Generating Accumulated Cost Surfaces with Irregular Landscape Graphs

Joseph Stachelek

2015-09-16

This vignette describes the `irlgraph` package which provides functions to generate accumulated cost surfaces using irregular landscape graphs. Accumulated cost surfaces generated in this manner have several benefits as detailed by Etherington (2012). Most notably, there is a large decrease in processing time relative to regular graphs. There may be even larger decreases in processing time relative to straight implementations of Dijkstra's algorithm (Stachelek 2015).

Installation

Currently the `irlgraph` package can only be installed by compiling from source. One easy way to compile `R` packages is to use the `devtools` package (Wickham and Chang 2015). Eventually, `irlgraph` may be submitted to CRAN where compiled binaries would be made available (http://cran.r-project.org).

```
devtools::install_github("jsta/irlgraph", build_vignettes = TRUE)
```

Generating Irregular Landscape Graphs

The Etherington (2012) method of constructing irregular landscape graphs involves selecting a subset of points contained in the analysis domain. These subsets include:

- Points of Interest - optional points supplied to the `poicoords` argument of the `irl_graph` function.
- Very important point cells - cells that lie on either side of a boundary between different cost-categories in the underlying cost raster. Boundaries are defined based on the difference between the user-set `cutoff` parameter and the standard deviation of a 3x3 moving window calculation.
- Landscape limit cells - cells on the boundary of null data cells and cells bordering the cost raster extent.
- Ecological grain cells - a randomly selected number of cells. The proportion of cells selected in this manner is supplied to the `grainprop` argument of the `irl_graph` function.

- Null data cells

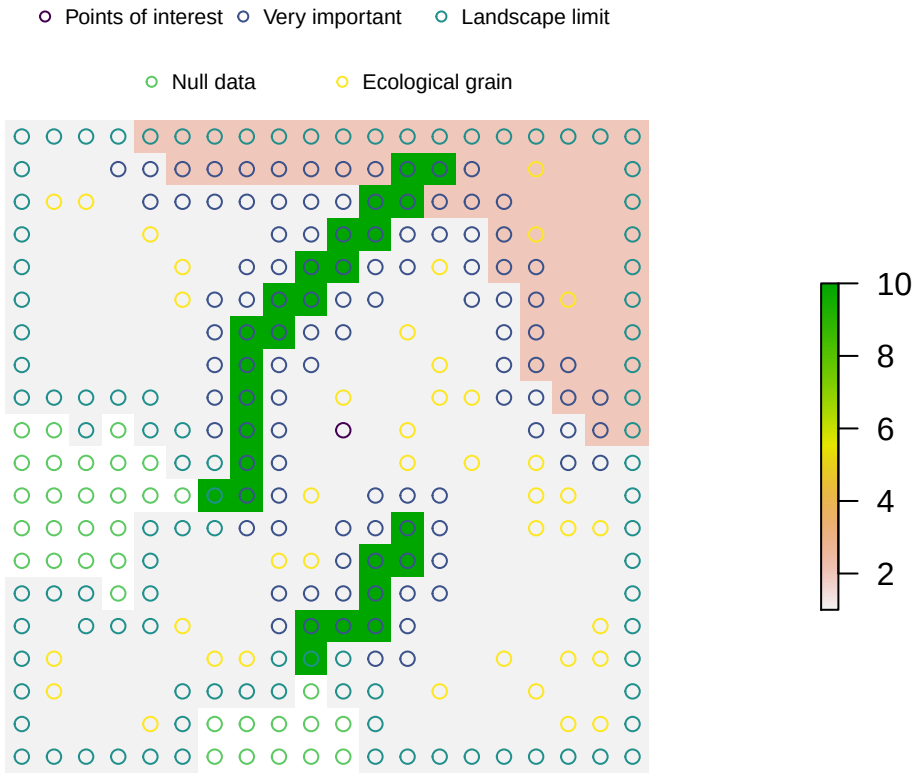
The following code block loads the cost raster from Etherington (2012) and calls the `irl_graph` function to generate an irregular landscape graph. Next, a sequence of plotting commands are called in order to visualize the point subsets in relation to the cost raster.

```
library(irlgraph)
library(raster, quietly = TRUE)

set.seed(123) #make reproducible

dm <- as.matrix(read.delim(system.file(
"extdata/etherington20120code/cost-surface20x20.txt", package = "irlgraph"),
skip = 6, na.strings = "-9999", header = FALSE, sep = " "))

poicoords <- matrix(c(10.5, 10.5), ncol = 2)
graph <- irl_graph(dm = dm, poicoords, grainprop = 0.25)
```

```
##
##      PLEASE NOTE:  As of version 0.3-5, no degenerate (zero area)
##      regions are returned with the "Qt" option since the R
##      code removes them from the triangulation.
##      See help("delaunayn").
```

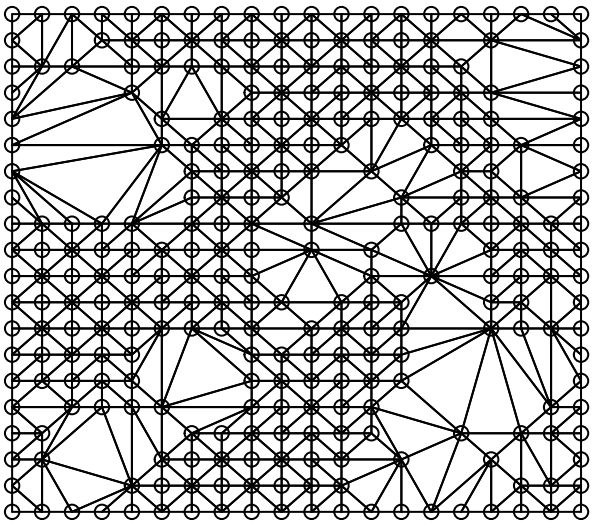
```
costsurf <- raster::raster(nrows = dim(dm)[1], ncols=dim(dm)[2],
resolution = 1, xmn = 0, xmx = dim(dm)[1], ymn = 0, ymx = dim(dm)[2]) #neccessary to set res
costsurf[] <- dm

par(mar = c(0,0,3,0))
plot(costsurf, box = FALSE, axes = FALSE)

legend("top", inset  = c(0, -0.2), legend = c("Points of interest", "Very important",
"Landscape limit"), col = viridis::viridis(5)[1:3], pch = 1 , horiz = TRUE,
xpd = TRUE, cex = 0.7, bty = "n")

legend("top", inset  = c(0, -0.1), legend = c("Null data", "Ecological grain"),
col = viridis::viridis(5)[4:5], pch = 1, horiz = TRUE, xpd = TRUE, cex = 0.7, bty = "n")

points(xyFromCell(costsurf, graph$poicells), col = viridis::viridis(5)[1])
points(xyFromCell(costsurf, graph$vipcells), col = viridis::viridis(5)[2])
```

```
points(xyFromCell(costsurf, graph$limitcells), col = viridis::viridis(5)[3])
points(xyFromCell(costsurf, graph$nullcells), col = viridis::viridis(5)[4])
points(xyFromCell(costsurf, graph$graincells), col = viridis::viridis(5)[5])
```



The underlying graph can be visualized with the following call to the `plot` command. This preliminary graph includes edge connections to null cell nodes. Note that these nodes and any edge connections to these cells are stripped in the code underlying the `irl_graph` function before the final graph is constructed.

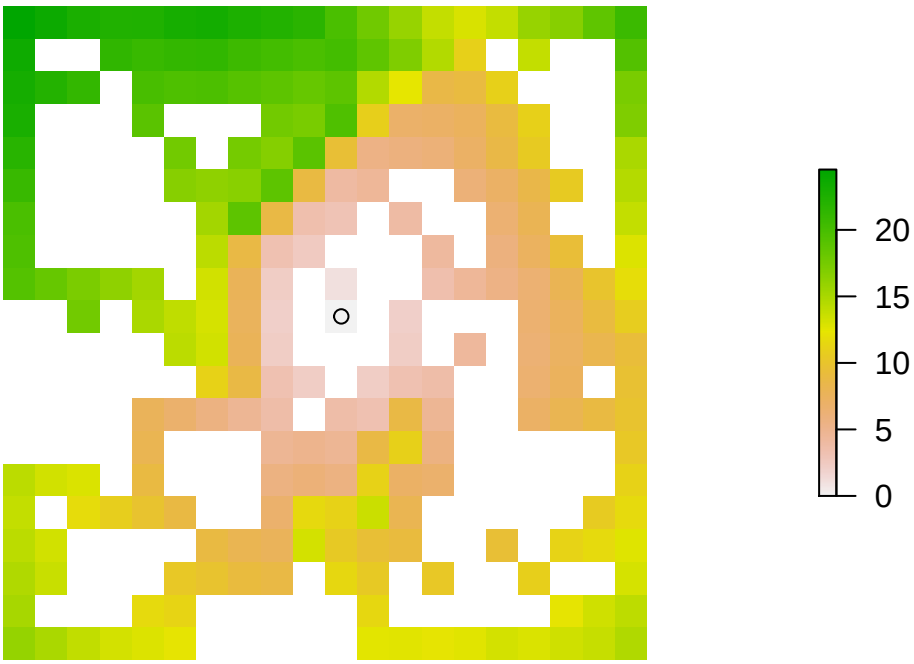
```
par(mar=c(2,0,0,0))
geometry::trimesh(graph$tri$tri, graph$allcoords)
points(graph$allcoords)
```



Generating Accumulated Cost Surfaces

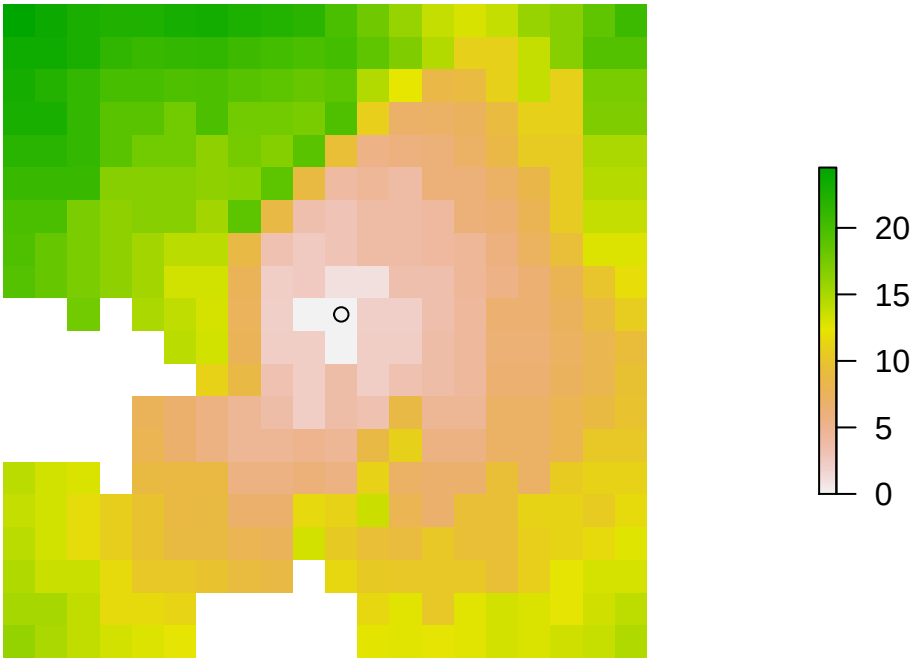
Partial accumlated cost surfaces can be generated from an irregular landscape graph by specifying a starting node to the `snode` or `scoord` arguments of the `acc_path` function.

```
result<-acc_path(graph = graph, scoord = c(10.5, 10.5), costsurf = costsurf)

par(mar=c(0,0,3,0))
plot(result, box = FALSE, axes = FALSE)
points(10.5, 10.5)
```



These partial surfaces can be converted to complete accumulated cost surfaces by using the `impute_na` function to perform nearest neighbor interpolation.

```
result <- impute_na(result, costsurf, graph)

par(mar=c(0,0,3,0))
plot(result, box = FALSE, axes = FALSE)
points(10.5, 10.5)
```



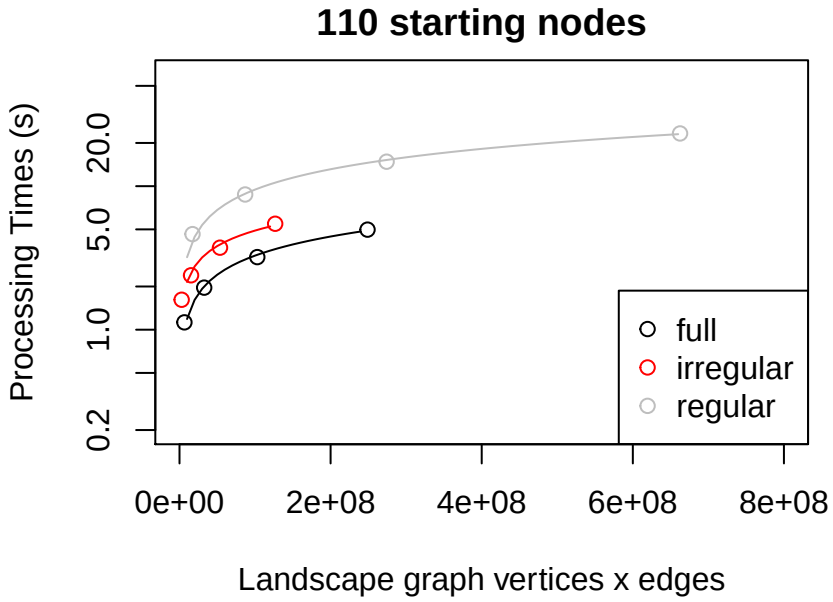
Processing Speeds

One advantage of using irregular landscape (IRL) graphs to generate accumulated cost surfaces is the reduction in processing times relative to regular landscape (RL) graphs. IRL graphs have less edge connections between nodes because they are formed from a delaunay triangulation rather than being fully diagonal. Apart from differences in edge density, it is difficult to compare the speed of the two approaches. Differences in programming language, implementation (compiled vs interpreted code), and style (looping vs vectorized code) can dramatically affect processing times and mask reliable estimation of processing speed. In the code chunk that follows, it appears that RL graphs are faster than IRL graphs for a single starting node. However, IRL graphs are faster when there are many (dozens) of starting nodes. IRL graphs afford greater increases in processing speed when greater numbers of starting nodes are calculated.

```
dm <- c("40", "60", "80", "100")
coordmat <- matrix(c(rbind(seq(0.5, 38.5, 0.5), 0.5), rbind(0.5, seq(0.5, 16.5, 0.5))), byro

read_dm <- function(x){
  as.matrix(read.delim(system.file(file.path(
"extdata/etherington20120code/", paste0("cost-surface", x, "x", x, ".txt")),
package = "irlgraph"), skip = 6, na.strings = "-9999.0", header = FALSE, sep = "\t"))
}
```

```
dm <- lapply(dm, function(x) read_dm(x))

make_costsurf <- function(x){
  costsurf <- raster::raster(nrows=dim(x)[1],ncols=dim(x)[2],resolution=1,xmn=0,
xmx = dim(x)[1], ymn = 0, ymx = dim(x)[2])
  costsurf[] <- x
  costsurf
}

costsurfs <- lapply(dm, function(x) make_costsurf(x))

times <- lapply(1:length(dm), function(x) system.time(irl_acc(dm[[x]],            poicoords
scoord = coordmat, cutoff = 0.2)))

fulltimes <- lapply(1:length(dm), function (x) system.time(irl_acc(dm[[x]],
poicoords = matrix(c(10.5, 10.5), ncol = 2), costsurf = costsurfs[[x]],
scoord = coordmat, irregular = FALSE, warn = FALSE)))

regtimes <- lapply(costsurfs,
function(x) system.time(gdist_acc(x, scoord = coordmat)))

times <- matrix(unlist(times), ncol = 5, byrow = TRUE)[,3]
fulltimes <- matrix(unlist(fulltimes), ncol = 5, byrow = TRUE)[,3]
regtimes <- matrix(unlist(regtimes), ncol = 5, byrow = TRUE)[,3]

count_vertedge <- function(graph){
  igraph::vcount(graph$graph) * igraph::ecount(graph$graph)
  #a triangulation is not fully diagonal...
  #vcount includes nodes with no connections (null cells)
}

vertxedges <- lapply(dm, function(x) count_vertedge(irl_graph(x,
  grainprop = 0.25, cutoff = 0.2)))
fullvertxedges <- lapply(dm, function(x) count_vertedge(irl_graph(x, irregular = FALSE)))
regvertxedges <- lapply(costsurfs, function(x) count_vertedge(gdist_acc(x,
  scoord = matrix(c(10.5, 10.5), ncol = 2, byrow = TRUE))[[1]]))

par(mar=c(4,4,2,2))
plot(cbind(unlist(fullvertxedges), fulltimes), ylim = c(0.2, 60), xlim = c(0, 8e+08),
  log = "y", ylab = "Processing Times (s)", xlab = "Landscape graph vertices x edges", main
points(cbind(vertxedges, times), col = "red")
points(cbind(regvertxedges, regtimes), col = "grey")

df <- data.frame(vertxedges = unlist(vertxedges), times,
fullvertxedges = unlist(fullvertxedges), fulltimes,
```

```
regvertxedges = unlist(regvertxedges), regtimes)

fit <- nls(times ~ b * vertxedges^power, data = df, start = list(b = 0.001, power = 0.3))
fullfit <- nls(fulltimes ~ b * fullvertxedges^power,
  data = df, start = list(b = 0.001, power = 0.3))
regfit <- nls(regtimes ~ b * regvertxedges^power,
  data = df, start = list(b = 0.001, power = 0.3), control = list(maxiter = 500))

s <- seq(0, max(df$vertxedges), 1e+07)
fs <- seq(0, max(df$fullvertxedges), 1e+07)
rs <- seq(0, max(df$regvertxedges), 1e+07)

lines(s, predict(fit, list(vertxedges = s)), col = "red")
lines(fs, predict(fullfit, list(fullvertxedges = fs)), col = "black")
lines(rs, predict(regfit, list(regvertxedges = rs)), col = "grey")

legend("bottomright", legend = c("full","irregular", "regular"),
col = c("black", "red", "grey"), pch = 1)
```



References

Etherington, Thomas R. 2012. "Least-Cost Modelling on Irregular Landscape Graphs." *Landscape Ecology* 27: 957–68.

Stachelek, Joseph. 2015. *ipdw: Spatial Interpolation by Inverse Path Distance Weighting.* http://CRAN.R-project.org/package=ipdw.

Wickham, Hadley, and Winston Chang. 2015. *Devtools: Tools to Make Developing R Packages Easier.* http://CRAN.R-project.org/package=devtools.