R Package gdistance: Distances and Routes on Geographical Grids

Jacob van Etten

Abstract

The R package **gdistance** provides classes and functions to calculate various distance measures and routes in heterogeneous geographic spaces represented as grids. Least-cost distances as well as more complex distances based on (constrained) random walks can be calculated. Also the corresponding routes or probabilities of passing each cell can be calculated. The package implements classes to store the data about the probability or cost of transitioning from one cell to another on a grid in a memory-efficient sparse format. These classes make it possible to manipulate the values of cell-to-cell movement directly, which offers flexibility and the possibility to use asymmetric values. The novel distances implemented in the package are used in geographical genetics (applying circuit theory), but may also have applications in other fields of geospatial analysis.

Keywords: geospatial analysis, landscape genetics, circuit theory, connectivity, dispersal, travel, least-cost path, least-cost distance, random walk, R.

1. Introduction: The Crow, the Wolf, and the Drunkard

This article describes **gdistance**, a package written for use in the R environment (R Development Core Team 2012). It provides functionality to calculate various distance measures and routes in heterogeneous geographic spaces represented as grids. Distances are fundamental to geospatial analysis (Tobler 1970). Distances and routes are closely related concepts in geography. The most commonly used geographic distance measure is the great-circle distance, which represents the shortest line between two points, taking into account the curvature of the earth. The great-distance distance could be conceived of as the distance measured along a route of a very efficient traveller who knows where to go and has no obstacles to deal with. In common language, this is referred to as a distance 'as the crow flies'.

When travel is less goal-directed and affected by the environment, grid-based distances and routes become relevant. The least-cost distance is implemented in most GIS software and mimics route finding 'as the wolf runs'[1], taking into account obstacles and the local 'friction' of the landscape. The random walk, which is also called the drunkard's walk, has no predetermined destination, so a destination point is hit by accident. The distance travelled to hit the destination point is a measure used to characterize dispersal processes in geography.

Package **gdistance** was designed to determine such grid-based distances and routes and to make it possible to use these measures in combination with other functionality available within R. It has functionality that is comparable to other software such as ArcGIS Spatial Analyst (Mc-

[1]There are some variations on this expression, involving mostly other animals, or telecom cables.

Coy and Johnston 2002), GRASS GIS (GRASS Development Team 2012), and CircuitScape (McRae *et al.* 2008). The gdistance package also contains specific functionality for geographical genetic analyses, not found in other software yet. The package implements measures to model dispersal histories first presented by van Etten and Hijmans (2010). Example 2 below introduces with an example how **gdistance** can be used in geographical genetics.

2. Theory

Calculations are done in various steps in **gdistance**. At first, this tends to be somewhat confusing for those who are used to distance and route calculations in GIS software, which tend to be done in a single step. However, an important goal of **gdistance** is to make the calculations of distances and routes more flexible, which also makes the package somewhat more complicated to use. Users, therefore, need to have a basic understanding of the theory behind distance and route calculations.

Calculations of distances and routes start with raster data. In geospatial analysis, rasters are rectangular, regular grids that represent continuous data over geographical space. Cells arranged in rows and columns and each holds a value. A raster is accompanied by metadata that indicate the resolution, extent and other properties.

Distance and route calculations on rasters rely on graph theory. So as a first step, rasters are converted into graphs by connecting cell centres to each other, which become the nodes in the graph. This can be done in various ways (Figure 1).

- Cells can be connected orthogonally to their four immediate neighbours, which is called the von Neumann neighbourhood.
- Cells can be connected with their eight orthogonal and diagonal nearest neighbours, the Moore neighbourhood. The resulting graph is called the 'king's graph', because it reflects all the legal movements of the king in chess. This is the most common and often only way to connect grids in GIS software.
- Connecting in 16 directions combines king's and knight's moves. The function r.cost in the software GRASS (GRASS Development Team 2012) has this as an option, which inspired its implementation in **gdistance**. The section on distance transforms in de Smith *et al.* (2009) also discusses 16-cell neighbourhoods. Connecting in 16 directions may increase the accuracy of the calculations.

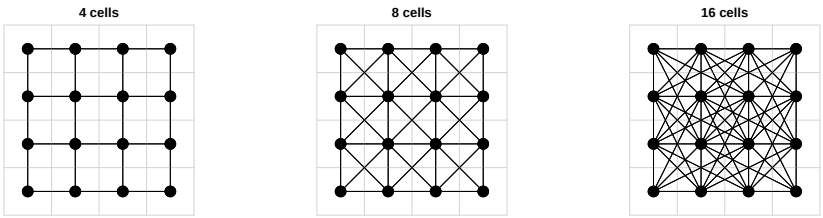

Figure 1: Rasters can be converted into graphs in different ways.

When the raster is converted into a graph, weights are given to each edge (connections between nodes). These weights correspond to different concepts. In most GIS software, distance

analyses are done with calculations using cost, friction or resistance values. In graph theory, weights can also correspond to conductance (1/resistance), which is equivalent to permeability (a term used in landscape ecology). The weights can also represent probabilities of transition.

Graphs are mathematically represented as matrices to do calculations. Matrices can include transition probability matrices, adjacency matrices, resistance/conductance matrices, Laplacian matrices, among others. In **gdistance**, we refer collectively to matrices that represent graphs as 'transition matrices'. These transition matrices are the central object in the package; all distance calculations need one or more transition matrices as an input.

In **gdistance**, usually conductance rather than resistance values are expected in the transition matrix. An important advantage of using conductance is that it makes it possible to use computer memory very efficiently, using so-called *sparse* matrices. Sparse matrices only record the non-zero values and information about their location in the matrix. In most cases, cells are connected only with adjacent cells, and the conductance for direct connections between remote cells is zero. Consequently, most values in a conductance matrix are zero and occupy no memory in a sparse matrix. Another reason is that in graph theory the analogy of a graph with an electrical circuit is often used (see below). For most calculations based on this analogy, the conductance matrix or the transition probability matrix are used.

The calculation of the actual edge weights is usually based on the values of the grid cells, which represents a property of the landscape. For instance, from a grid with altitude, a value for the ease of walking can be calculated for each transition between cells. It is possible to create asymmetric matrices, in which the conductance from i to j is not always the same as the conductance from j back to i. This is relevant, among other things, for modelling travel in hilly terrain, as shown in Example 1 below. On the same slope, a downslope traveler experiences less friction than an upslope traveler. In this case, the function to calculate conductance values is non-commutative: $f(i, j) \neq f(j, i)$.

A problem that arises in grid-based modelling is the choice of weights that should be given to diagonal edges in proportion to orthogonal ones. For least-cost path distance and routes, this is fairly straightforward: weights are given in proportion to the distances between the cell centres. In a grid in which the orthogonal edges have a length of 1, the diagonal edges are $\sqrt{2}$ long. McRae (2006) also applies this same idea to random walks. However, as Birch (2006) explains, this is generally not the best discrete approximation of a random walk dispersal process in continuous space. Different orthogonal and diagonal weights could be considered based on his analytical results.

For random walks on longitude-latitude grids, there is an additional consideration to be made. Considering the eight neighbouring cells in a Moore's neighbourhood, the three cells that are located nearer to the equator are larger in area than the three cells that are closer to the nearest pole, as the meridians approach each other. So the latter should have a slightly lower probability of being reached during a random walk from the central cell. More theoretical work is needed to investigate possible solutions to this problem. For projected grids, we can safely ignore this distortion problem.

When the transition matrix has been constructed, different algorithms to calculate distances and routes are applied.

- The least-cost distance mimics route finding 'as the fox runs', taking into account obstacles and the local 'friction' of the landscape. The least-cost path between two cells on the grid and the associated distance can be obtained with Dijkstra's algorithm or

similar algorithms.

- A second type of route-finding is the random walk, which has no predetermined destination (a 'drunkard's walk'). Commute distance represents the random walk commute time, which is the average number of edges traversed during a random walk from an starting point on the graph to a destination point and back again to the starting point (Chandra *et al.* 1996). Resistance distance reflects the average travel *cost* during this walk (McRae 2006). When taken on the same graph these two measures differ only in their scaling (Kivimäki *et al.* 2012). Commute and resistance distances are calculated using the analogy with an electrical circuit (see Doyle and Snell 1984, for an introduction). The algorithm that **gdistance** uses to calculate commute distances was developed by Fouss *et al.* (2007).
- Randomised shortest paths are an intermediate form between shortest paths and Brownian random walks, introduced by Saerens *et al.* (2009). van Etten and Hijmans (2010) applied randomised shortest paths in geospatial analysis (and see Example 2 below).

3. Raster Basics

Analyses in **gdistance** start with one or more rasters. For this, it relies on another R package, **raster** Hijmans and van Etten (2012). The **raster** package provides comprehensive geographical grid functionality. Here, we briefly discuss this package, referring the reader to the documentation of **raster** itself for more information.

The following code shows how to create a raster object.

```
R> r <- raster(ncol=3,nrow=3)
R> r[] <- 1:ncell(r)
R> r

class       : RasterLayer
dimensions  : 3, 3, 9  (nrow, ncol, ncell)
resolution  : 120, 60  (x, y)
extent      : -180, 180, -90, 90  (xmin, xmax, ymin, ymax)
coord. ref. : +proj=longlat +datum=WGS84
data source : in memory
names       : layer
values      : 1, 9  (min, max)
```

The first line loads the package. The second line creates a simple raster with 3 columns and 3 rows. The third line assigns the values 1 to 9 as the values of the cells. The resulting object is inspected in the fourth line. As can be seen in the output, the object does not only hold the cell values, but also holds metadata about the geographical properties of the raster.

It can also be seen that this is an object of the class `RasterLayer`. This class is for object that hold only one layer of grid data. There are other classes which allow more than one layer of data: `RasterStack` and `RasterBrick`. Collectively these classes are referred to as `Raster*`.

A class is a static entity designed to represent objects of a certain type using 'slots', which each hold different information about the object. Both **raster** and **gdistance** use so-called S4 classes, a formal object-oriented system in R. An advantage of using classes is that data and metadata stay together and remain coherent. Consistent use of classes makes it more difficult to have contradictions in the information about an object. For instance, changing the number of rows of a grid also has an effect on the total number of cells. Information about these two types of information of the same object could become contradictory if we were allowed to change one without adjusting the other. Classes make operations more rigid to avoid such contradictions. Operations that are geographically incorrect, such as adding the values of two rasters of different projections, are detected by first comparing the content of the slots that hold the projection information of the two objects. If the information about the projections used for the rasters are not compatible, the operation will produce an error.

Classes also make it easier for the users to work with complex data and functions. Since so much information can be stored in a consistent way in objects and passed to functions, these functions need fewer options. Functions can deduce from the class of the object that is given to it, what it needs to do. The use of classes, if well done, tends to produce cleaner, better readable, and more consistent scripts.

One important thing to know about **raster** is how grid data are stored internally in `Raster*` objects. Cell numbers in rasters go from left to right and from top to bottom. The 3 x 3 raster we just created with its cell numbers is shown in Figure 2.

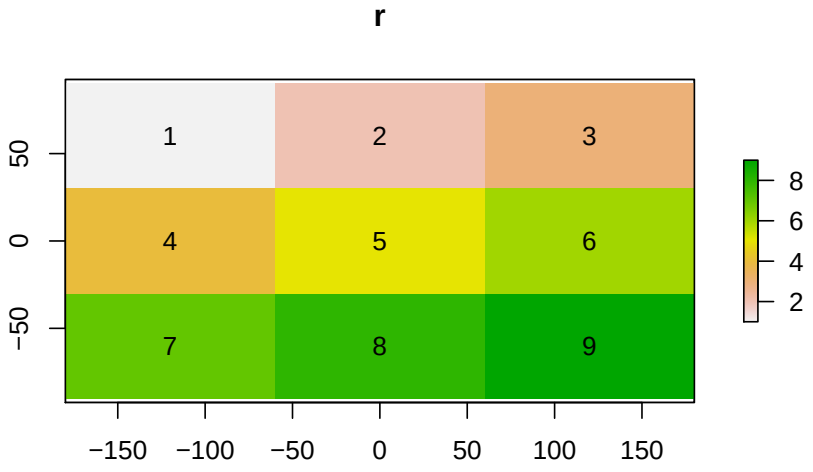

Figure 2: Cell numbers of a 3 x 3 raster.

Figure 2 can be made with this code.

```
R> plot(r, main="r")
R> text(r)
```

4. `Transition*` Classes

As explained in Section 2 on the theory behind **gdistance**, transition matrices are the backbone of the package. The central classes in **gdistance** are `TransitionLayer` and `TransitionStack`. Most operations have an object of one of these classes either as input and sometime also as their output.

`Transition*` objects can be constructed from an object of class `Raster*`. The class `Transition*` takes the necessary geographic references (projection, resolution, extent) from the original `Raster*` object. It also contains a matrix which represents a transition from one cell to another in the grid. Each row and column in the matrix represents a cell in the original `Raster*` object. Row 1 and column 1 in the transition matrix corresponds to cell 1 in the original raster, row 2 and column 2 to cell 2, and so on. For instance, the raster we just created would produce a 9 x 9 transition matrix with rows and columns numbered from 1 to 9 (see Figure 3 below).

The matrix is stored in a sparse format, as discussed in Section 2. The package **gdistance** makes use of sparse matrix classes and methods from the package `Matrix`, which gives access to fast procedures implemented in the C language (Bates and Maechler 2012).

The construction of a `Transition*` object from a `Raster*` object is straightforward. We can define an arbitrary function to calculate the conductance values from the values of each pair of cells to be connected.

In the following chunk of code, the `RasterLayer` created earlier is used. Then we set all its values to unit. The next line makes a `TransitionLayer`, setting the transition value between each pair of cells to the mean of the two cell values that are being connected. The `directions` argument is set to 8, which connects all cells to their 8 neighbours (Moore neighbourhood).

```
R> library("gdistance")
R> r[] <- 1
R> tr1 <- transition(r, transitionFunction=mean, directions=8)
```

If we inspect the object we created, we see that the resulting `TransitionLayer` object keeps much information from the original `RasterLayer` object.

```
R> tr1
```

```
class       : TransitionLayer
dimensions  : 3, 3, 9  (nrow, ncol, ncell)
resolution  : 120, 60  (x, y)
extent      : -180, 180, -90, 90  (xmin, xmax, ymin, ymax)
coord. ref. : +proj=longlat +datum=WGS84
values      : conductance
matrix class: dsCMatrix
```

To make an asymmetric transition matrix, the `symm` argument in `transition` needs to be set to `FALSE`.

```
R> r[] <- runif(9)
R> ncf <- function(x) max(x) - x[1] + x[2]
```

```
R> tr2 <- transition(r, ncf, 4, symm=FALSE)
R> tr2

class       : TransitionLayer
dimensions  : 3, 3, 9  (nrow, ncol, ncell)
resolution  : 120, 60  (x, y)
extent      : -180, 180, -90, 90  (xmin, xmax, ymin, ymax)
coord. ref. : +proj=longlat +datum=WGS84
values      : conductance
matrix class: dgCMatrix
```

From the 'matrix class' we can deduce if the matrix is symmetric or not. These classes are defined in the package **Matrix** (Bates and Maechler 2012). The class `dsCMatrix` is for matrices that are symmetric. The class `dgCMatrix` holds an asymmetric matrix.

Different mathematical operations can be done with `Transition*` objects. This makes it possible to flexibly model different components of landscape friction.

```
R> tr3 <- tr1*tr2
R> tr3 <- tr1+tr2
R> tr3 <- tr1*3
R> tr3 <- sqrt(tr1)
```

Operations with more than one object require that the different objects have the same resolution and extent.

Also, it is possible to extract and replace values in the matrix using indices, in a similar way to the use of indices with simple matrices.

```
R> tr3[cbind(1:9,1:9)] <- tr2[cbind(1:9,1:9)]
R> tr3[1:9,1:9] <- tr2[1:9,1:9]
R> tr3[1:5,1:5]

5 x 5 sparse Matrix of class "dgCMatrix"

[1,] .         0.4532410 0.5412002 0.9307281 .
[2,] 1.4742440 .         0.6291593 .         0.1387396
[3,] 1.3862849 0.4532410 .         .         .
[4,] 0.9967569 .         .         .         0.1387396
[5,] .         0.7677424 .         1.7227166 .
```

The functions `adjacent` (from **raster**) and `adjacencyFromTransition` (from **gdistance**) can be used to create indices. Example 1 below gives an example.

Some functions require that `Transition*` objects do not contain any isolated 'clumps', islands that are not connected to the rest of the raster cells. This can be avoided when creating `Transition*` objects, for instance by giving conductance values between all adjacent cells a small minimum value. It can be checked visually if there are any clumps. There are several ways to visualize a `Transition*` object. For the first method, you can extract the transition

matrix with function `transitionMatrix`. This gives a sparse matrix which can be vizualized with function `image`. This shows the rows and columns of the transition matrix and indicates which has a non-zero value, which represents a connection between cells (Figure 3).

```
R> image(transitionMatrix(tr1))
```

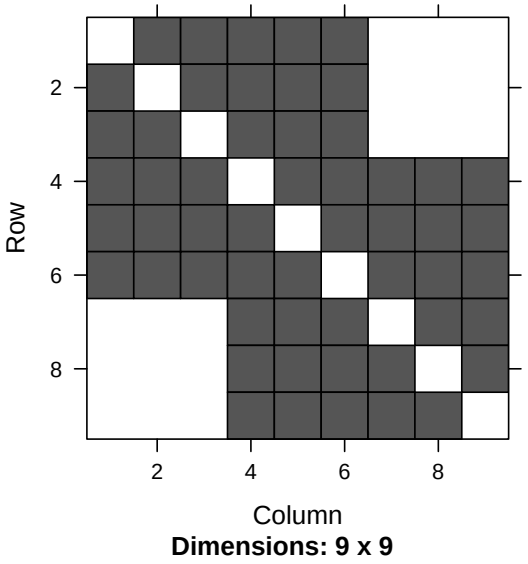

Figure 3: Visualizing a TransitionLayer with function `image`.

Figure 3 shows which cells are connected to each other. A close observer of Figure 3 may wonder why even cell 1 is connected to 5 different cells, as this cell is located in the upper left corner of the original grid. This is explained by the extent of the grid. Since it covers the whole world, the outer meridians (180 and -180 degrees) touch each other. The software takes this into account and as a result the cells in the extreme left column are connected to the extreme right column.

Figure 3 shows which cells contain non-zero values, but gives no further information about levels of conductance. This can be visualized by transforming the transition matrix back into a raster. To summarize the information in transition matrix, we can take means or sums across rows or columns, for instance. You can do this with function `raster`. Applied to a `TransitionLayer`, this function converts it to a `RasterLayer`. For the different options see `method?raster("TransitionLayer")`. The default, shown in Figure 4, takes the column-wise means of the non-zero values.

5. Correcting Transition Matrix Values

The function `transition` calculates transition values based on the values of adjacent cells in the input raster. However, diagonal neighbours are more remote from each other than orthogonal neighbours. Secondly, on equirectangular (longitude-latitude) grids, West-East

```
R> plot(raster(tr3), main="raster(tr3)")
```

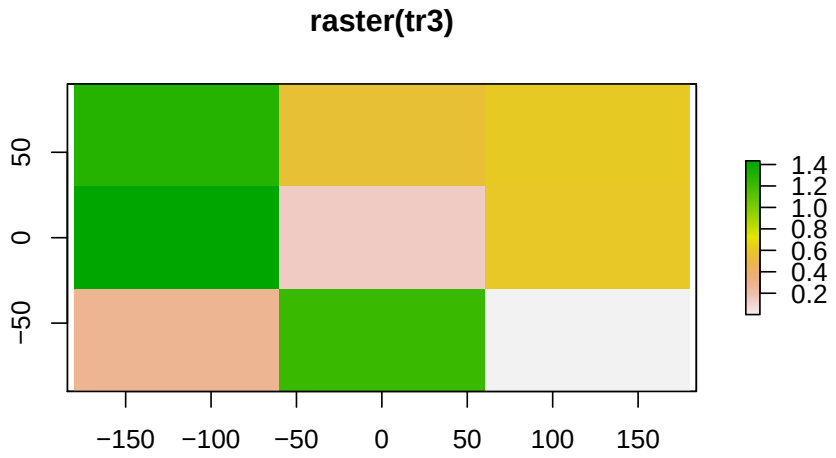

Figure 4: Visualizing a TransitionLayer using the function `raster`.

connections are longer at the equator and become shorter towards the poles, as the meridians approach each other. Therefore, the values in the matrix need to be corrected for these two types of distortion. Both types of distortion can be corrected by dividing each conductance matrix value by the distance between cell centres. This is what function `geoCorrection` does.

```
R> tr1C <- geoCorrection(tr1, type="c", multpl=FALSE)
R> tr2C <- geoCorrection(tr2, type="c", multpl=FALSE)
```

For least-cost type distances and routes, this works fine. However, as explained in Section 2 above, this does not work equally well for commute distances (random walks). The function geoCorrection corrects this distortion by multiplying the North-South transition values with the cosine of the average latitude of the cell centres. This type of correction is done by setting the argument `type` to `"r"`.

```
R> r3 <- raster(ncol=18, nrow=9)
R> r3 <- setValues(r3, runif(18*9)+5)
R> tr3 <- transition(r3, mean, 4)
R> tr3C <- geoCorrection(tr3, type="c", multpl=FALSE, scl=TRUE)
R> tr3R <- geoCorrection(tr3, type="r", multpl=FALSE, scl=TRUE)
```

As mentioned in Section 2, the effect of these distortions and corrections needs more research.

The argument `scl` is set to `TRUE` to get reasonable values. If the values are too large, commute distance and randomized shortest path functions will not work well.

When similar `Transition*` objects with equal resolution and extent need to be corrected repetitively, computational effort may be reduced by preparing an object that only needs to

be multiplied with the `Transition*` object to produce a corrected version of it. The following chunk of code is equivalent to the previous one.

```
R> CorrMatrix <- geoCorrection(tr3, type="r", multpl=TRUE, scl=TRUE)
R> tr3R <- tr3 * CorrMatrix
```

Object `CorrMatrix` is only calculated once. It can be multiplied with `Transition*` objects, as long as they have the same extent, resolution, and directions of cell connections. We need to take special care that the geo-correction multiplication matrix (`CorrMatrix`) contains all non-zero values that are present in the `Transition*` object with which it will be multiplied (`tr3` in this case).

6. Calculating Distances

Only now that we have the corrected `Transition*` object we can calculate distances between points. It is important to note that all distance functions require a `Transition*` object with *conductance* values, even though distances will be expressed in 1/conductance (friction or resistance) units (see Section 3).

To calculate distances, we need to have the coordinates of point locations. This is done by creating a two-column matrix of coordinates. Functions will also accept a `SpatialPoints` object or, if there is only one point, a vector of length two.

```
R> sP <- cbind(c(-100, 100, -100), c(-50, 50, 50))
```

Calculating a distance matrix is straightforward now.

```
R> costDistance(tr3C, sP)
```

```
          1         2
2 1.5105050
3 0.8672558 0.9873723
```

```
R> commuteDistance(tr3R, sP)
```

```
          1         2
2 1091.0908
3 1011.4789  990.0745
```

```
R> rSPDistance(tr3R, sP, sP, theta=1e-12, totalNet="total")
```

```
         [,1]     [,2]     [,3]
[1,]  0.00000 61.83572 57.28111
[2,] 62.60077  0.00000 56.44307
[3,] 58.07582 56.47273  0.00000
```

The `costDistance` function relies on the package **igraph** (Csardi and Nepusz 2006) for the underlying calculation. It gives a symmetric or asymmetric distance matrix, depending on the `TransitionLayer` that is used as input.

Commute distance represents the random walk commute time, which represents the number of cells traversed on the trip (Chandra *et al.* 1996).

`rSPDistance` gives the cost incurred during the same walk (theta approaches zero, so the walk is nearly random). By summing the corresponding off-diagonal elements (Dij + Dji), we obtain the commute costs. In this case, the commute costs are only slightly higher than (and proportional to) the commute distances. This is because the TransitionLayer object has been scaled, so transition costs are close to unit for each step. So the total number of steps and the total distance are in the same order.

7. Dispersal Paths

To determine dispersal paths of a (constrained) random walk, we use the function `passage`. This function can be used for both random walks and randomised shortest paths. The function calculates the number of passages through cells before arriving in the destination cell. Either the total or net number of passages can be calculated. The net number of passages is the number of passages that are not reciprocated by a passage in the opposite direction.

Figure 5 shows the probability of passage through each cell, assuming randomised shortest paths with the parameter theta set to 3.

```
R> origin <- SpatialPoints(cbind(0, 0))
R> rSPraster <- passage(tr3C, origin, sP[3,], theta=3)
```

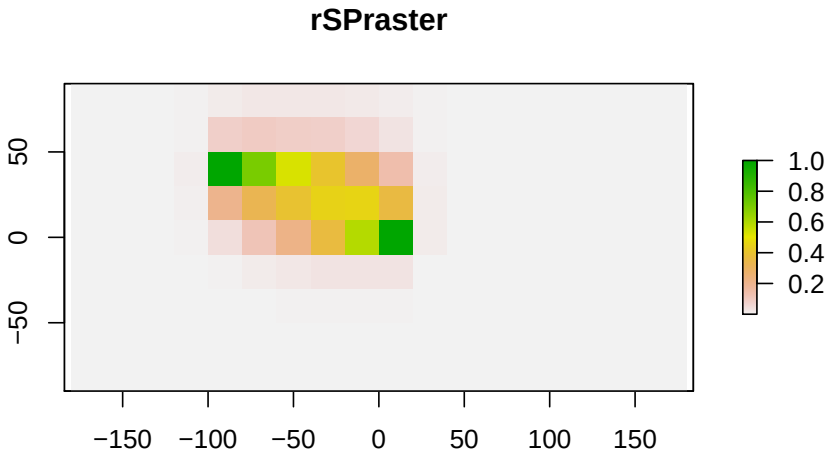

Figure 5: Probability of passage.

8. Path Overlap and Non-Overlap

One of the specific uses, for which package gdistance was created, is to look at trajectories coming from the same source (van Etten and Hijmans 2010). The degree of coincidence of two trajectories can be visualized by multiplying the probabilities of passage (Figure 6). With a more complex formula, we can approximate the non-overlapping part of the trajectory (Figure 7).

```
R> r1 <- passage(tr3C, origin, sP[1,], theta=1)
R> r2 <- passage(tr3C, origin, sP[3,], theta=1)
R> rJoint <- min(r1, r2) #Figure 6
R> rDiv <- max(max(r1, r2) * (1 - min(r1, r2)) - min(r1, r2), 0) #Figure 7
```

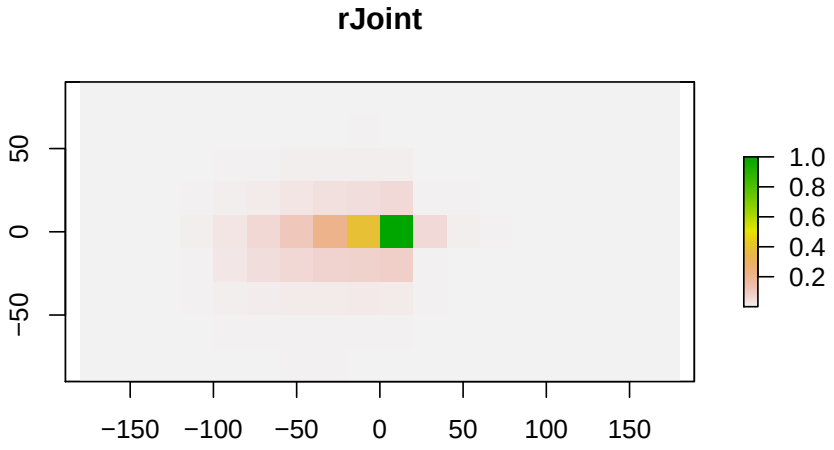

Figure 6: Overlapping part of the two routes.

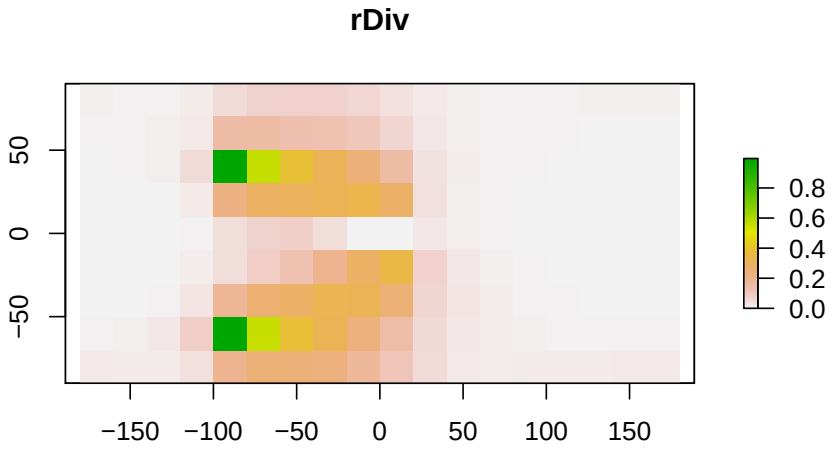

Figure 7: Non-overlapping part of the two routes.

With the function `pathInc` we can calculate measures of path overlap and non-overlap for a large number of points. These measures can be used to predict patterns of diversity if these are due to dispersal from a single common source (van Etten and Hijmans 2010). If the argument type contains two elements (divergent and joint), the result is a list of distances matrices.

```
R> pathInc(tr3C, origin, sP)

$function1layer1
         1        2
2 1.033639
3 1.166870 1.146882

$function2layer1
         1        2
2 1.393100
3 1.125216 1.196323
```

9. Example 1: Hiking around Maunga Whau

The previous examples were somewhat theoretical, based on randomly generated values. More realistic examples serve to illustrate the various uses that can be given to this package.

Determining the fastest route between two points in complex terrain is useful for hikers. Tobler's Hiking Function provides a rough estimate for the the maximum hiking speed given the slope of the terrain (Tobler 1993). The maximum speed of off-path hiking (in m/s) is:

$speed = exp(-3.5 * abs(slope + 0.05))$

Note that the function is not symmetric around 0 (see Figure 8).

We use the Hiking Function to determine the shortest path to hike around the volcano Maunga Whau (Auckland, New Zealand). First, we read in the altitude data for the volcano. This is a geo-referenced version of a R base dataset (see `?volcano`).

```
R> r <- raster(system.file("external/maungawhau.grd",
+    package="gdistance"))
```

The Hiking Function requires the slope as input.

$slope = difference in height / distance travelled$

The units of height and distance should be identical. Here, we use meters for both. We calculate the height differences between cells first. Then we use the function `geoCorrection` to divide by the distance between cells.

```
R> heightDiff <- function(x){x[2] - x[1]}
R> hd <- transition(r,heightDiff,8,symm=FALSE)
R> slope <- geoCorrection(hd, scl=FALSE)
```

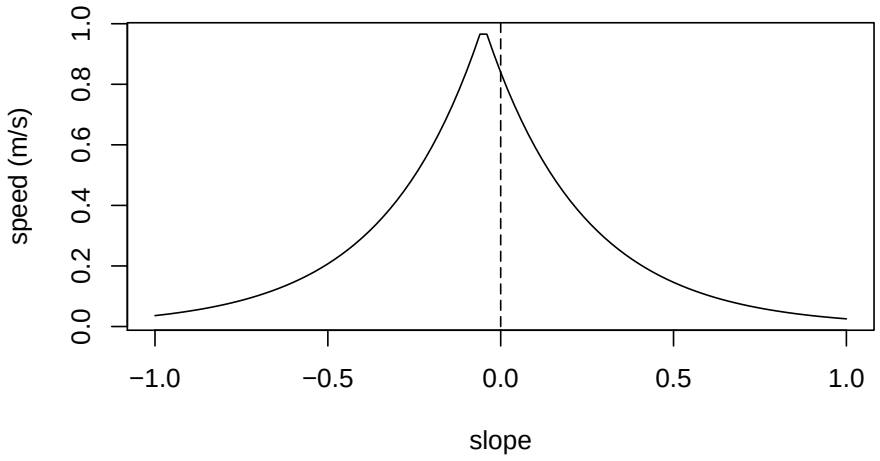

Figure 8: Tobler's Hiking Function.

Subsequently, we calculate the speed. We need to exercise special care, because the matrix values between non-adjacent cells is 0, but the slope between these cells is not 0! Therefore, we need to restrict the calculation to adjacent cells. We do this by creating an index for adjacent cells (`adj`) with the function `adjacent`. Using this index, we extract and replace adjacent cells, without touching the other values.

```
R> adj <- adjacent(r, cells=1:ncell(r), pairs=TRUE, directions=8)
R> speed <- slope
R> speed[adj] <- exp(-3.5 * abs(slope[adj] + 0.05))
```

Now we have calculated the speed of movement between adjacent cells. We are close to having the final conductance values. Attainable speed is a measure of the ease of crossing from one cell to another on the grid. However, we also need to take into account the distance between cell centres. Travelling with the same speed, a diagonal connection between cells takes longer to cross than a straight connection. Therefore, we use the function `geoCorrection` again!

```
R> x <- geoCorrection(speed, scl=FALSE)
```

This gives our final 'conductance' values.

What do these values mean? The function `geoCorrection` divides the values in the matrix between the distance between cell centres. So, with our last command we calculated this:

$conductance = speed/distance$

This looks a lot like a measure that we are more familiar with:

$traveltime = distance/speed$

In fact, the conductance values we have calculated are the *reciprocal* of travel time.

$1/traveltime = speed/distance = conductance$

Maximizing the reciprocal of travel time is exactly equivalent to minimizing travel time!

Now we define two coordinates, A and B, and determine the paths between them. We test if the quickest path from A to B is the same as the quickest path from B back to A. The following code creates the shortest paths.

```
R> A <- c(2667670,6479000)
R> B <- c(2667800,6479400)
R> AtoB <- shortestPath(x, A, B, output="SpatialLines")
R> BtoA <- shortestPath(x, B, A, output="SpatialLines")
```

And this code was used to make Figure 9.

```
R> plot(r, main="")
R> lines(AtoB, col="red", lwd=2)
R> lines(BtoA, col="blue")
R> text(A[1]-10,A[2]-10,"A")
R> text(B[1]+10,B[2]+10,"B")
```

A small part of the A-B (red) and B-A (blue) lines in Figure 9 do not overlap. This is a consequence of the asymmetry of the Hiking Function.

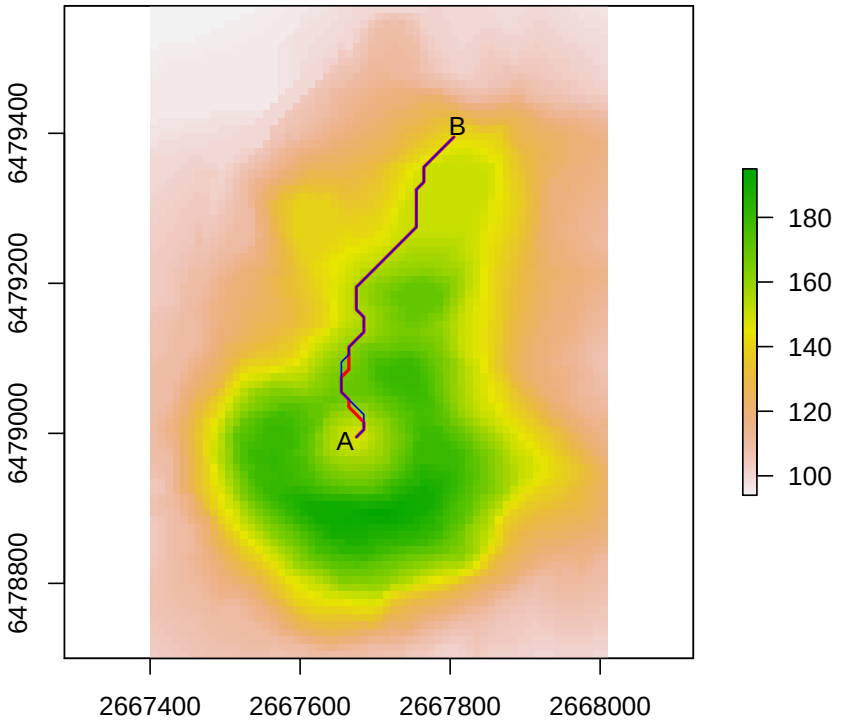

Figure 9: Quickest hiking routes on Maunga Whau (A to B is red, B to A is blue).

10. Example 2: Geographical Genetics

The direct relation between genetic and geographic distances is known as *isolation by distance* (Wright 1943). Recent work has expanded this relationship to random movement in heterogeneous landscapes (McRae 2006). Also, the geography of dispersal routes can explain observed geospatial patterns of genetic diversity. For instance, diffusion from a single origin (Africa) explains much of the current geographical patterns of human genetic diversity (Ramachandran *et al.* 2005). As a result, the mutual genetic distance between a pair of humans from different parts from the globe depends on the extent they share their prehistoric migration history.

Within a single continent, however, human genetic diversity may have to do with more recent events. Let's look at diversity in Europe, using the data presented by Balaresque *et al.* (2010). Within Europe, genetic diversity is often thought to be a result of the migration of early Neolithic farmers from Anatolia (Turkey) to the west.

First we read in the data, including the coordinates of the populations (see Figure 10) and mutual genetic distances.

```
R> Europe <- raster(system.file("external/Europe.grd",
+    package="gdistance"))
R> Europe[is.na(Europe)] <- 0
R> data(genDist)
R> data(popCoord)
R> pC <- as.matrix(popCoord[c("x","y")])
```

Then we create three geographical distance matrices. The first corresponds to the great-circle distance between populations. The second is the least-cost distance between locations. Travel is restricted to the land mass. The third is the commute distance (using the same conductance matrix), which is related to effective resistance between points if we conceive of the grid as an electrical circuit (Chandra *et al.* 1996; McRae 2006).

```
R> geoDist <- pointDistance(pC, longlat=TRUE)
R> geoDist <- as.dist(geoDist)
R> Europe <- aggregate(Europe,3)
R> tr <- transition(Europe, mean, directions=8)
R> trC <- geoCorrection(tr, "c", scl=TRUE)
R> trR <- geoCorrection(tr, "r", scl=TRUE)
R> cosDist <- costDistance(trC,pC)
R> resDist <- commuteDistance(trR, pC)
R> cor(genDist,geoDist)

[1] 0.5962655

R> cor(genDist,cosDist)

[1] 0.5889319

R> cor(genDist,resDist)
```

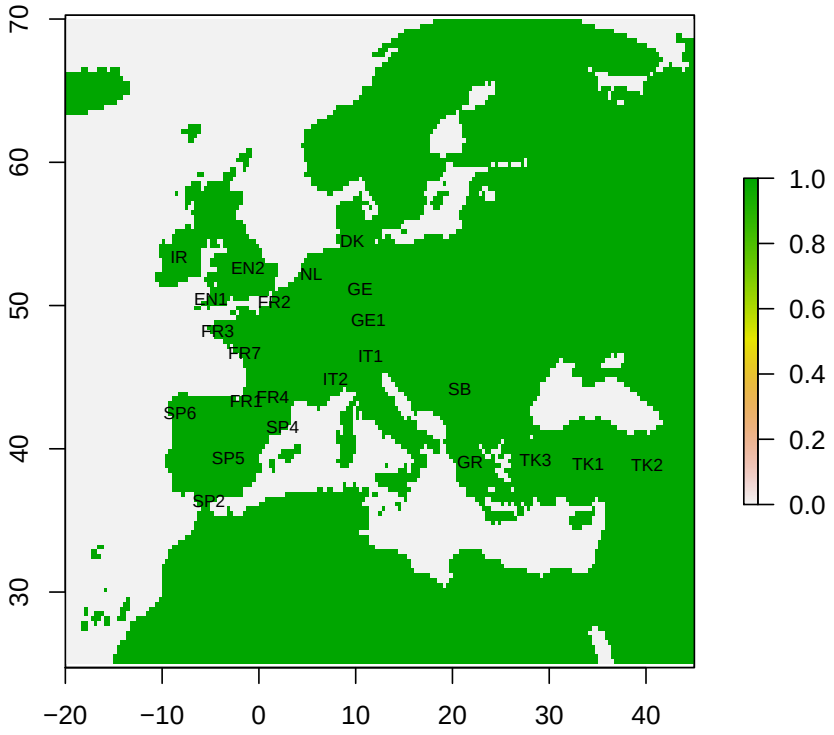

Figure 10: Map of genotyped populations.

```
[1] 0.1921118
```

Amont the measure evaluated until now, the great-circle distance between points turns out to be the best predictor of genetic distance. The other distance measures incorporate more information about the geographic space in which geneflow takes place, but do not improve the prediction. But how well does an expansion from Anatolia explain the spatial pattern?

```
R> origin <- unlist(popCoord[22,c("x","y")])
R> pI <- pathInc(trC, origin=origin, from=pC,
+    functions=list(overlap))
R> cor(genDist,pI[[1]])
```

```
[1] -0.7178576
```

At least at first sight, the overlap of dispersal routes explain the spatial pattern better than any of the previous measures. The negative sign of the last correlation coefficient was expected,

as more overlap in routes is associated with lower genetic distance. Additional work would be needed to improve predictions and compare the different models more rigorously.

11. Future Work

Improvements of **gdistance** and methodological refinements are expected in various areas.

- All measures based on random walks depend critically on solving sparse linear systems. This is the most time-consuming part of the calculations. Faster libraries could improve the **gdistance** package and may become available in R in the future.
- Research on distances in graph theory is a very dynamic field in the computational sciences. New measures and algorithms could be added to **gdistance** when they become available.
- More research on the consequences of connecting grids in different ways is necessary, as indicated in Section 2. This should bring more precision to random walk calculations in geospatial analysis. Comparing the results of grid-based calculations to continuous space simulations or analytical solutions would be the way forward (Birch 2006).

12. Final Remarks

Questions about the use of **gdistance** can be posted on the r-sig-geo email list. Bug reports and requests for additional functionality can be mailed to jacobvanetten@yahoo.com.

References

Balaresque P, Bowden G, Adams S, Leung HY, King T, Rosser Z, Goodwin J, Moison JP, Richard C, Millward A, Demain A, Barbujani G, Previdere C, Wilson I, Tyler-Smith C, Jobling M (2010). "A Predominantly Neolithic Origin for European Paternal Lineages." *PLoS Biology*, **8**(1), e1000285.

Bates D, Maechler M (2012). ***Matrix**: Sparse and Dense Matrix Classes and Methods*. R package version 1.0-5, URL http://CRAN.R-project.org/package=Matrix.

Birch C (2006). "Diagonal and orthogonal neighbours in grid-based simulations: Buffon's stick after 200 years." *Ecological Modelling*, **192**(3-4), 637–644.

Chandra A, Raghavan P, Ruzzo W, Smolensky R, Tiwari P (1996). "The Electrical Resistance of a Graph Captures its Commute and Cover Times." *Computational Complexity*, **6**(4), 312–340.

Csardi G, Nepusz T (2006). "The **igraph** Software Package for Complex Network Research." *InterJournal*, **Complex Systems**, 1695. URL http://igraph.sf.net.

de Smith M, Goodchild M, Longley P (2009). *Geospatial Analysis*. Matador.

Doyle P, Snell J (1984). *Random Walks and Electric Networks*. Carus Mathematical Monographs 22. Mathematical Association of America.

Fouss F, Pirotte A, Renders JM, Saerens M (2007). "Random-Walk Computation of Similarities between Nodes of a Graph with Application to Collaborative Recommendation." *IEEE Transactions on Knowledge and Data Engineering*, **19**(3), 355 –369. ISSN 1041-4347. doi:10.1109/TKDE.2007.46.

GRASS Development Team (2012). *Geographic Resources Analysis Support System (GRASS GIS) Software*. Open Source Geospatial Foundation, USA. URL http://grass.osgeo.org.

Hijmans RJ, van Etten J (2012). *raster: raster: Geographic data analysis and modeling*. R package version 2.0-31, URL http://CRAN.R-project.org/package=raster.

Kivimäki I, Shimbo M, Saerens M (2012). "Developments in the Theory of Randomized Shortest Paths with a Comparison of Graph Node Distances." *ArXiv e-prints*. 1212.1666.

McCoy J, Johnston K (2002). *Using ArcGIS Spatial Analyst*. ESRI.

McRae B (2006). "Isolation by Resistance." *Evolution*, **60**(8), 1551–1561.

McRae B, Dickson B, Keitt T, Shah V (2008). "Using Circuit Theory to Model Connectivity in Ecology, Evolution, and Conservation." *Ecological Modelling*, **89**(10), 2712–2724. doi:10.1890/07-1861.1.

R Development Core Team (2012). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org/.

Ramachandran Sand Deshpande O, Roseman C, Rosenberg N, Feldman M, Cavalli-Sforza L (2005). "Support from the Relationship of Genetic and Geographic Distance in Human Populations for a Serial Founder Effect Originating in Africa." *Proceedings of the National Academy of Science*, **102**(44), 15942–15947.

Saerens M, Yen L, Fouss F, Achbany Y (2009). "Randomized Shortest-path Problems: Two Related Models." *Neural Computation*, **21**(8), 2363–2404.

Tobler W (1970). "A Computer Movie Simulating Urban Growth in the Detroit Region." *Economic Geography*, **46**(2), 234–240.

Tobler W (1993). "Three Presentations on Geographical Analysis and Modeling." URL http://www.ncgia.ucsb.edu/Publications/Tech_Reports/93/93-1.PDF.

van Etten J, Hijmans R (2010). "A Geospatial Modelling Approach Integrating Archaeobotany and Genetics to Trace the Origin and Dispersal of Domesticated Plants." *PLoS ONE*, **5**(8), e12060.

Wright S (1943). "Isolation by Distance." *Genetics*, **28**(2), 114–138.

Affiliation:

Jacob van Etten
Climate Change Adaptation Theme Group
Agrobiodiversity and Ecosystem Services Programme
Bioversity International
c/o CATIE, Turrialba, Costa Rica
E-mail: jacobvanetten@yahoo.com
URL: http://bioversityinternational.org