2])
        j <- i
    }
    return(area/2)
}
```

calc.A_tr.R and calc.A_rec.R

```
#############################################################################################
#Given one vector with coordinates x1,y1,x2,y2 the function return
#Areas and MI of the triangle and rectangle for the line between the two vertex
#v[1]=x1
#v[2]=y1
#v[3]=x2
#v[4]=y2
#To be applied to matrix V
#Caclulate area
#area of the triangle
calc.A_tr <- function(v){
    A_tr <- (v[3]-v[1])*(v[4]-v[2])/2 #equation (3)
    return(A_tr)
```

```
}
#area of the rectangle
calc.A_rec <- function(v){
    A_rec <- (v[3]-v[1])*v[2] #equation (6)
    return(A_rec)
}
```

calc.coords_g.R

```
################################################################################
#calculate x_g and y_g coordinates
#specify the required coordinate "x" or "y"
#specify the shape "tri" or "rec"
calc.coords_g <- function(v, x_or_y = "x", shape = "tri"){
    if(x_or_y == "x"){
        #centroid's x_gt of the triangle
        if(shape == "tri"){return((v[1]+2*v[3])/3)} #equation (4)
        #centroid's x_gt of the rectangle
        if(shape == "rec"){return((v[1]+v[3])/2)} #equation (7)
    }
    if(x_or_y == "y"){
        #centroid's y_gr of the triangle
        if(shape == "tri"){return((2*v[2]+v[4])/3)} #equation (4)
        #centroid's y_gr of the rectangle
        if(shape == "rec"){return(v[2]/2)} #equation (7)
    }
}
```

calc.I_tr.R and calc.I_rec

```
################################################################################
#Calculate moment of inertia for
#triangle
calc.I_tr <- function(v){v[5]*((v[3]-v[1])^2 + (v[4]-v[2])^2)/18}  #equation (5)

#rectangle
calc.I_rec <- function(v){v[6]*((v[3]-v[1])^2 + v[2]^2)/12} # #equation (8)
################################################################################
```

create.df.R

```
#Create dataframe to df_MI used to perform the final calculation of the MIg
#Takes as an input the four columns matrix of the coordinates x1,y1 and x2,y2
#of the polygon's edges
create.df <- function(V){
    A_tr <- apply(V, 1, calc.A_tr)#Area triangle
    A_rec <- apply(V, 1, calc.A_rec)#Area rectangle
    #x coords of the triangle's centroid
```

```
    x_gt <- apply(V, 1, calc.coords_g, x_or_y = "x", shape = "tri")
    y_gt <- apply(V, 1, calc.coords_g, x_or_y = "y", shape = "tri")
    x_gr <- apply(V, 1, calc.coords_g, x_or_y = "x", shape = "rec")
    y_gr <- apply(V, 1, calc.coords_g, x_or_y = "y", shape = "rec")
    V2 <- cbind(V, A_tr, A_rec)
    I_tr <- apply(V2, 1, calc.I_tr)#MI of the triangle about its centroid
    I_rec <- apply(V2, 1, calc.I_rec)#MI of the triangle about its centroid
    V_df <- data.frame(line_ID = seq(1, nrow(V)),
                       cbind(V, x_gt, y_gt, A_tr,
                             I_tr, x_gr, y_gr, A_rec, I_rec))
    names(V_df)[2:5] <- c("x1", "y1", "x2", "y2")
    #total area triangle and rectangle
    A <- sum(A_tr + A_rec) #equation (9)
    #Polygon's centroid
    V_df$x_g_partial <- (x_gt*A_tr + x_gr*A_rec)/A #equation (10)
    V_df$y_g_partial <- (y_gt*A_tr + y_gr*A_rec)/A #equation (10)
    x_g <- sum(V_df$x_g_partial)#equation (10)
    y_g <- sum(V_df$y_g_partial)#equation (10)

    #distances of the triangles centroids from the polygon centroid
    V_df$Dist_gt <- sqrt((x_g - x_gt)^2 + (y_g - y_gt)^2)
    V_df$dist2_A_tr <- (V_df$Dist_gt^2)*A_tr #partial of equation (12)
    #distances of the triangles centroids from the polygon centroid
    V_df$Dist_gr <- sqrt((x_g - x_gr)^2 + (y_g - y_gr)^2)
    V_df$dist2_A_rec <- (V_df$Dist_gr^2)*A_rec #partial of equation (12)
    V_df$Ig_partial <- with(V_df, I_tr + dist2_A_tr + I_rec + dist2_A_rec) #equation (12)
    return(V_df)
}
```

extract.areas

```
#Input a SpatialPolygonDataFrame and extract all the polygons' areas
extract.areas <- function(sp){
    require(sp)
    n <- length(sp@polygons)
    areas <- rep(0, n)
    for(i in 1:n){
        areas[i] <- sp@polygons[i][[1]]@Polygons[[1]]@area
    }
    return(areas)
}
```

fetch.lines.R

```
##############################################################################################
#Takes a SpatialPolygonDataFrame and a polygon's id and
#creates the matrix V with coordinates x1,y1,x2,y2
#of each segment of the polygon
fetch.lines <- function(sp, id){
    require(sp)
```

```
    #extract the coordinates of the vertex
    coords <- sp@polygons[id][[1]]@Polygons[[1]]@coords
    n <- nrow(coords)
    #if the area of the polygon from the coordinates is negative,
    #invert the coordinates
    if(calc.A_polygon(sp, id) < 0){coords <- coords[n:1, ]}
    v <- cbind(coords[1:(n-1),], coords[2:n,])
    return(v)
}
```

FUN_Plot_map_byClassInt.R.R

```
##### Plot a colour map ########################################################
### Show all the colour schemes available
#display.brewer.all()

plot_spdf_byClassInt <- function(spdf, #SpatialPolygonDataFrame
                      plot_variable = c("Area", "C_mi"), #Pick the variable to be plotted
                      n_breaks = 5, #Number of breaks
                      col_pal = "Set3", #Colour palette
                      border_col = "cornsilk3",
                      style_intervals = "jenks"){ #Pick the style for classIntervals
    require(sp)
    require(rgeos)
    require(maptools)
    require(RColorBrewer)
    require(classInt)
    col_palette <- brewer.pal(n_breaks, col_pal)
    breaks <- classIntervals(spdf@data[,plot_variable], n = n_breaks, style = style_intervals)
    breaks <- breaks$brks
    plot(spdf,
         col=col_palette[findInterval(spdf@data[,plot_variable],
                                      breaks,
                                      all.inside = TRUE)],
         axes=F, asp=T, lwd = 0.001, border = border_col)
    legend(x=681623.5, y=5727840, legend=leglabs(round(breaks, 2)),
           fill=col_palette, bty="n", cex = .7, title = plot_variable)
}
```

wssplot.R

```
###Plot of the total within-groups sums of squares against
###the number of clusters in a K-means solution
###From the book "R in action", by Robert Kabacoff 2011
###http://www.r-statistics.com/2013/08/k-means-clustering-from-r-in-action/

wssplot <- function(data, nc=15, seed=1234){
    wss <- (nrow(data)-1)*sum(apply(data,2,var))
    for (i in 2:nc){
        set.seed(seed)
```

```
    wss[i] <- sum(kmeans(data, centers=i)$withinss)}
  plot(1:nc, wss, type="b", xlab="Number of Clusters",
       ylab="Within groups sum of squares")}
```