The huff-tools package (version 0.6)

Michail Pavlis, Les Dolega, Alex Singleton

Department of Geography and Planning, University of Liverpool

November 10, 2014

Abstract

This vignette describes the use of the huff-tools package in R. The huff-tools package implements the basic Huff algorithm to estimate the probability of customers from a given point of origin patronising a retail centre. The package provides functions to calculate the Euclidean and shortest road distance. In addition, it can be used to manipulate the estimated Huff probabilities e.g. to obtain the highest probabilities for each point of origin or to extract the primary and secondary catchment areas from a SpatialPolygonsDataFrame object.

1. Introduction

The huff-tools package implements the basic Huff algorithm:

$P_{ij} = A_j^{\alpha} * D_{ij}^{-\beta} / \Sigma(A_j^{\alpha} * D_{ij}^{-\beta})$

Where P_{ij} the probability of customers from a location i patronising a retail area j, A_j the attractiveness parameter of the retail area j, alpha (α) the exponent of the attractiveness parameter, D_{ij} a measure of distance between the point of origin i and the point of destination j and beta (β) the exponent of distance.

The package requires the installation of the rgdal, rgeos, igraph, dplyr and fastshp R packages. rgdal is used to create and manipulate spatial objects while rgeos provides the gDistance function to calculate Euclidean distances. igraph provides tools to convert a road network to graph object and consequently to estimate the shortest path using the Dijkstra algorithm with the shortest.paths function. It is also used to extract the connected part of the road network using the decompose.graph function. The huff-tools package requires the road network to be imported in R as a list with the fastshp package (see examples below). Finally, dplyr is used for fast data manipulation and data transformation.

2. Using the huff-tools package

2.1. The get_connected function

If the user intends to calculate the shortest road distance it might be a good idea to verify

first that the road network at hand is connected. The get_connected function can be used to extract the most connected part of the road network and optionally to visualise the disconnected parts of the road network. It is up to the user to decide whether or not the road network is properly connected. As a general guideline if the majority of the disconnected parts of the network are on the edges of the study area then the road network is most likely adequately connected, if on the other hand a large part of the central area was removed then it might be useful to correct the road network with standard GIS tools available in ArcGIS or QGIS. The function accepts five arguments. These are:

1) The path to the shapefile (path_to_shp),

2) The name of the shapefile without the .shp extention (shp_name_in),

3) The path to save the returned road network as shapefile (path_to_save),

4) The name of the new shapefile without the .shp extention (shp_name_out)

5) True or False to specify whether or not to plot the disconnected parts of the road network (plot_output).

The last three arguments are optional. The third and the fourth arguments can be used if the user wants to save the connected part of the road network as a shapefile, **by default the function returns the (connected) road network as a list** so as to be used in other functions of the huff-tools package. Bear in mind that depending on the size of the road network it might take a considerable amount of time to convert the returned list to a SpatialPolygonsDataFrame object and plot it in R. Finally, the default value for plotting the output is False, change it to True if you want to plot the results. An example of using the get_connected function is provided below.

```
# Import the huff-tools package
source("/home/r_script/huff-tools.r")
# Use the get_connected function to obtain the most connected part of the road network,
# select to save the connected road network as a shapefile, plot the output
roads2 ← get_connected(path_to_shp = "/home/data/input", shp_name_in = "roads", path_to_save
                       = "/home/data/results",  shp_name_out = "roads_conn", plot_output = True)
```

The selected and removed parts of the network can be seen in Figure 1, in total there are 84 disconnected parts (including the main part of the network returned by the function). Roads marked with red were removed (you might have to zoom in to be visible). Most of the unconnected parts are in the periphery of the road network so the returned part can be used to estimate the Huff probabilities.

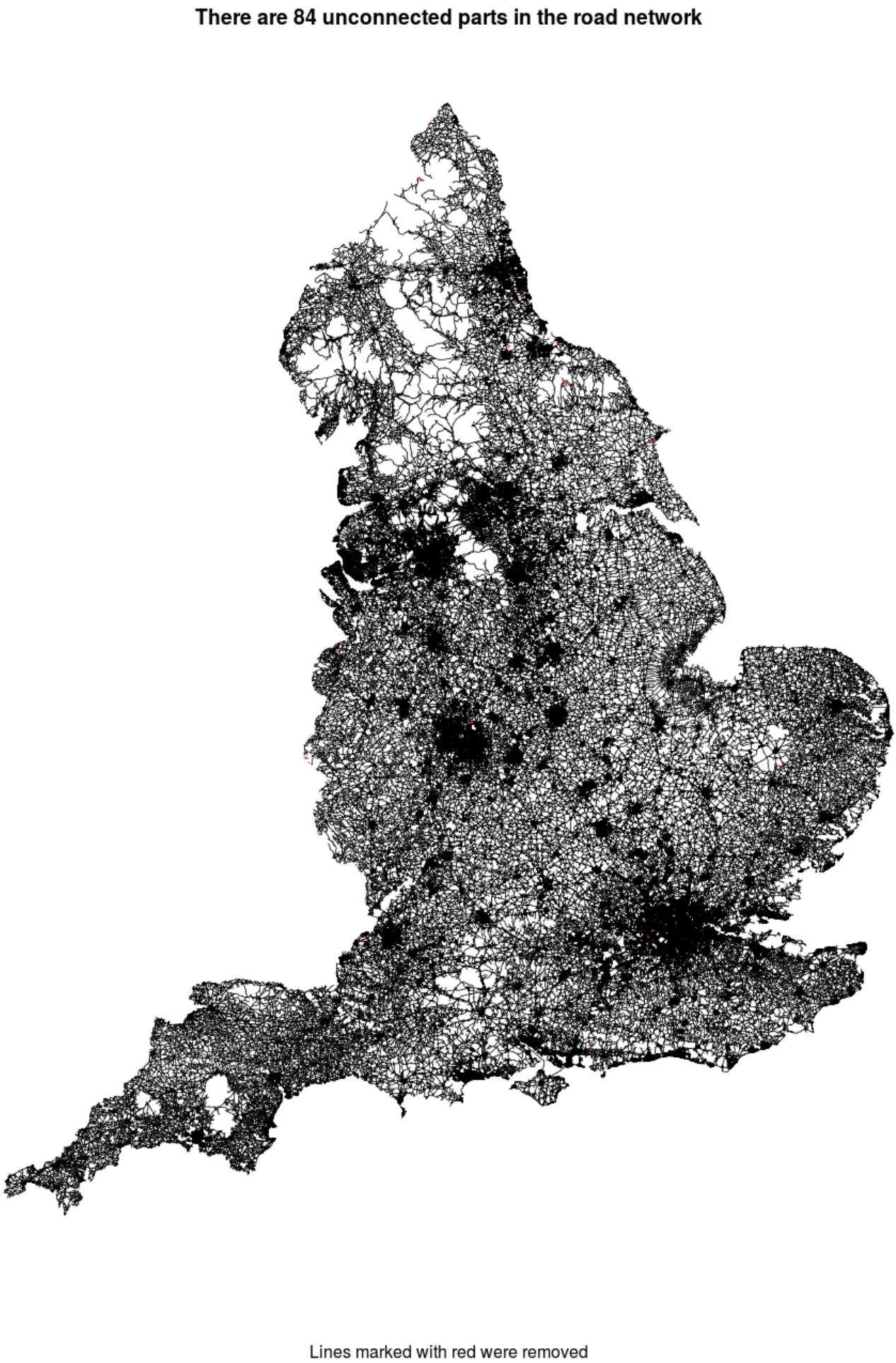
There are 84 unconnected parts in the road network
Lines marked with red were removed

Figure 1

2.2. The euclidean_distance function

If a road network is not available it is possible to obtain a measure of distance by calculating the Euclidean distance between the points of origin and the points of destination. The euclidean_distance function accepts 6 arguments:

1) A shapefile of destinations (destinations) e.g. shopping centres, town centres, it can be either a shapefile of polygons or points. If polygons, their centroids will be used as points of destination.

2) A vector of **unique** names or numbers for each destination point (destinations_name). If there are NA values the function will exit. The length of the destinations_name vector should be equal to the number of points of destination.

3) A shapefile of origins (origins).

4) A vector of **unique** names for these origins (origins_name), all requirements mentioned for the destinations apply for the origins as well.

5) A number of closest destinations to each origin (neighbours_constraint).

6) Distance threshold within which to select the closest destinations from each origin (distance_constraint).

The last two arguments are optional. All the shapefiles should be projected in the same reference system, and the unit of the returned distance is kilometres. The euclidean_distance function returns a data frame where the first two columns represent the unique pairwise combinations of origin and destination names while the third column is the Euclidean distance. An example of using the euclidean_distance function is provided below.

```
# load the huff-tools library
source("/home/r_script/huff-tools.r")
# import the destinations shapefile using rgdal
destinations ← readOGR("/home/data/input", "destinations")
# import the origins shapefile
origins ← readOGR("/home/data/input", "origins")
# apply the euclidean_distance function,
# select only distances within 50 km and the 10 closest destinations from each origin
distance ← euclidean_distance(destinations = destinations, destinations_name = destinations@data$Name, origins = origins, origins_name = origins@data$Name, neighbours_constraint =10, distance_constraint = 50)
```

2.3. The shortest_distance function

The shortest_distance function can be used to calculate the shortest road distance when a road network shapefile is available. The function accepts 7 arguments.

1) A shapefile of destinations (destinations) e.g. shopping centres, town centres, it can be either a shapefile of polygons or points. If polygons, their centroids will be used as points of destination. It is also possible to use multiple points for each destination so as, for example, to calculate the

distance to the boundary of retail centres. In order to extract the coordinates of the boundary of the destinations you can use the polygons_to_points function (see example below). If multiple points are provided for each destination, the shortest_distance function will calculate all the pairwise distances between these points and the origin points and then aggregate the data based on the destination name so as for each unique destination-origin pair to select the minimum distance.

2) A vector of names (not necessarily unique as explained above) or numbers for each destination point (destinations_name). If there are NA values the function will exit. The length of the destinations_name vector should be equal to the number of points of destination.

3) A shapefile of origins (origins).

4) A vector of unique names for these origins (origins_name).

5) A road network (roads) in the form of a list as obtained from the read.shp function of the fastshp library, or from the output of the get_connected function presented in section 2.1.

6) A number of closest destinations to each origin (neighbours_constraint).

7) Distance threshold within which to select the closest destinations from each origin (distance_constraint).

The output of the function is a data frame where the first two columns represent all the unique pairwise combinations between origins and destinations and the third column is the shortest road distance (in kilometres). The shortest_distance function can be used as follows:

```
# load the huff_tools library
source(“/home/r_script/huff-tools.r”)
# import a road network as a list object using the read.shp function
roads ← read.shp(“/home/data/input/roads.shp”, “list”)
# import the boundaries of retail centres using rgdal
destinations ← readOGR(“/home/data/input”, “destinations_boundaries”)
# obtain the coordinates of the destination boundaries using the polygons_to_points function
# two inputs: the shapefile of boundaries (polys) and the name of a field in the shapefile (name)
# with entries being unique names for each boundary
destinations_pnt ← polygons_to_points(polys = destinations, name= “Name”)
# import the origins shapefile
origins ← readOGR(“/home/data/input”, “origins”)
# run the shortest_distance function
distance ← shortest_distance(destinations = destinations_pnt, destinations_name = destinations_pnt@data$Names, origins = origins, origins_name = origins@data$Name, roads = roads)
```

2.4. The huff_basic function

The huff_basic function estimates the probability of customers from a given point of origin patronising a particular retail centre. There are 6 arguments for this function:

1) A vector of destinations names (destinations_name).

2) A vector of attractiveness scores (destinations_attractiveness) for the destinations.

3) A vector of origins names (origins_name).

4) A vector of distances (distance) between each unique pair of origins-destinations.

5) A vector or scalar for the alpha exponent (alpha).

6) A vector or scalar for the beta exponent (beta).

The first, second and fourth arguments, should be obtained either from the output of euclidean_distance function or from the output of the shortest_distance function. For the alpha and beta arguments if they are vectors they should be of the same length as the other vectors passed into the function. By default, alpha is equal to 1 and beta is equal to two. If no inputs are provided for either beta or alpha, the default value will be used. The output of the huff_basic function is a data frame with 6 columns, the first two provide all the pairwise combinations of origins and destinations names, the third the distance between the two, the fourth the value of the numerator of the Huff algorithm, the fifth the value of the denominator of the Huff algorithm and the sixth the Huff probabilities.

In the following example assume that we have calculated the shortest distances and saved the output as distances.csv. Also assume that we have a csv file with the attractiveness score for the destinations and that we have ranked the destinations in four classes based on their size which we will use to develop the beta values. More specifically, based on our ranking classification we will assign a beta value equal to 1.2 for very large retail centres, 1.4 for large retail centres, 1.6 for medium size retail centres and finally a beta value of 2.0 for small retail centres. Remember that for the beta exponent of the distance decay factor the smaller the value the slower the attractiveness decay as function of distance. We will import the csv files as data frames in R, merge them based on the name of the destinations and then develop the alpha and beta exponents of the Huff algorithm. For the alpha exponent we will assume that it is more likely for someone to patronise a store if it is within 0.5 kilometres from the point of origin and therefore we decide to assign an alpha value of 2 for distances below that threshold and an alpha value of 1 to all other distances. So in R:

```
# load the huff_tools library
source(“/home/r_script/huff-tools.r”)
# import the distances.csv file
distances ← read.csv(“/home/data/input/distances.csv”)
# import the attractiveness.csv file
attractiveness ← read.csv(“/home/data/input/attractiveness.csv”)
# get the beta values
attractiveness$beta ← with(attractiveness,
                    ifelse(Rank == 1, 1.2, ifelse(Rank == 2, 1.4, ifelse(Rank == 3, 1.6, 2))))
# merge the two data frames
distances ← merge(distances, attractiveness[,c("Name", "destinations_attractiveness", "beta")],
              by.x = "destinations_name", by.y = "Name")
```

```
# get the alpha values
alpha ← with(distances, ifelse(distance <= 0.5, 2, 1))
# estimate the huff probabilities
huff_probs ← huff_basic(destinations_name = distances$destinations_name, destinations_attractiveness = distances$destinations_attractiveness, origins_name = distances$origins_name, distance = distances$distance, alpha =  alpha, beta = distances$beta)
```

2.5. Group by origin and extract by probabilities

The result of the huff_basic function is a table of probabilities of all pairwise combinations of origins-locations. It is often required, however, to extract from the table the highest Huff probability for each origin (indicating the most likely destination from that origin). This might be needed, for example, in order to delineate the primary and secondary catchment areas of the destination points. One way to do this would be by grouping the data based on the name of each origin area and then extracting a number of origin areas based on the huff probability. So for example if we wanted to extract all of the origin areas that had the highest probability of patronising the provided destinations, this could be achieved using the select_by_probs function. This function can accept 6 arguments, namely:

1) The table of huff probabilities (huff_probs), which is the output of the huff_basic function.

2) The number of highest probabilities per origin (nr).

3) The name of the origins field in the table (origins_name).

4) The name of the destinations field (destinations_field).

5) The name of the distance field (distance).

6) The name of the huff probabilities field (huff_probabilities).

If the names of the fields of the table created by the huff_basic function have not been altered then the last 4 arguments can be omitted as the default names will be used. So extracting the highest probability for each origin:

```
# load the huff_tools library
source("/home/r_script/huff-tools.r")
# open the huff probabilities csv file
huff_probs ← read.csv("/home/data/results/huff_probs.csv")
# extract the highest probability per origin
higher_probs ← select_by_probs(huff_probs  = huff_probs, nr = 1)
```

Then it is possible to join the extracted probabilities with a shapefile based on a common "origins" field. So assuming that we applied the Huff model using the 2011 LSOAs as the origin locations, we could use the 2011 LSOA boundary shapefile provided by the Office for National Statistics to map the extracted Huff probabilities like so:

```
origins ← readOGR("/home/input/GIS", "lsoa_11")
origins@data ← data.frame(origins@data, higher_probs[match(origins@data$LSOA11CD,
                                higher_probs$origins_name), ])
```

writeOGR(origins, “/home/results/GIS”, “higher_probs”, driver = “ESRI Shapefile”)

2.6. Extracting the catchment area based on the Huff probabilities

Following the previous step, the catchment area of the destinations can be extracted from the shapefile created earlier using the get_catchment function. The get_catchment function accepts 9 arguments and outputs a shapefile. The arguments are:

1) The previously created shapefile (poly).
2) The path to save the output of the function (path_to_save).
3) A name for the output shapefile (shp_name_out).
4) The name of the destination (town_centre).
5) The upper bound of the Huff probability (huff_prob_upper).
6) The lower bound of the Huff probability (huff_prob_lower).
7) The name of the “origins” field in the shapefile (origins_name).
8) The name of the “destinations” field in the shapefile (destinations_name).
9) The name of the huff probabilities field in the shapefile (huff_probability).

The last 3 arguments are optional as a default value can be used, but if you saved the previously created shapefile and you are importing it again in R, you should either rename the fields or change the default values in the function as the names of the fields of the shapefile have been truncated. Assuming that the primary catchments are the LSOAs with huff_probability greater than 0.5 we can extract them for each destination as follows:

```
#  load the huff_tools library
source("/home/r_script/huff-tools.r")
# load the shapefile of the selected huff probabilities that was created earlier
huff ← readOGR("/home/results/GIS", "higher_probs")
# rename the truncated field names to the default names used by the function,
# assuming that the first field provides origin names, the second field provides destination names
# and the third field provides the huff probabilities
names(huff) ← c("origins_name", "destinations_name", "huff_probability")
# Get the unique destination names for which we will create primary catchment areas
towncentres ← unique(huff@data$destinations_name)
# Apply the get_catchment function in a loop to extract catchments
for (tn in towncentres){
    get_catchment(poly = huff, path_to_save = “/home/results/GIS/primary”, shp_name_out = tn,
                  town_centre = tn, huff_prob_upper = 1, huff_prob_lower = 0.5)
}
```