The GRAINSCAPE package for R: Tutorials using simulated landscapes (Prepared: January 15th, 2013)

P. Galpern[1,2]

[1] *Natural Resources Institute, University of Manitoba, Winnipeg, Canada*
[2] *email: pgalpern@gmail.com (for support, bug reports, etc.)*

Contents

1 Introduction **2**

2 Tutorials **2**
The minimum planar graph . . . 3
Step 1: Preparing the resistance surface . . . 3
Step 2: Extracting the MPG . . . 4
Step 3: Quick visualization of the MPG . . . 4
Step 4: Reporting on the MPG . . . 4
Step 5: Thresholding the MPG . . . 7
Step 6: Visualizing a thresholded graph . . . 9
Step 7: Next Steps . . . 9
Patch-based grains of connectivity . . . 9
Step 1: Begin with an MPG . . . 9
Step 2: Exploring the Voronoi tessellation . . . 9
Step 3: Building GOC models . . . 11
Step 4: Visualizing a GOC model . . . 11
Step 5: Identifying locations on a GOC model . . . 11
Step 6: Finding the distance between points on a GOC model . . . 11
Lattice grains of connectivity . . . 14
Step 1: Building a lattice GOC model . . . 14
Step 2: Producing and visualizing a lattice GOC model . . . 15
Step 3: Next steps . . . 15
Advanced visualization for grains of connectivity models . . . 15
Section 1: Visualizing GOC models with vectorized Voronoi polygons . . . 15
Section 2: Visualizing the complete GOC model, corridors and shortest paths . 18

1 Introduction

This document is a supplement to the GRAINSCAPE package, and is included as an R vignette. GRAINSCAPE can be downloaded at http://grainscape.r-forge.r-project.org. Here, we illustrate the use of the package in a tutorial format.

Are animals free to move across the landscape? What, if anything, may limit or encourage their movement and dispersal. These are the questions of landscape connectivity research, and natural resource managers and practitioners of conservation are increasingly asking them in the course of their work. What are the effects of habitat fragmentation, and can organisms find new habitat when they need to? Can corridors help? Is it possible to stop the spread of pests and pathogens by managing the connectivity of their hosts?

The GRAINSCAPE package was developed for landscape connectivity analyses. The approach comes from the patch-based landscape graphs tradition (Urban and Keitt, 2001; Fall et al., 2007; Galpern et al., 2011) where a mathematical graph or network is used to represent the relationships among habitat patches. GRAINSCAPE provides two types of landscape connectivity models. The first, a minimum planar graph (Fall et al., 2007) is an efficient approximation of the potential for connectivity among a set of habitat patches.

The second model type is the grain of connectivity, which is based on the minimum planar graph, but extends in a way that may be useful for highly mobile terrestrial animals, such as ungulates and carnivores. For these organisms the patch may be not be a discrete and definable feature or landcover type, but is rather defined probabilistically (e.g. as the probability of resource selection). By modelling the relationships among a polygons in a Voronoi tessellation of the landscape rather than discrete patches, grains of connectivity offers improvements in the ability to model highly-mobile organisms (Galpern et al., 2012). Importantly it provides continuous coverage of the entire landscape surface, in a way that a typical patch-based graph does not, and the ability to measure landscape connectivity at multiple scales. The scalability, in particular, offers flexibility to accommodate uncertainty in how species may perceive landscape features.

GRAINSCAPE provides functions to extract the minimum planar graph and create two types of grains of connectivity: patch-based and lattic forms. The minimum planar graph, an essential step in all cases is extracted using a Windows-based binary SELES (Fall and Fall, 2001) that is distributed with the software. Consequently, the package works only on a Windows platform (or running R on a Windows virtual machine).

2 Tutorials

There are four tutorials. The first demonstrates how to extract a minimum planar graph and illustrates how it may be used to ask questions about landscape connectivity. The second demonstrates patch-based grains of connectivity modelling where animals are known to have some affinity to a resource patch, and the third uses lattice grains of connectivity modelling, an approach that is more appropriate where a patch concept does not clearly apply. A final tutorial demonstrates how to produce more advanced visualizations of grains of connectivity and the corridors connecting locations.

In each case we draw on artificial landscape data that is provided with the package. All analyses presented here can be run with the basic GRAINSCAPE installation.

Tutorial 1 The minimum planar graph

The minimum planar graph (hereafter MPG) is a spatial representation of a graph or a network that provides an efficient approximation of all possible pairwise connections between graph nodes (Fall et al., 2007). In graph-based landscape connectivity analyses, graph nodes have typically been patches of habitat that are demonstrably important for the species in question (Fall et al., 2007).

An MPG has links (i.e. models the possibility for organism movement and dispersal) between spatially adjacent habitat patches. In some cases spatially adjacent patches may not be linked, if the shortest connection between them can be made through a third patch. In practice, this property means that the MPG can be used to make a simple and easily visualized picture of how a set of habitat patches is connected. The alternative, the complete graph, can quickly become challenging to interpret because these may contain a dense set of graph links making the pattern difficult to discern. A second advantage of the MPG is the much reduced set of graph links; this can be valuable where computational efficiency is important, and essential where the number of habitat patches being modelled numbers in the thousands.

However, there are some types of connectivity analyses where the MPG approximation of the complete graph is not appropriate. For example, assessing community structure within a landscape patch network (i.e. finding sets of patches that are densely connected) is not possible as redundant connections have been removed intentionally. Equally, the MPG is a poor choice for prioritizing the influence of a patch for connectivity, the objective in a number of landscape graph studies e.g. (Pascual-Hortal and Saura, 2006). Please see (Galpern et al., 2011) for further discussion of these limitations and of the MPG.

Step 1 Preparing the resistance surface

The MPG has typically been constructed using least-cost path links between the perimeters of habitat patches. This implies that landscape structure in the "matrix" between patches is influencing movement, and that the organism in question is on average minimizing its costs when moving through this matrix (an assumption possibly appropriate for terrestrial animals, and terrestrial animal-dispersed plants). Equally, MPGs can be constructed using Euclidean links, where the only influence of the matrix on movement is the effect of spatial separation (i.e. distance it presents between neighbouring habitat). In the following example we illustrate just the case where links are least-cost paths on a resistance surface. Euclidean links can be produced by passing a uniform cost surface (a constant raster), and a raster describing the patches. The only difference in analysis is at the first step below.

We begin by loading a landscape raster distributed with the package. Note that any raster format readable using the `raster` package can be used here. The `.asc` format rasters distributed with the package are `ESRI ArcASCII` format.

```
> patchy <- raster(system.file("extdata/patchy.asc", package="grainscape"))
```

Then, for convenience, we use R to turn this raster into a resistance surface. In this example we will assume the feature class 1 are the patches, so we will set them to resistance value also equal to 1 (i.e. no additional resistance to movement than distance alone). The river, feature class 2, is assigned the highest resistance of 10. Other features are assigned values in between. The parameterization of resistance surfaces is itself a big topic (Zeller et al., 2012).

```
> patchyCost <- reclassify(patchy, rcl=cbind(c(1,2,3,4,5), c(1, 10, 8, 3, 6)))
> plot(patchyCost)
```

The result is shown in Figure 1.

Step 2 Extracting the MPG

With a resistance surface in hand the next step is to create the MPG. Basic use of the function, `gsMPG()` to do this is shown here. Here for simplicity we assume that all areas with the resistance value equal to 1 on the raster are patches.

```
> #patchyMPG <- gsMPG(patchyCost, patch=patchyCost==1)
```

Note that we defined the patches by passing a binary raster `patchyCost==1` as input. If patches are to be defined in a different way (e.g. using resistance classes 1 and 2) then a binary patch raster can be produced as `patch=patchyCost %in% c(1,2)`. Equally, we could use the `raster` package to load a binary raster produced manually in GIS software and specify the resulting raster object as the `patch` parameter.

Upon execution of this function, GRAINSCAPE calls SELES, a binary executable distributed with the package, in order to extract the graph, and then automatically imports the output of the SELES run back into R transparently. More control can be had over this process if required (see manual).

Step 3 Quick visualization of the MPG

A quick way to visualize the MPG is provided in the `gsMPG` object produced by `gsMPG()`.

```
> plot(patchyMPG$mpgPlot)
```

However, I've done something a little more advanced so that it shows up cleanly in the PDF you are now reading. The result is shown in Figure 2.

Step 4 Reporting on the MPG

Following extraction, the MPG is available as an `igraph` object (see `patchyMPG$mpg`) and can be analyzed using any of the functions in this package. A quick way to report on the structure of the graph in tabular format is provided by the GRAINSCAPE function `gsGraphDataFrame()`:

```
gsGraphDataFrame(patchyMPG)
```

The output shows the structure of the vertices (nodes) and their attributes under the list element `$v` as well as the structure of the graph in the form of an edge list (i.e. pairs of edges `e1` and `e2` that are connected) and associated edge (link) attributes under the list element `$e`. Note that only the first three lines of each

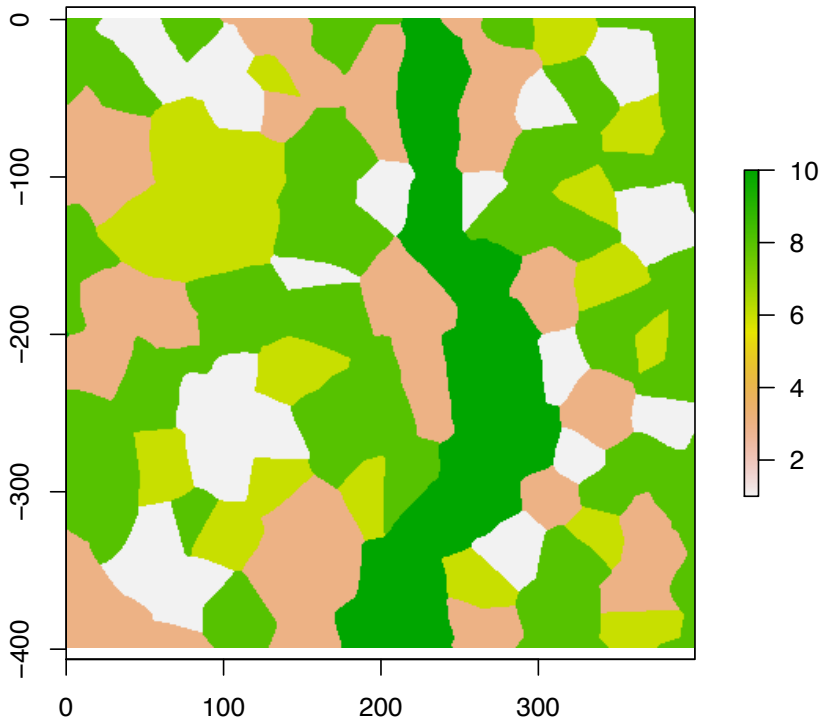

Figure 1: Input raster resistance surface to create MPG. Features with value of 1 will be the patches in the graph. A river (dark green) has the highest resistance in this example.

```
> ## Plot the patches / nodes
> plot(patchyMPG$patchId, col="black", main="", legend=FALSE)
> ## Before plotting the links we'll make them wider so they print nicely
> links <- patchyMPG$lcpPerimWeight > 0
> links[raster::edge(links, type="outer")] <- 1
> plot(links, col="darkgrey", add=TRUE, legend=FALSE)
```

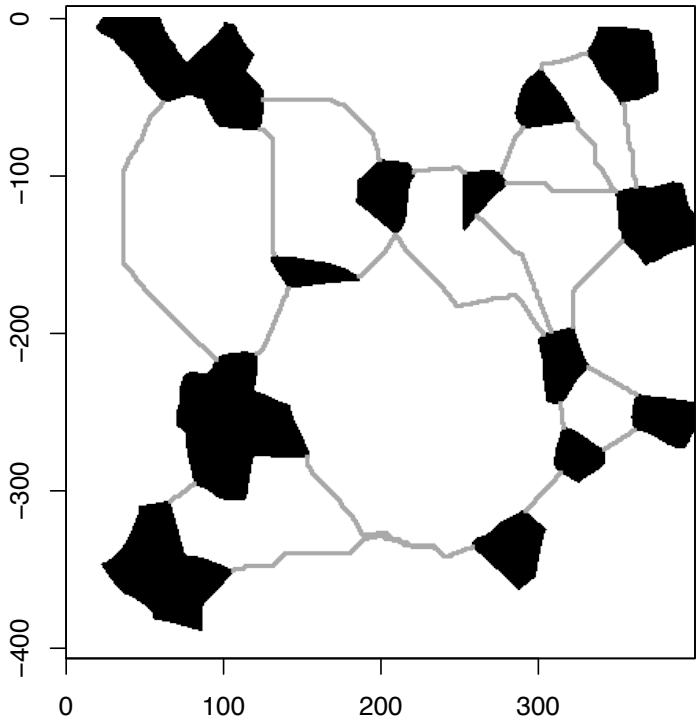

Figure 2: A quick visualization of the MPG. Black areas are patches (nodes or vertices) in the graph, and grey lines are links (edges) showing the least-cost path between the perimeters of the patches. In depth discussion of how the MPG is generated can be found elsewhere (Fall et al., 2007)

has been reproduced here. Please see the manual for the interpretation of the attributes.

Note! however that the attribute `eucPerimWeight` is not the same as extracting the MPG on a uniform (Euclidean) cost surface. This attribute describes the Euclidean distance between patch perimeters given the Voronoi tessellation of the cost surface (the dual of the MPG). Please see the next tutorial for information about the Voronoi tessellation.

```
[[1]]
[[1]]$v
  name patchId patchArea patchEdgeArea coreArea centroidX centroidY
1    1       1      4005           389     3616  81.04227 -32.63843
2    2       2      1686           189     1497 355.53782 -27.94127
3    3       3       939           154      785 302.14102 -54.40094

[[1]]$e
  e1 e2 linkId lcpPerimWeight eucPerimWeight startPerimX startPerimY
1 12 13     24       843.1065       151.5432   106.36687  -350.56063
2  9 13     23       788.6869       118.1933   153.30812  -278.65062
3  1  9     22       762.3942       165.2791    61.42312   -53.93187
  endPerimX endPerimY linkType
1 258.17687 -335.5794        3
2 258.17687 -335.5794        2
3  95.38062 -216.7281        2
```

Step 5 Thresholding the MPG

A frequent step in the analysis of a patch-based landscape graph is to threshold the graph into a series of subgraphs or components representing connected areas for an organism (Galpern et al., 2011, 2012; Urban and Keitt, 2001). This has sometimes been called a scalar analysis (Brooks, 2003).

The function `gsThreshold()` provides a way to conduct a scalar analysis at multiple scales. Here we ask for 5 thresholds, and the function finds five approximately evenly-spaced threshold values in link length:

```
scalarAnalysis <- gsThreshold(patchyMPG, nThresh=5)
```

```
   maxLink nComponents
1   0.0000          13
2 210.7766           6
3 421.5532           1
4 632.3299           1
5 843.1065           1
```

The `$summary` of this analysis can be plotted to explore scales of aggregation in the landscape. Figure 3 shows a scalar analysis of this landscape with 100 thresholds, where the response variables is the number of components or subgraphs created by the thresholding.

Other independent variables about the components (e.g. area of patches) could, of course, be calculated by processing the thresholded graphs `scalarAnalysis$th` and their attributes using the `igraph` function `clusters()`.

```
> scalarAnalysis <- gsThreshold(patchyMPG, nThresh=100)
> plot(scalarAnalysis$summary[, "maxLink"],
+      scalarAnalysis$summary[, "nComponents"],
+      xlab="Link Threshold (resistance units)",
+      ylab="Number of components", type="l")
```

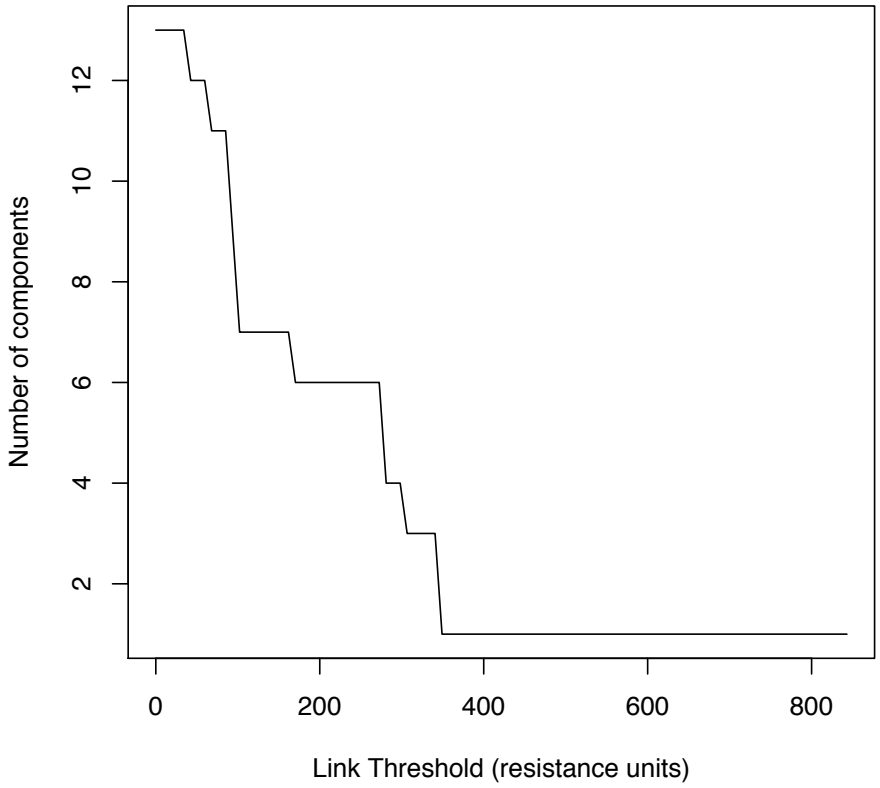

Figure 3: A scalar analysis at 100 thresholds of the MPG in Figure 2. When the landscape is a single component at higher link thresholds all patches are completely connected. As an example, an organism able to disperse 200 resistance units would experience this landscape as 6 connected regions.

Step 6 Visualizing a thresholded graph

Consider an organism able to disperse a maximum of 200 resistance units. According to Figure 3 this organism would experience this landscape as 6 connected regions. This is visualized in Figure 4.

Step 7 Next steps

With the MPG in hand, all sorts of subsequent analyses about landscape connectivity are possible. Grains of connectivity (GOC) the subject of the next three tutorials is an example of using the MPG and its complement the Voronoi tessellation.

When programming your own analyses in R based on the MPG it is helpful to observe that the `patchyMPG$patchId` and `patchyMPG$lcpLinkId` rasters contain the numerical IDs of the patches (nodes) and links respectively. These are are also contained as attributes in the `igraph` object `patchyMPG$mpg`. Using these three data objects together gives flexibility to visualize any graph analysis. To explore the attributes further use `gsGraphDataFrame()`.

Tutorial 2 Patch-based grains of connectivity

Grains of connectivity (hereafter GOC) was initially developed in two papers (Galpern and Manseau, 2013; Galpern et al., 2012). Please refer to these papers for much more detail on this method. In summary, grains of connectivity describes a tessellation of functionally-connected regions of the landscape. These may be especially useful for modelling landscape connectivity for highly-mobile terrestrial organisms that are not obligate patch occupants.

Patch-based GOC, shown in this tutorial, may be most relevant for oganisms that have patch affinities, but experience "habitat" as a probabilistic surface. Lattice GOC, shown in the next tutorial, is likely more appropriate when a patch affinity cannot be supported (i.e. there is no resource for which proximity is demonstrably important). This may also be the case when modelling landscape connectivity for large terrestrial mammals.

Step 1 Begin with an MPG

Here, we repeat the steps in Tutorial 1 for the same resistance surface. Any of the variations imaginable for the MPG are equally valid here.

```
> patchy <- raster(system.file("extdata/patchy.asc", package="grainscape"))
> patchyCost <- reclassify(patchy, rcl=cbind(c(1,2,3,4,5), c(1, 10, 8, 3, 6)))
> #patchyMPG <- gsMPG(patchyCost, patch=patchyCost==1)
```

Step 2 Exploring the Voronoi tessellation

Before we build the GOC graph, we should explore the essential building block of GOC, which is the Voronoi tessellation. This particular Voronoi tessellation was first described elsewhere (Fall et al., 2007) and is the complement of the MPG. It is found by finding the region of proximity in resistance units around a resource patch. In contrast to a typical Voronoi tessellation where the generators are points and distance is Euclidean, this tessellation uses two-dimensional patches as generators

```
> ## Plot the patches
> plot(patchyMPG$patchId, main="", col="black", legend=FALSE)
> ## Plot the links, but emphasize them first for printing
> thresholdLink <- patchyMPG$lcpPerimWeight <= 200
> thresholdLink[thresholdLink==0] <- NA
> thresholdLink[raster::edge(thresholdLink, type="outer")] <- 1
> plot(thresholdLink, add=TRUE, col="darkgrey", legend=FALSE)
```

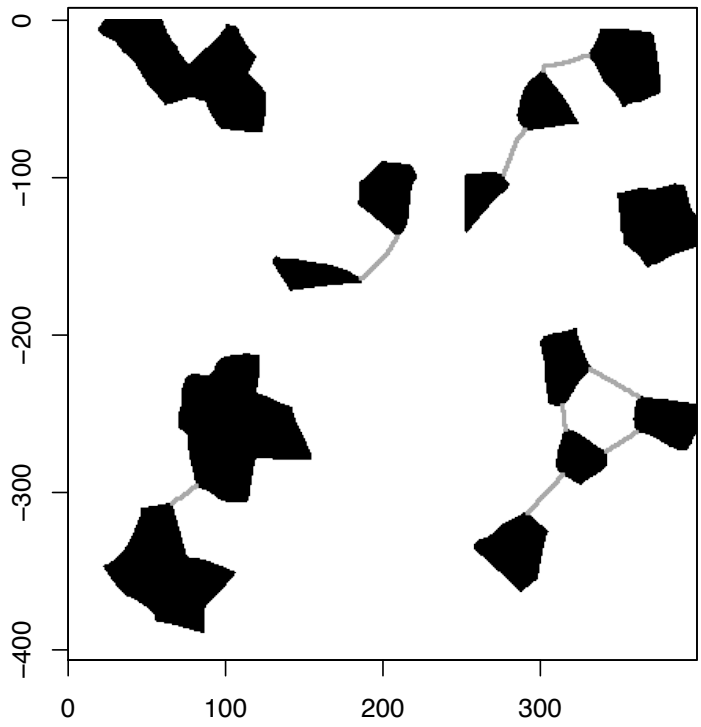

Figure 4: The thresholded MPG depicted with a link length of 200 resistance units. An organism that can disperse a maximum of 200 resistance units would experience this landscape as 6 connected regions in the depicted spatial configuration. Note that the plotting has been made a little more complicated than it needs to be in order to emphasize the links for printing.

and distance is calculated in cost or resistance space. Ultimately, the tessellation is found in SELES using a marching algorithm.

The Voronoi tessellation was extracted by `gsMPG()` in order to build the MPG. It can be plotted simply as follows:

```
plot(patchyMPG$voronoi)
```

A plot with the Voronoi tessellation and the patches superimposed is shown in Figure 5.

Step 3 Building GOC models

GOC analysis is essentially a scalar or thresholding analysis (see `gsThreshold()` and Tutorial 1) except the graph being thresholded is one of Voronoi polygons rather than patches. In particular, it is the MPG of Voronoi polygons that is thresholded. As links are added, and Voronoi polygons linked, the relevant polygons are combined describing larger connected regions. An additional step is that the links connecting a pair of polygons are the mean value of all links connecting any patches in each of the two polygons from the MPG (Galpern et al., 2012).

The function `gsGOC()` builds GOC models at multiple thresholds. As with `gsThreshold()` we can specify the number of thresholds, or grains of connectivity models, we want to create using the `nThresh` parameter.

```
patchyGOC <- gsGOC(patchyMPG, nThresh=100)
```

Step 4 Visualizing a GOC model

To get a quick sense of the connected regions described by a GOC model at a given threshold (or scale of movement) we can use the function `gsGOCVisualize()`. This example uses the functions plotting mechanism to plot the 62nd threshold in the `patchyGOC` object.

```
gsGOCVisualize(patchyGOC, whichThresh=4, doPlot=TRUE)
```

Figure 6 shows the resulting plot.

Step 5 Identifying locations on a GOC model

Organisms have locations in space. It is often necessary to know which connected region contains a given location in a GOC model. This can be done using the function `gsGOCPoints()`. First we create a thousand random locations that are on the landscape, and then run the function

```
> loc <- cbind(runif(1000)*399, runif(1000)*-399)
> locGOC <- gsGOCPoint(patchyGOC, loc)
```

The resulting object `locGOC` reports on the polygon which contains each location at each threshold (`pointPolygon`) as well as several other variables describing features of the polygons in which point locations are contained.

Step 6 Finding the distance between points on a GOC model

```
> plot(patchyMPG$voronoi, main="")
> plot(patchyMPG$patchId, add=TRUE, col="black", legend=FALSE)
```

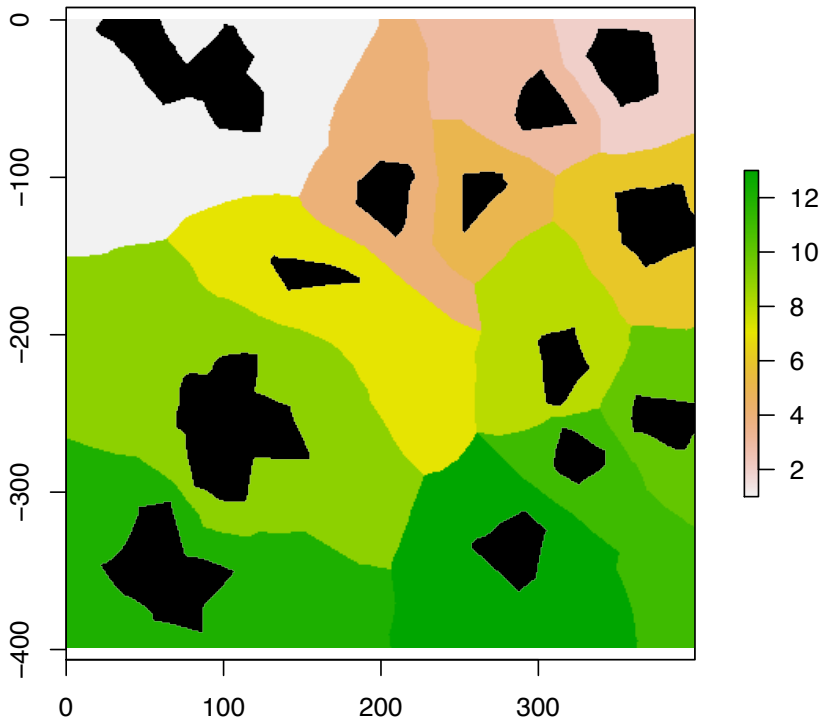

Figure 5: A Voronoi tessellation. This is the complement of the MPG. The patches are used as generators, and regions of proximity (polygons of different colours) are found in cost or resistance units. The method was first described by (Fall et al., 2007).

```
> gsGOCVisualize(patchyGOC, whichThresh=62, doPlot=TRUE)
```

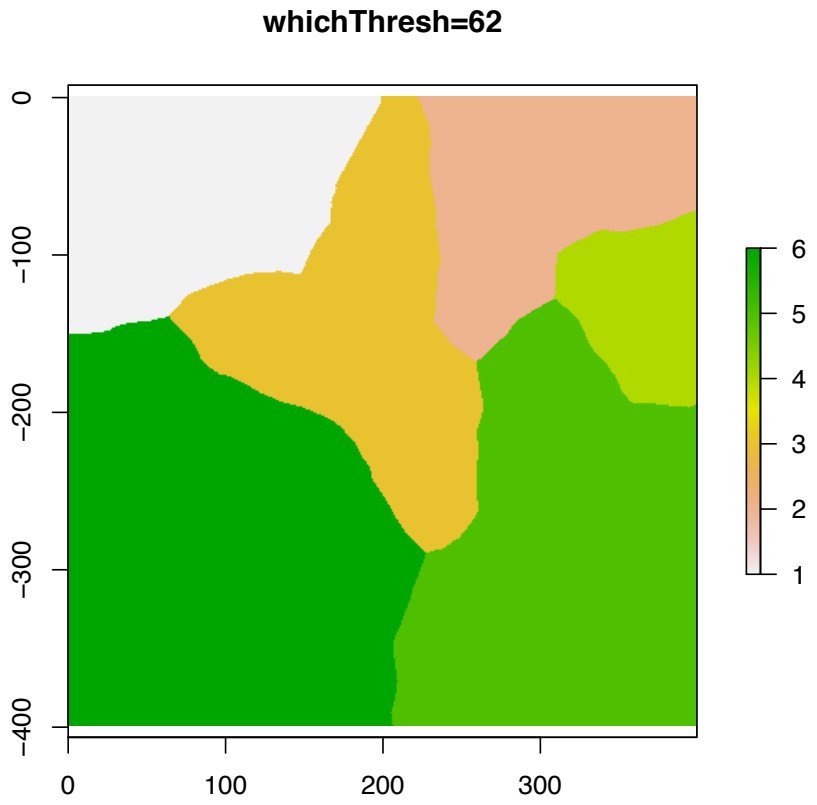

Figure 6: A visualization of a GOC model. In this case it is the 62nd scale or threshold extracted, which represents a link threshold of 168 resistance units, and a landscape of 6 components. Voronoi polygons imply regions that are functionally-connected at the given movement threshold.

As mentioned at an earlier step, at each threshold (or scale) there is also a GOC graph describing the relationships among the polygons. We can find the distance between the polygons containing pairs of points using the `gsGOCDistance()` function. This may be particularly useful in certain types of landscape genetic analyses where genetic distances are correlated with landscape distances (Galpern et al., 2012). A key advantage is that thresholding permits scaling of these distances for coarse representations of the landscape, removing variation from the calculation of distance that may be uncorrelated with dispersal.

Here we'll use four locations on the landscape.

```
> loc <- rbind(c(100,-100), c(200, -200), c(300, -300), c(100, -300))
> distGOC <- gsGOCDistance(patchyGOC, loc)
```

The resulting object `locGOC` reports on the pairwise distances in accumulated resistance units between each of those four points given the GOC model at each threshold. The results for the 62nd threshold are shown, where the row and column names give the `polygonId` containing each of the four points.

```
> distGOC$th[[62]]

$grainD
         1        3        5        6
1   0.0000 365.4186 976.5450 706.8379
3 365.4186   0.0000 611.1264 341.4193
5 976.5450 611.1264   0.0000 815.8967
6 706.8379 341.4193 815.8967   0.0000
```

Tutorial 3 Lattice grains of connectivity

A lattice grains of connectivity model uses a lattice of focal points superimposed over the landscape rather than patches on the landscape as nodes in the MPG. These points are also the generators used in the Voronoi tessellation. The result is a more generic way to scale up a resistance surface to coarser scales, without requiring assumptions about a particular habitat type that may be "focal" for connectivity. All that is required for analysis is an input resistance surface, reducing the ecological parameter set of the model.

Step 1 Building a lattice GOC model

For lattice GOC modelling a decision is required regarding the spacing of points in the lattice. This amounts to a decision about the finest scale, or grain, to be analyzed. Analysis time quickly increases as more nodes (lattice points) are included, so this decision should be made carefully.

The steps in extracting the MPG are the same as before, except instead of specifying a raster for the `patch` parameter, we specify an integer. This integer gives the spacing in cells between focal points in the square lattice. Thus a square raster of `400` cells in dimension, and `patch=40` produces a lattice of `10 * 10 = 100` focal points.

```
> #patchyMPGLattice <- gsMPG(patchyCost, patch=40)
```

Let's check how many focal points were actually used in this case:

```
> max(patchyMPGLattice$patchId[], na.rm=TRUE)

[1] 100
```

Which is what we expected. A visualization of these focal points and the MPG is shown in Figure 7.

Step 2 Producing and visualizing a lattice GOC model

These steps proceeds much as with the patch-based GOC model. The next step is to produce GOC model.

```
> #patchyGOCLattice <- gsGOC(patchyMPGLattice, nThresh=100)
```

Visualization of the Voronoi tessellation for the 35th threshold is shown in Figure 8.

Step 3 Next steps

Further analyses can proceed as with the patch-based GOC model. For example `gsGOCPoints` and `gsGOCDistance` can be used, as can other `igraph` analyses of the graphs contained in `patchyGOCLattice$th` and `patchMPGLattice$mpg`.

Tutorial 4 Advanced visualization for grains of connectivity models

So far we have been visualizing the products of analysis using raster formats. Since all analyses take place in raster format this has the added advantage of speed. However, for publications, or on-screen visuals, it tends to be better to have high resolution vector representations; particularly of the Voronoi polygons, as this makes it easier to understand what they show without confusing blocks of colour.

In the two sections of this tutorial we demonstrate how to use the RGEOS package in R to produce polygon representations of the Voronoi polygons in a GOC model, as well as using a GRAINSCAPE function intended for advanced visualizations.

Beware! One caution is that the RGEOS package continues to have memory leak problems from its parent C GEOS library which extend to GRAINSCAPE. Occasionally when running analyses, particularly with thousands of nodes (i.e. polygons), RGEOS can consume all available memory and lead to a crash of the R instance. Also, when running multiple visualizations in succession it is often necessary to terminate the R instance to free up memory before running a subsequent visualization.

Section 1 Visualizing GOC models with vectorized Voronoi polygons

For final analyses where visualization is a goal, build the GOC model with the `sp=TRUE` parameter set. Also, the RGEOS package must be installed. Here, we do this for both the patch-based and lattice GOC models of Tutorials 2 and 3, respectively. Note that this can be very slow, consuming orders of magnitude more computational time!

```
> #patchyGOC_SP <- gsGOC(patchyMPG, nThresh=100, sp=TRUE)
> #patchyGOCLattice_SP <- gsGOC(patchyMPGLattice, nThresh=100, sp=TRUE)
```

Note how we can now visualize the Voronoi tessellation also with the `sp=TRUE` parameter to make things sharper. Figure 9 shows this for the lattice GOC model

```
> ## Plot the links but make them wider so they print nicely
> links <- patchyMPGLattice$lcpPerimWeight > 0
> links[raster::edge(links, type="outer")] <- 1
> plot(links, main="", col="darkgrey", legend=FALSE)
> ## Add the focal points.  We'll get the locations from the MPG object
> focalPts <- cbind(V(patchyMPGLattice$mpg)$centroidX, V(patchyMPGLattice$mpg)$centroidY)
> points(focalPts, pch="+", cex=1.25)
```

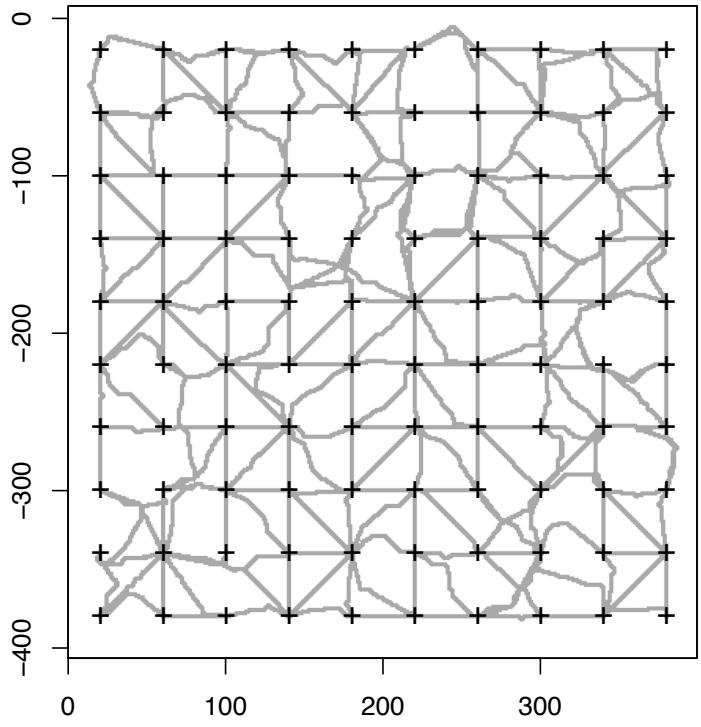

Figure 7: A visualization of an MPG of a lattice of focal points for the resistance surface in Figure 1. Again, we have complicated plotting a bit to emphasize the links better.

```
> plot(gsGOCVisualize(patchyGOCLattice, whichThresh=35)$voronoi,
+      col=sample(terrain.colors(100)), main="whichThresh=35")
```

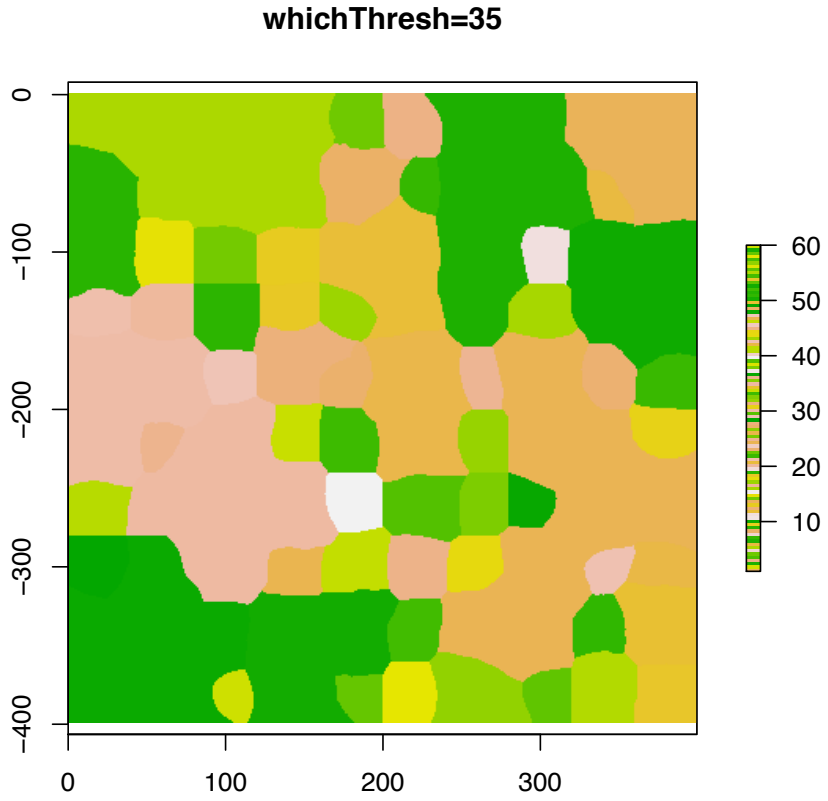

Figure 8: A visualization of a GOC model. In this case it is the 35th scale or threshold extracted, which represents a link threshold of 122 resistance units. Voronoi polygons imply regions that are functionally-connected at this movement threshold. It is easy to imagine, here, how using more lattice points makes it possible to define grains of connectivity with higher resolution.

at the finest threshold possible, superimposed over the input resistance surface used to generate the GOC model.

Section 2 Visualizing the complete GOC model, corridors and shortest paths

GRAINSCAPE provides a function called `gsGOCCorridor()` to simplify advanced visualizations when the goal is to show a GOC, the graph connecting Voronoi polygons and a shortest path or corridor between certain points on the landscape.

Figure 10 shows a visualization for the patch-based GOC model, and Figure 11 shows a visualization for the lattice GOC model. Finally Figure 12 demonstrates how to take manual control of the visualization to liven up the product!

References

Brooks, C.P. (2003). A scalar analysis of landscape connectivity. Oikos 102:433-439.

Fall, A. and J. Fall. (2001). A domain-specific language for models of landscape dynamics. Ecological Modelling 141:1-18.

Fall, A., M.J. Fortin, M. Manseau, and D. O'Brien. (2007). Spatial graphs: Principles and applications for habitat connectivity. Ecosystems 10:448-461.

Galpern, P., M. Manseau. (2013). Modelling the influence of landscape connectivity on animal distribution: a functional grain approach. Ecography. In press.

Galpern, P., M. Manseau, and P.J. Wilson. (2012). Grains of connectivity: analysis at multiple spatial scales in landscape genetics. Molecular Ecology 21:3996.4009.

Galpern, P., M. Manseau, and A. Fall. (2011). Patch-based graphs of landscape connectivity: A guide to construction, analysis and application for conservation. Biological Conservation 144:44-55.

Pascual-Hortal, L. and S. Saura. (2006). Comparison and development of new graph-based landscape connectivity indices: towards the priorization of habitat patches and corridors for conservation. Landscape Ecology 21:959-967.

Urban, D. and T. Keitt. (2001). Landscape connectivity: A graph-theoretic perspective. Ecology 82:1205-1218.

Zeller, K.A., K. McGarigal, and A.R. Whiteley. (2012). Estimating landscape resistance to movement: a review. Landscape Ecology:1-21.

```
> plot(patchyCost, legend=FALSE, main="")
> plot(gsGOCVisualize(patchyGOCLattice_SP, whichThresh=1, sp=TRUE)$voronoiSP,
+     add=TRUE)
> focalPts <- cbind(V(patchyMPGLattice$mpg)$centroidX, V(patchyMPGLattice$mpg)$centroidY)
> points(focalPts, pch="+", cex=1.25)
```

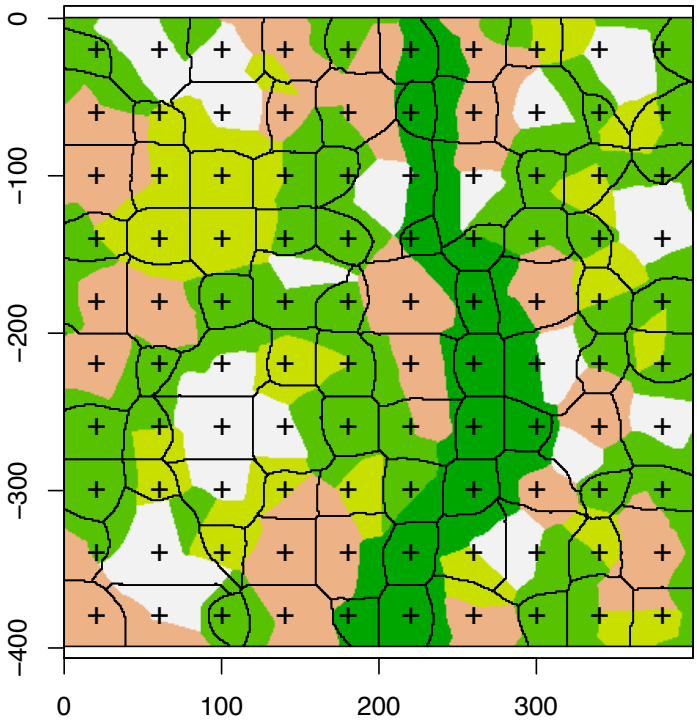

Figure 9: A visualization of a lattice GOC model using vectorized Voronoi polygons, superimposed over the input resistance surface. In this case the Voronoi polygons show the finest grain possible using the 100 point lattice (+) initially specified. Plotting only the outlines of the Voronoi polygons makes it easier to show other information on this map (e.g. resistance surface) and retain clarity. Recall, when interpreting this figure, that the polygons show regions of proximity from each focal lattice point given distance in resistance units.

```
> loc <- rbind(c(50, -350), c(350, -50))
> corridor1 <- gsGOCCorridor(patchyGOC_SP, whichThresh=1, coords=loc, doPlot=TRUE)
> points(loc, pch=c("P", "Q"), cex=1.25)
> mtext(paste("Shortest path between P and Q:",
+     round(corridor1$corridorLength, 2), "resistance units",
+     sep=" "),
+     side=1)
```

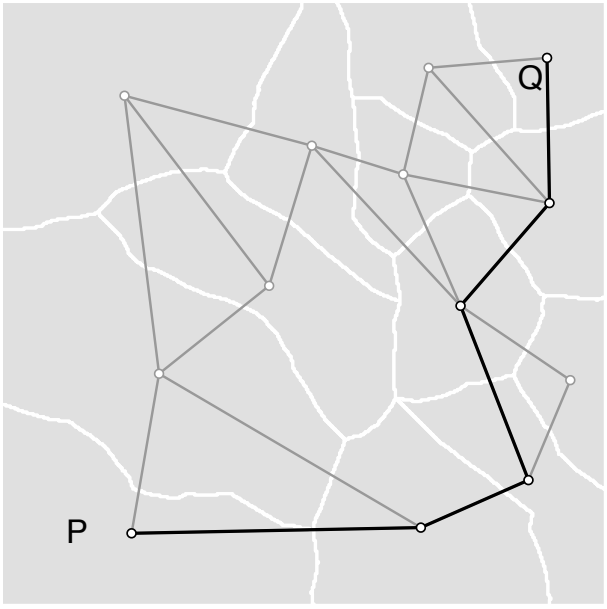

Figure 10: A visualization of the complete patch-based GOC model at its finest scale and the shortest path or least-cost corridor given the GOC graph between two locations on the map (P and Q) at this scale. Grey regions are Voronoi polygons in the patch-based GOC model.

```
> loc <- rbind(c(50, -350), c(350, -50))
> corridor1 <- gsGOCCorridor(patchyGOCLattice_SP, whichThresh=45,
+    coords=loc, doPlot=TRUE)
> points(loc, pch=c("P", "Q"), cex=1.25)
> mtext(paste("Shortest path between P and Q:",
+     round(corridor1$corridorLength, 2), "resistance units",
+     sep=" "),
+     side=1)
```

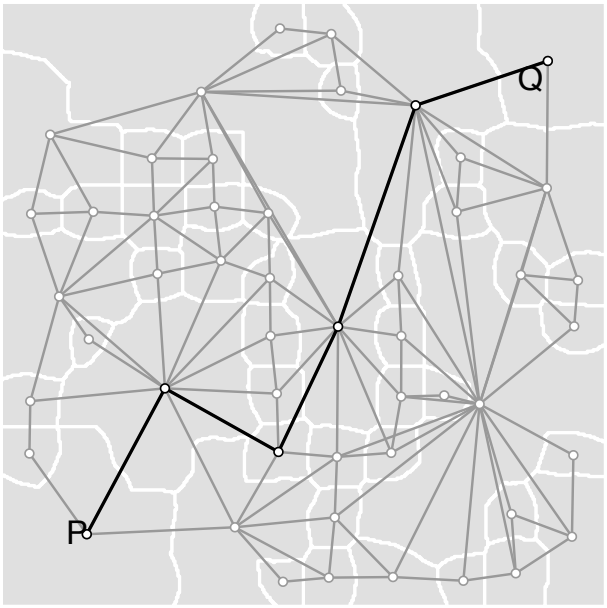

Figure 11: A visualization of the complete lattice GOC model at a relatively coarse scale and the shortest path or least-cost corridor given the GOC graph between two locations on the map (P and Q) at this scale. Grey regions are Voronoi polygons in the lattice GOC model.

```
> tiny <- raster(system.file("extdata/tiny.asc", package="grainscape"))
> ## Create a resistance surface from a raster using an is-becomes reclassification
> tinyCost <- reclassify(tiny, rcl=cbind(c(1, 2, 3, 4), c(1, 5, 10, 12)))
> ## Produce a patch-based MPG where patches are resistance features=1
> #tinyPatchMPG <- gsMPG(cost=tinyCost, patch=tinyCost==1)
> ## Extract a representative subset of 5 grains of connectivity using sp=TRUE
> #tinyPatchGOC <- gsGOC(tinyPatchMPG, nThresh=5, sp=TRUE)
> ## Visualization of a corridor
> corridorStartEnd <- rbind(c(10,10), c(90,90))
> tinyPatchCorridor <- gsGOCCorridor(tinyPatchGOC, whichThresh=3,
+     coords=corridorStartEnd)
> plot(tinyPatchCorridor$voronoiSP, col="lightgrey", border="white", lwd=2)
> plot(tinyPatchCorridor$linksSP, col="darkred", lty="dashed", add=TRUE)
> plot(tinyPatchCorridor$nodesSP, col="darkred", pch=21, bg="white", add=TRUE)
> plot(tinyPatchCorridor$shortestLinksSP, col="darkred", lty="solid",
+     lwd=2, add=TRUE)
> plot(tinyPatchCorridor$shortestNodesSP, col="darkred", pch=21,
+     bg="darkred", add=TRUE)
> mtext(paste("Corridor shortest path length:",
+              round(tinyPatchCorridor$corridorLength, 2),
+              "resistance units"), side=1)
```

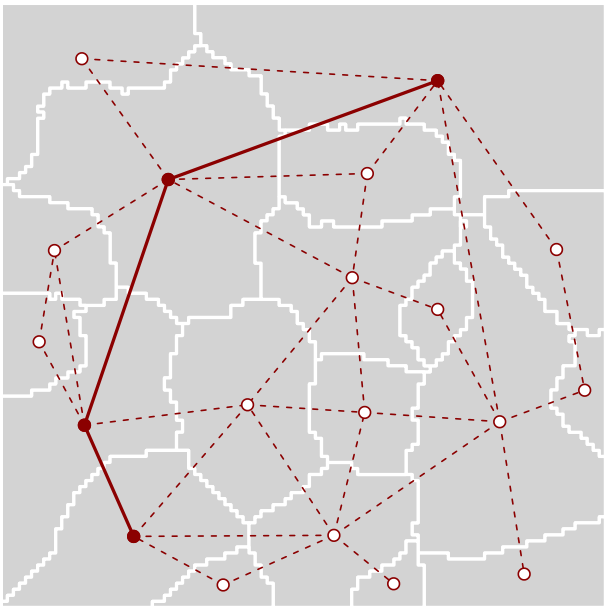

Figure 12: More control over a corridor visualization using an example from the manual, and the tiny.asc resistance surface