Connecting Landscapes - *CoLa*

Supporting conservation programs with habitat connectivity tools

User's manual v1.0

Patrick Jantz Ph.D. & Iván González Garzón Ph.D. (c)

20th October, 2024

Funded by:

Developed by:

This project was funded by NASA grant No. 80NSSC21K1942 - Strengthening Natural Resource Management with New Protected Area Connectivity Tools.

Cite this work as: González, I. & Jantz, P. 2024. Cola: open-source software for landscape, corridors, kernels, and scenario modeling and prioritization.

Patrick Jantz, Ph.D. - Assistant professor at Northern Arizona University

Iván González Garzón Ph.D. (c) - Student at Northern Arizona University

Chapters tables

Document details

Version	Date	Author
V0.0. Initial version with installation, backend usage	2024-03-30	Iván González and Patrick Jantz Ph.D.
V1.0. New parameters in the back-end functions. Front-end usage section	2024-10-20	Iván González and Patrick Jantz Ph.D.

Table 1. Document evolution

Introduction

Motivation

Increasing loss and fragmentation of habitats have resulted in an urgent need to identify areas for conservation that also maintain and enhance the ecological connectivity of protected areas. The Connecting Landscapes (CoLa) decision support system (DSS) integrates and enhances landscape genetics and habitat connectivity tools (Landguth and Cushman 2010, Landguth et al. 2012) to aid in identifying connectivity conservation priorities and areas of potential human-wildlife conflict.

The DSS is supported by NASA Biological Diversity & Ecological Conservation program Grant No. 80NSSC21K1942 - Strengthening Natural Resource Management with New Protected Area Connectivity Tools, and was co-developed with the Wildlife Conservation Research Unit (WildCRU) of the University of Oxford, the United States Forest Service (USFS) Rocky Mountain Research Station, USFS International Programs, and Montana State University.

Primary inputs to the tools include habitat suitability layers or spatially explicit estimates of animal movement potential across different land use types (resistance layers), as well as population source points. When properly parameterized, the toolkit can generate estimates of population genetic structure, population density, core movement areas, and long-distance dispersal corridors.

For recent examples of how the individual tools have been used for wildlife research and conservation assessments, see Kaszta et al. (2020a), Kaszta et al. (2020b), Zeller et al. (2021), Ash et al. (2023), and Makwana et al. (2023).

Kaszta, Ż., Cushman, S.A. and Macdonald, D.W., 2020a. Prioritizing habitat core areas and corridors for a large carnivore across its range. *Animal Conservation*, 23(5), pp.607-616.

Kaszta, Ż., Cushman, S.A., Htun, S., Naing, H., Burnham, D. and Macdonald, D.W., 2020b. Simulating the impact of Belt and Road initiative and other major developments in Myanmar on an ambassador felid, the clouded leopard, Neofelis nebulosa. *Landscape Ecology*, 35, pp.727-746.

Zeller, K.A., Schroeder, C.A., Wan, H.Y., Collins, G., Denryter, K., Jakes, A.F. and Cushman, S.A., 2021. Forecasting habitat and connectivity for pronghorn across the Great Basin ecoregion. *Diversity and Distributions*, 27(12), pp.2315-2329.

Ash, E., Cushman, S., Kaszta, Ż., Landguth, E., Redford, T. and Macdonald, D.W., 2023. Female-biased introductions produce higher predicted population size and genetic diversity in simulations of a small, isolated tiger (Panthera tigris) population. *Scientific Reports*, 13(1), p.11199.

Makwana, M., Vasudeva, V., Cushman, S.A. and Krishnamurthy, R., 2023. Modelling landscape permeability for dispersal and colonization of tigers (Panthera tigris) in the Greater Panna Landscape, Central India. *Landscape Ecology*, 38(3), pp.797-819.

Definitions

- Rasters: Files representing geospatial data. Rasters contain gridded information in a rectangular structure, storing numbers in each cell. Typically, rasters consist of a single file, but some extension files are generated with accessory or summary information.
- Discrete rasters: Rasters with discrete values, where numbers represents categories
- Continuous rasters: Rasters with continuous values
- Point layers: Files representing geospatial data at particular locations. Typically they consist of multiple files (e.g. shapefile format). The information contained is a set of X and Y coordinates, with some attributes associated with each pair of coordinates.
- Nodes or patches: Spatial elements in the landscape representing hubs or areas that provides habitat for species
- Cost units: Geographic distance (in meters) multiplied by the cost of moving through the landscape (i.e. resistance). If, for example, you want to model corridors for a target species, and your target species has a maximum dispersal distance of 50,000 meters (50 km), and the average resistance of the landscape is 10, your target species will on average only be able to move 1/10th of the geographic distance, so 5000 meters (5 km). If you want to ensure that the modeled movement distance of your target species is similar to the geographic distance, you can multiply its maximum dispersal distance (in meters) by the average resistance of the landscape. In this case, multiplying 50,000 meters by 10 would yield a maximum dispersal distance of 500,000 in cost units. If not, you can simply use the maximum dispersal distance of your species in meters.

Software

This software is based on Python and C++ libraries for the main base functions. We built an R wrapper for each of the Python functions. At the same time, we provide a R-Shiny front end so users can use the tool with minimum code. The software can be installed locally in Windows, Linux, MacOS, and also can be hosted on web servers.

The software consists of four levels:

- Python: Base functions. Each of the functions is represented in a separate python file. Some other ancillary files are also required
- R: The R wrapper of the python functions. Contains the conventional R package structure.
- Shiny: Is the “front end” of the DSS based on R functions and rely on the R package cola functions.
- Web application: It is the DSS deployed on a cloud server.

The structure of the whole components of the software is presented below:

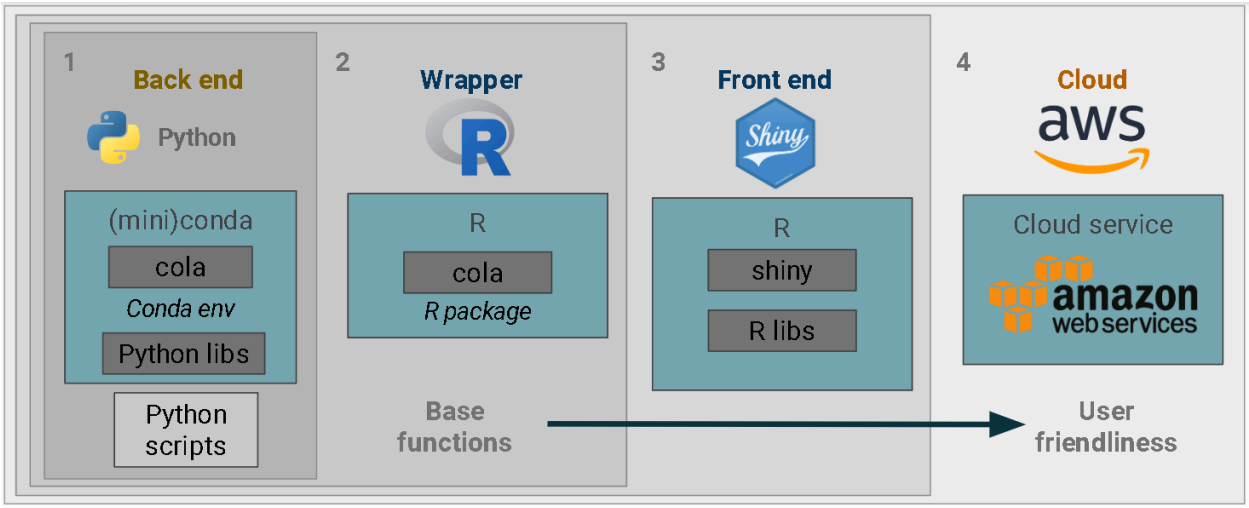


Figure 2. The four software modules. Starting at the left, Python (conda) is the base of the algorithms. Each module to the right is an add-on to the previous one.

Each of the elements depends on the modules at their left, having the python module as the baseline. In your local machine, you will be able to install modules one, two and three.

There are numerous libraries required for R and Python functioning. Below is a list of the modules and packages.

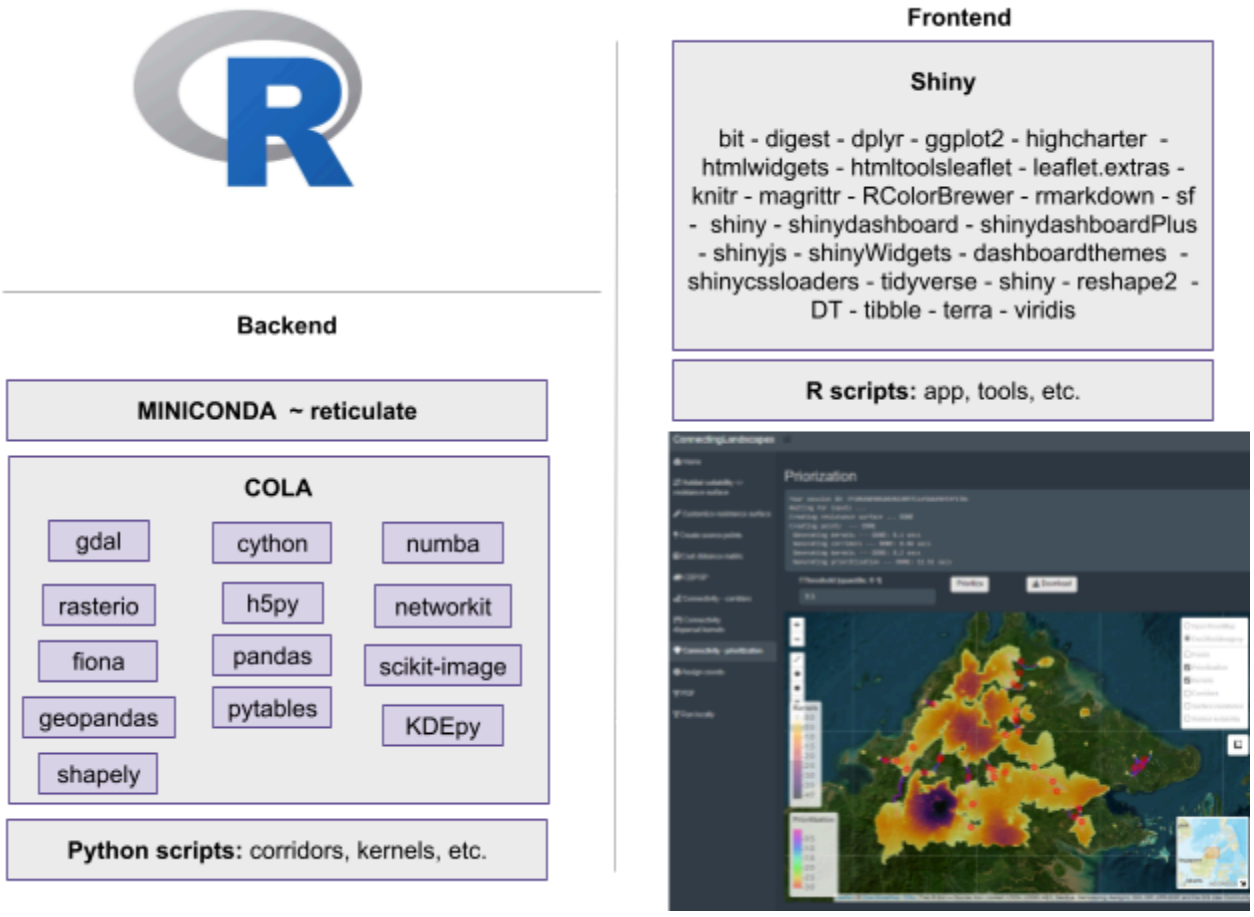


Figure 3. List of Python modules for running the back end, and R packages required for the front end or DSS.

Installation guide

There are two main steps for installing ***CoLa*** on your computer. The first one requires Python under the Conda environment manager, the second step is based in R. Remember that the R modules require that Python files are installed correctly.

The different system configurations require several approaches to installing Conda and R. Please refer to this link for installing Conda, and this one for installing R in your computer. Refer to the **appendix 1** for Windows instructions. After both, Conda and R are installed on your computer.

Installing CoLa conda environment (specialized users)

This option is automated into the R functions in the next section. Follow these steps if you **only want the Python module (1/4) of CoLA. Otherwise, you can skip this section and move onto the next one, "Installing CoLa R package".**

At the Conda prompt use the following command to install the *cola* conda environment with the required libraries:

```
conda create -n cola gdal h5py numexpr rasterio pytables pandas cython
numba networkit fiona shapely geopandas kdepy scikit-image
```

Table 2.

This instruction creates the *cola* environment. The next words are the name of the required libraries

In the appendix, we guide those program's installations.

Installing CoLa R package (most users)

The following R instructions install miniconda and all dependencies required, modules one through three, so the previous section is not required unless you want to run algorithms from a Python IDE (e.g. Spyder). When installing the cola package, some options will come up on the console. Choose the option "no to update libraries".

On the R console type the following commands:

```
# Install the devtools package
If (!require) {install.packages("devtools")}

# Install cola package. Choose option 3 None
devtools::install_github('connectingLandscapes/cola', dependencies = NA,
upgrade = 'never')

library(cola)
cola::setup_cola()
```

Once the package is installed, we need to set up *CoLa*, and install Python dependencies, and define internal paths. There are five (5) main steps during the installation.

```
# Set up cola
cola::setup_cola_dss()
```

The installation process has five main steps in R:

- Step 1/5: Installing & checking the `reticulate` R package, required to use the Python scripts in R
- Step 2/5 Installing & checking miniconda: the python executable that will execute the python scripts
- Step 3/5 Installing & checking conda environment: Creates a particular miniconda environment with the configuration
- Step 4/5 Installing & checking conda modules: Download and installing specific Python modules required for running the functions.
- Step 5/5 Setting up local variables: Set up the internal paths and instructions that allow R to communicate with the particular conda environment, modules and scripts.

You should have the following message in the console:

```
 +Step 1/5: Installing & checking reticulate R package
    `reticulate` installed already!
  +Step 2/5 Installing & checking miniconda
    miniconda found at C:/Users/Admin/AppData/Local/r-miniconda!
  +Step 3/5 Installing & checking conda environment
    `cola` conda environment installed in
C:\Users\Admin\AppData\Local\r-miniconda\envs\cola/python.exe
    `cola` conda environment is named correctly!
    The python version is Python 3.9.19
  +Step 4/5 Installing & checking conda modules
    All required conda modules installed!
  +Step 5/5 Setting up local variables
    === Ready to connect landscapes! ===
```

Please refer to the Installation details webpage. In the case of any error, you can check the installation process, and which of the steps presented problems:

```
cola::diagnose_cola()
```

Table 6.

Which should return:

```
We will look for errors. Running `cola::setup_cola()` should help you to
configure the package.
 Please refer to
https://github.com/connectingLandscapes/cola/blob/main/known-issues.md for
more details.
Here the diagnostic: (please wait a moment)
  1. `reticulate` package installed. Evaluating next step
  2. `miniconda` software installed. Evaluating next step
  3. `cola` conda environment installed. Evaluating next step
  4. All `cola` conda environment packages installed. Evaluating next step

 === All dependencies and requirements installed ===

Look for further details in the repository documentation
```

Table 7.

Different computer configurations might lead to problems. On this page, we have documented some common errors and provide some potential solutions. Take a look if run into any issues installing CoLa on your computer.

Once the R package (CoLa module 2/3), you can install the DSS front end. In this step, R will download all the required R packages. This might take some minutes.

```
cola::setup_cola_dss()
```

Table 8.

Which should return:

```
   === All libraries required for COLA's DSS installed ===
```

Table 9.

At this point the three software modules are installed in your local machine.

Set up your local path, memory limits and resample parameters

There's some parameters we can customize for CoLa performance, in each of the following parameters:

- Temporal directory `COLA_DATA_PATH`: where a folder will be created for each DSS session, and will be used to save the results. By default we use R temporal folder `tempdir()`. Keep the last slash or backslash and the quotes in the new path.
- Number of cores `COLA_NCORES`: To be used for running the programs. Default is 1.
- Maximum size of the layer `COLA_DSS_UPL_MB`: Size to load in the shiny dashboard in megabytes (MB) units. Default is 250.
- Maximum resolution for the layer representation `COLA_VIZ_THREs_PIX`, `COLA_VIZ_RES_NCOL, COLA_VIZ_RES_NROW`: The original layer created or uploaded will be resampled in the front end leaflet visor. This value is the upper threshold for pixel numbers in the layer. Layers with the number of pixels above this number will be resampled to c(1000, 1000) pixels. The default value is 1000000 for the threshold, and 1000 for the resampling size in columns and rows. Increase this number to allow your machine to load bigger original files in the front end. Decrease this number to avoid loading big files in your front end.

Run the following instructions for installing the system:

```
## Open the file in the tex editor
file.edit(file.path(Sys.getenv("HOME"), ".Renviron"))

# After saving the modifications in the previous file, restart the session
.rs.restartR()

# Check if the changes were made. If not, close R.
Sys.getenv(c('COLA_PYTHON_PATH', 'COLA_SCRIPTS_PATH'))
Sys.getenv(c('COLA_DATA_PATH', 'COLA_SCRIPTS_PATH', 'COLA_DSS_UPL_MB',
'COLA_VIZ_THRES_PIX', 'COLA_VIZ_RES_NCOL', 'COLA_VIZ_RES_NROW', 'COLA_NCORES' ))
```

Updating CoLa

CoLa might have continuous updates, developments, and bug fixing. If you want to update the software to the latest version, run the following commands:

```
## Restart your Rstudio session. Works only in Rstudio, not in R
.rs.restartR()

## Remove CoLa scripts (this don’t delete conda installations or R packages)
remove.packages('cola')

## Restart your Rstudio session. Better if you close R/Rstudio
devtools::install_github('connectingLandscapes/cola', dependencies = NA, upgrade =
'never') ## option 3: None
```

T

This process should be way faster than the initial installation since we are updating only the R scripts, but the remaining Python configuration is the same.

Removing CoLa

CoLa is installed in your computer in two instances: The R library and the conda environment. CoLa backend can still work without the R package, but the conda environment is required for the R package. For delete the software from your computer you need to, run the following commands:

```
## Remove the R package
remove.packages('cola')

## Remove CoLa conda environment
reticulate::conda_remove('cola')
```

To force the package uninstalling you can also delete the full folders, but this is not the recommended way to do it.

Running the functions

The functions need to be executed in the command line using Python directly or through R. Use complete paths for the arguments. We need to point to three components in the sentence:

- **Python version**: The executable name or path. You might have several Python installations on your computer, but we need one that can use the particular libraries, environment, and configuration. That's the reason for using **conda**.

- **Python script**: The full file path with the instructions or script of the analysis we want to run
- **Parameters**: The input files and specific conditions to make the analysis

```
python script.py input output parameter1 parameter2 ...
```

Table 10.

An example in Windows looks like the following:

```
C:\Users\YOURUSER\AppData\Local\r-miniconda\envs\cola\python.exe
C:\Users\cola\python\s2res.py C:\Users\cola\data\sampleTif.tif
C:\Users\cola\data\out.tif 0 1 10 None None
```

Table 11.

An example in Linux looks like the following:

```
/home/shiny/.local/share/r-miniconda/envs/cola/bin/python
/home/shiny/R/4.4/cola/python/s2res.py /home/sampleTif.tif /data/out.tif 0
1 10 None None
```

Table 11.

The path in your computer will be different. Adapt the command accordingly.

The R instructions look different, but at the back end, it is running the same instruction as the previous one. You can run the functions as the command line, bui also using R as follows:

```
library(cola)
# Create a surface resistance from habitat suitability
new_sr <- s2res_py(intif = '/path/to/hs.tif', outtif = 'new_sr.tif', minval
= 0, maxval = 1, maxout = 10, shape = 1, nodata = NULL, prj = 'None')
```

Table 12.

CoLa Python scripts require a specific order for the arguments. Be sure to provide them in the order given in this tutorial. In the same way, R requires an exact order for the arguments if they are not named.

First, let's check the Python executable. Cola has a Python script that helps you know if it is the proper one. You should be able to find the conda folders in your computer, some common locations for the correct **Python** version include:

Windows:

- C:\Users\YOURUSER\AppData\Local\r-miniconda\envs\cola\python.exe
- C:\ProgramData\anaconda3\envs\cola\python.exe

Linux:

- /home/user/anaconda3/envs/cola/bin/python
- /home/user/miniconda3/envs/cola/bin/python
- /home/user/.local/share/r-miniconda/envs/cola/bin/python

You can run the functions as the command line, but also using R as follows:

```
library(cola)
# Create a surface resistance from habitat suitability
new_sr <- s2res_py(intif = '/path/to/hs.tif', outtif = 'new_sr.tif', minval
= 0, maxval = 1, maxout = 10, shape = 1, nodata = NULL, prj = 'None')
```

Table 13.

Functions

This section also provides instructions about running each function separately on the command line and the step-by-step.

The software can produce independent products, which can also be used as inputs for other functions:

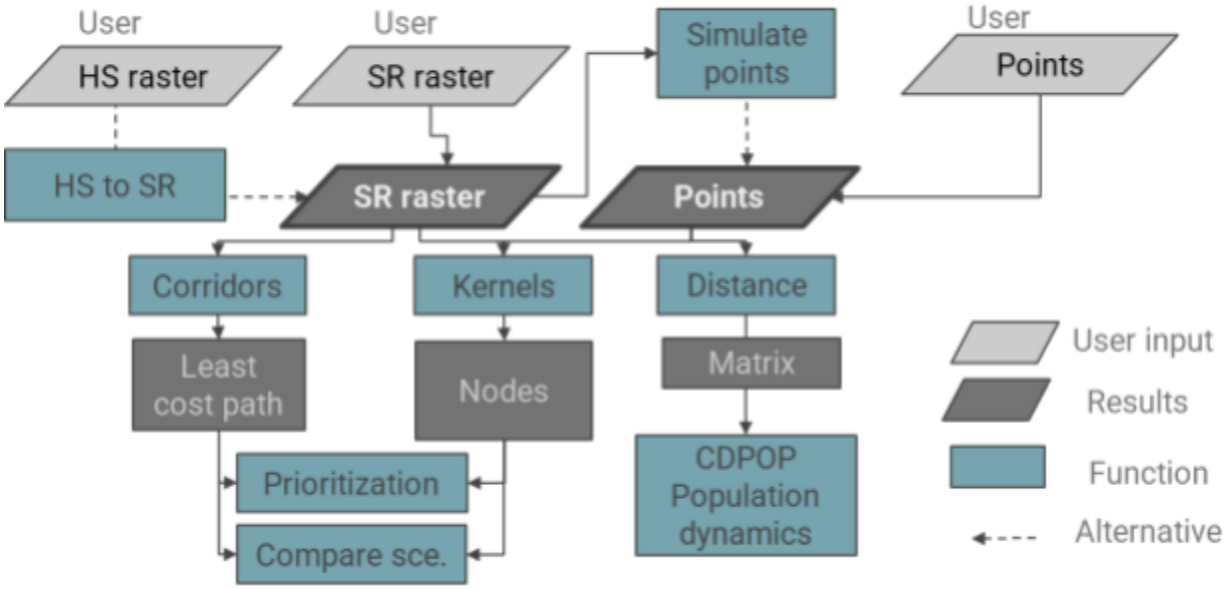


Figure 3. Connections and workflow between CoLa functions. In some cases you can provide the input or create them with other CoLa functions

The complete workflow can be be executed in the following order:

- Load Habitat suitability (HS) or use sample HS data
- Convert HS to surface resistance (SR), or load a SR
- Edit the SR layer
- Provide or load a point layer, or simulate points based on the HS or SR layers
- Create kernels using the SR and points layer
- Create corridors using the SR and points layer
- Get a corridors prioritization based on the corridors, kernels, and SR
- Compare multiple kernels or corridors
- Run CDPOP model

One possible pipeline looks like:

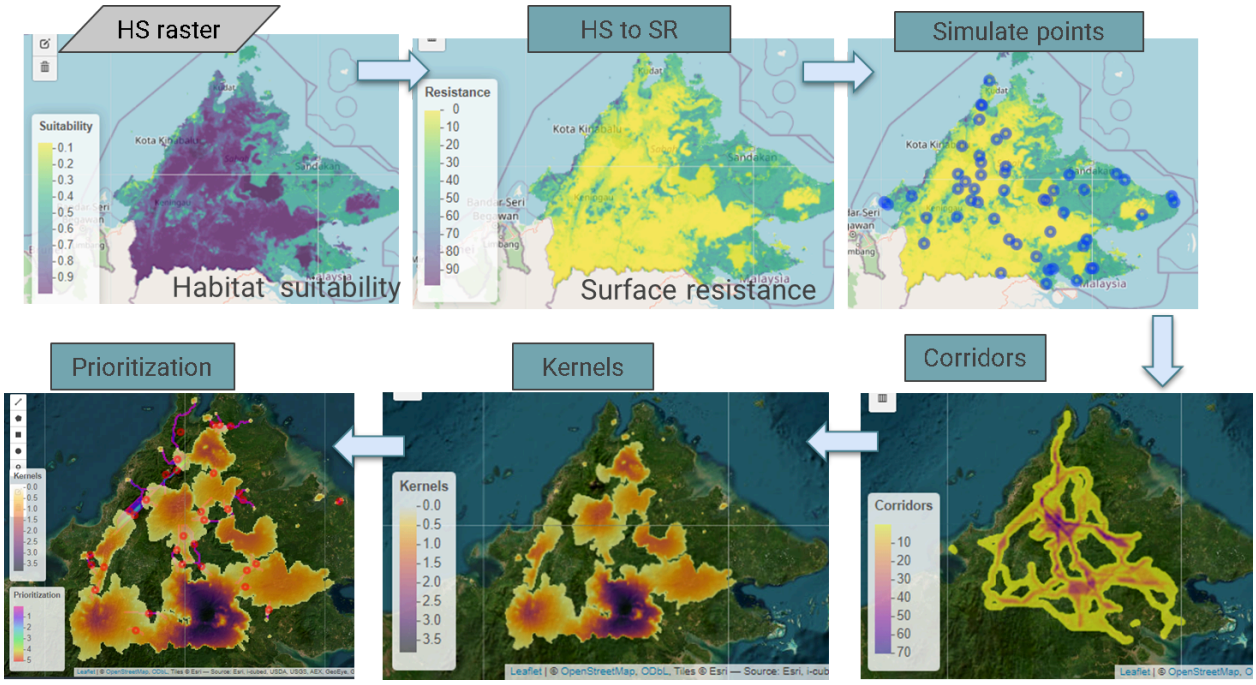


Figure 4. Example of CoLa functions results, only using Habitat Suitability (HS) raster as input

There are multiple potential combinations of functions. The DSS provides connections among the functionalities, in a way that the outputs of some algorithms are the inputs for the others. The following diagram shows the existing links among `cola` functionalities.

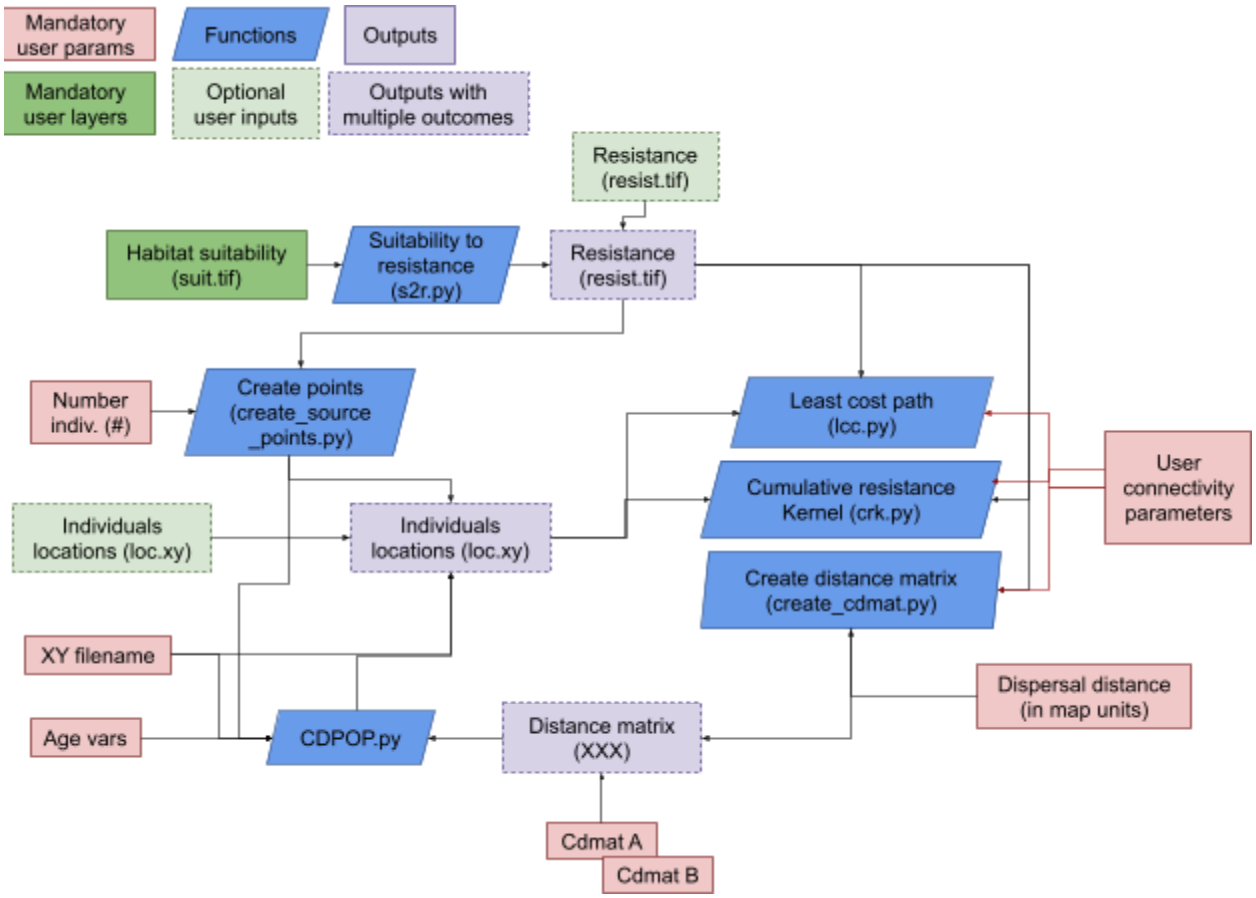


Figure 24. Diagram of the connections among `cola` *functionalities.*

There are five COLA functionalities, mainly by Python scripts, requiring specialized libraries managed by conda environments. Next, the details, parameters, and instructions to use them.

General aspects of the functions:

- Inputs are file paths, not internal R spatial objects
- The functions overwrites the outputs if they exists already
- Arguments follow a specific order in Python and CoLa. Follow the parameters order in the documentation.
- Most of R functions return a list of two slots:
    - file: The file path generated. In the case of multiple files (ESRI Shapefiles), only the file with ".shp" extension will be returned. Other expected files will contain different extensions but same basename
    - log: message containing any log from the original python script

- The function requires a projected coordinate system. No on-the-fly projections are implemented inside the functions.
- The output files paths are provided as inputs. The returning value will not provide new file names.
- The last two arguments of the functions are python executables established by default by the R package:
    - `py: Sys.getenv("COLA_PYTHON_PATH")`, the executable instruction for R using Python.
    - `pyscript: system.file(package = 'cola', 'python/SCRIPT-HERE.py')`, the location of the particular python script

The inputs need to meet the following criteria:

- Saved on a file on the disk (no saved on RAM), under a path with no special characters or spaces
- Have a projected coordinate system. Don't use a geographical coordinate system
- Raster files should contain only a single band
- Raster files should have a valid NoData value

> Habitat suitability to surface resistance

Converts a given raster into a new one using a mathematical transformation function and numeric range. `s2res.py` in the command line, or `s2res_py( )` in R

Parameters

- **[1] Suitability surface (intif):** String. File path to the input raster
- **[2] Output raster file name (outtif):** String. File path of the output surface resistance
- **[3] Suitability input grid minimum value (minval):** Numeric. The lower value on the input raster to cut off. Pixels with values under the given number will be ignored. In the front end/DSS the values are automatically derived from the input file.
- **[4] Suitability input grid maximum value (maxval):** Numeric. The upper value on the input raster to cut off. Pixels with values under the given number will be ignored. In the front end the values are automatically derived from the input file.
- **[5] Maximum resistance value (maxout)**: Numeric. This is the maximum resistance value after transformation from suitability. The default value is 100. The minimum value is set to 1.
- **[6] Shape parameter (shape):** Numeric. A statistical parameter that defines the transformation pattern between the input and output. The shape value determines the relationship between suitability and resistance. For a linear relationship, use a value close to 0, such as 0.01. Positive values result in a greater increase in resistance as suitability declines. This is appropriate for animals that are more sensitive to the matrix in between habitats. Negative values result in a lesser increase in resistance as suitability declines. This is appropriate for animals that are less sensitive to the matrix between habitats. The more positive or more negative, the greater the effect on the shape of the relationship. Values generally range between +10 and -10, 0 is not allowed.
- **[7] No data value (nodata):** Numeric. The no data value of the input file. For GeoTiffs, this is automatically determined. For text-based files like ASCII or RSG rasters, the user must input them. The default value is 'None'.
- **[8] Projection parameter (prj):** String. Projection information in the case the input raster [1] has no spatial projection. For GeoTiffs, this is automatically determined. For text-based files like ASCII or RSG rasters, the user must input them. Provide it as EPSG or ESRI string, e.g. "ESRI:102028". The default value is 'None'.

Output

Creates a raster layer with a minimum value of 1 and maximum value given the parameter 5. The internal R object is a list of two slots. The first one contains the path of the created raster, if any, and the second slot includes any function message or log, if any.

Usage

In the command line:

```
/path/to/python s2res.py myhabitatsuitability.tif outsurfresistance.tif 0
100 100 -9999
```

In R:

```
library(cola)
library(terra)
wd <- 'cola_examples'
dir.create(wd)
setwd(wd)
hs <- system.file(package = 'cola', 'sampledata/sampleTif.tif')
# hs <- system.file('/path/to/your/raster.tif')

srp30 <- s2res_py(intif = hs, outtif = 'hs_p3_0.tif', minval = 0, maxval = 1,
maxout = 10, shape = 1, nodata = NULL, prj = 'None')
```

Examples

Considering having the sample data set (REF HERE):

```
library(cola)
library(terra)
wd <- 'cola_examples'
dir.create(wd)
setwd(wd)
hs <- system.file(package = 'cola', 'sampledata/sampleTif.tif')
hs_rast <- terra::rast(hs)
plot(hs_rast, main = 'Habitat suitability')
```

Table 16.

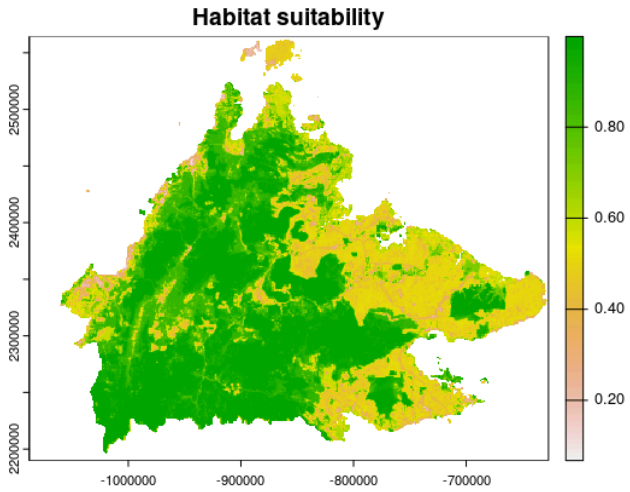


Figure 5.

Parameter 3: Minimum value to consider in the input raster

Console CMD:

```
path/to/python /path/to/s2res.py /path/to/sampleTif.tif hs_p3_0.tif 0 1 10 1 -9999
None
```

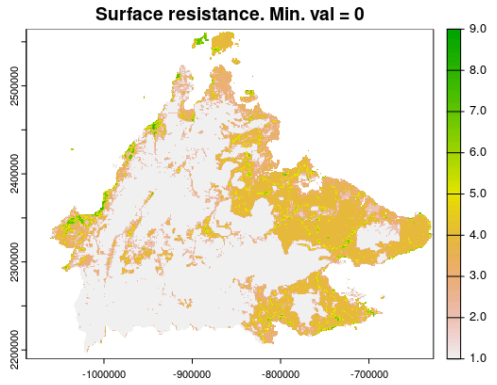
Table 17.

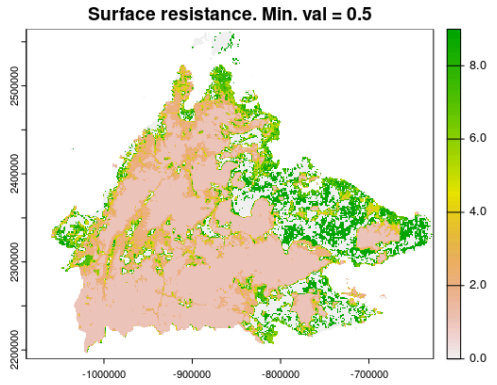
R CMD:

```
## Creates the surface resistance
sr <- s2res_py(intif = hs, outtif = paste0(wd, '/hs_p3_0.tif'), minval = 0, maxval
= 1, maxout = 10,  shape = 1,  nodata = NULL, prj = 'None')

# Create color ramp
breakss <- seq(0, 100, by = 10)
colx <- rev(terrain.colors(length(breakss)))

## Plot The result
plot(rast(sr$file), main = 'Surface resistance, breaks = breakss, col = colx)
```

Table 18.

Parameters	Result

minval: 0
 	
cmd:
`0 1 10 1 -9999 None`
 	
R:
`minval = 0, maxval = 1, maxout = 10, shape = 1, nodata = NULL, prj = 'None'`
	

minval: 0.5
 	
cmd:
`0.5 1 10 1 -9999 None`
 	
R:
`minval = 0.5, maxval = 1, maxout = 10, shape = 1, nodata = NULL, prj = 'None'`
	

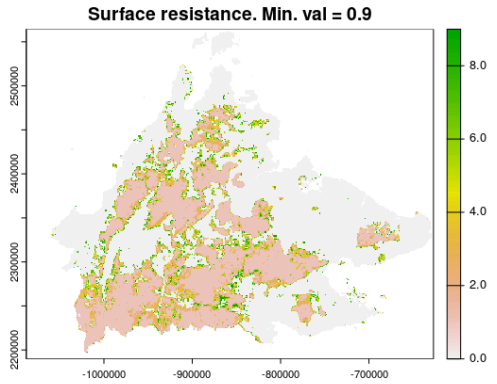
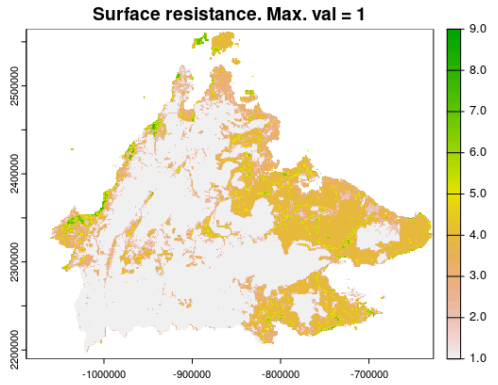
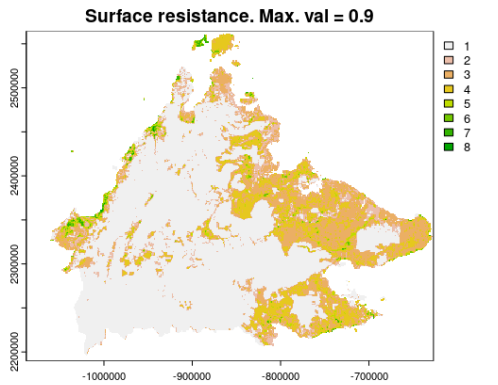
minval: 0.9
 	
cmd:
`0.9 1 10 1 -9999 None`
 	
R:
`minval = 0.9, maxval = 1, maxout = 10, shape = 1, nodaa = NULL, prj = 'None'`
	

Table 19.

Parameter 4: Maximum value to consider in the input raster

Parameters	Result

maxval: 1
 	
cmd:
`0 1 1 1 -9999 None`
 	
R:
`minval = 0, maxval = 1, maxout = 10, shape = 1, nodata = NULL, prj = 'None'`
	

maxval: 0.9
 	
cmd:
`0 1 0.9 1 -9999 None`
 	
R:
`minval = 0, maxval = 0.9, maxout = 10, shape = 1, nodata = NULL, prj = 'None'`
	

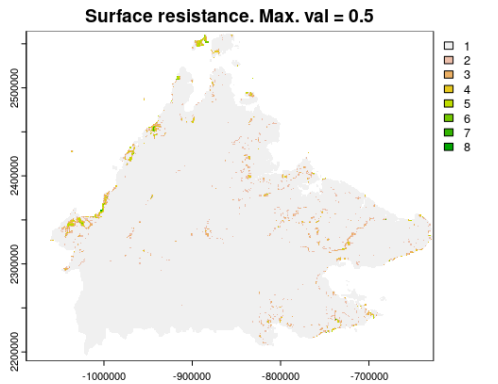
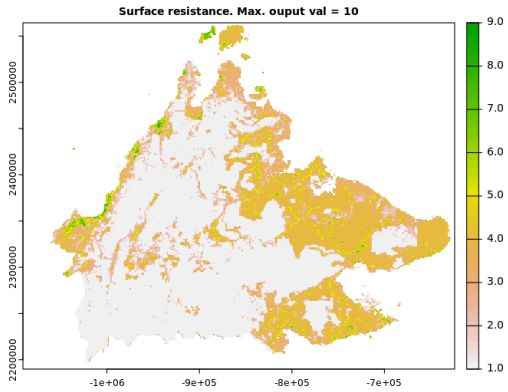
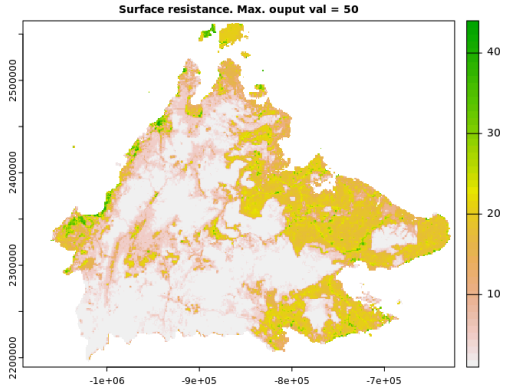
maxval: 0.5
 	
cmd:
`0 1 0.5 1 -9999 None`
 	
R:
`minval = 0, maxval = 0.5, maxout = 10, shape = 1, nodata = NULL, prj = 'None'`
	

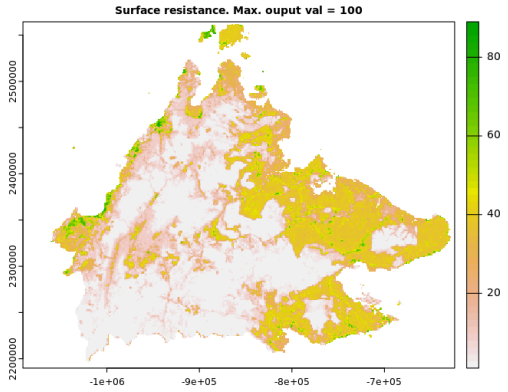
Table 26.

Parameter 5: Maximum value to consider in the input raster

Parameters	Result

maxout: 10
 	
cmd:
`0 1 10 1 -9999 None`
 	
R:
`minval = 0, maxval = 1, maxout = 10, shape = 1, nodata = NULL, prj = 'None'`
	

maxout: 50
 	
cmd:
`0 1 50 1 -9999 None`
 	
R:
`minval = 0, maxval = 1, maxout = 50, shape = 1, nodata = NULL, prj = 'None'`
	

maxout: 100
 	
cmd:
`0 1 100 1 -9999 None`
 	
R:
`minval = 0, maxval = 1, maxout = 100, shape = 1, nodata = NULL, prj = 'None'`
	

Parameter 6: Maximum value in the output raster

Considering two sets of values, the habitat suitability (HS) and its opposite, surface resistance (SR), it is possible to provide more relevance to small HS values in the transformation (shape

parameter bigger than zero) or make more abundant higher HS pixels (shape parameter smaller than zero).

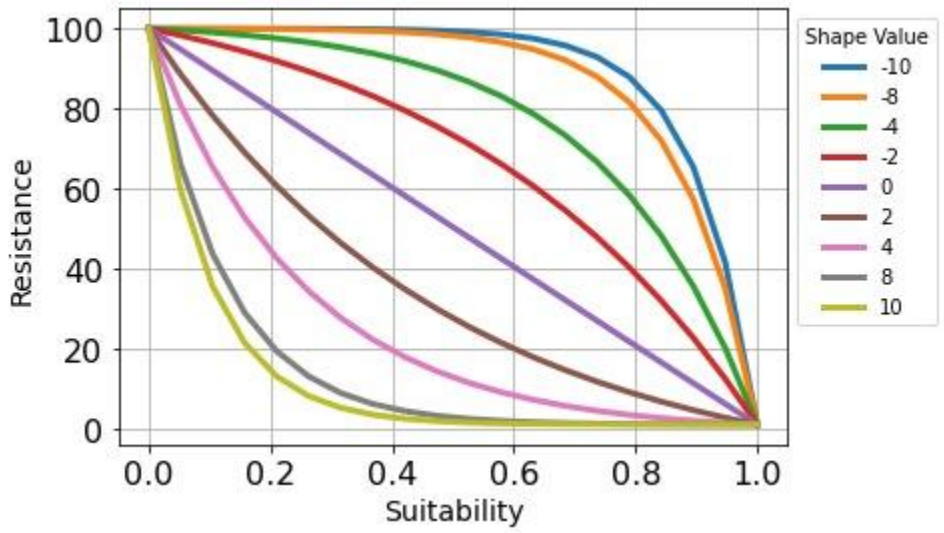
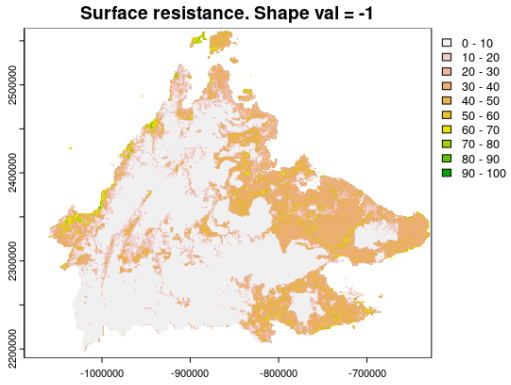


Figure 12. Transformation shape parameter. Values above zero (2, 4, 8, 10) converts most of the HS values into low SR. Shape parameter values below zero (-2, -4, -8, -10) returns higher SR.

Parameters	Result

shape: -1
 	
cmd:
`0 1 100 -1 -9999 None`
 	
R:
`minval = 0, maxval = 1, maxout = 100, shape = -1, nodata = NULL, prj = 'None'`
	

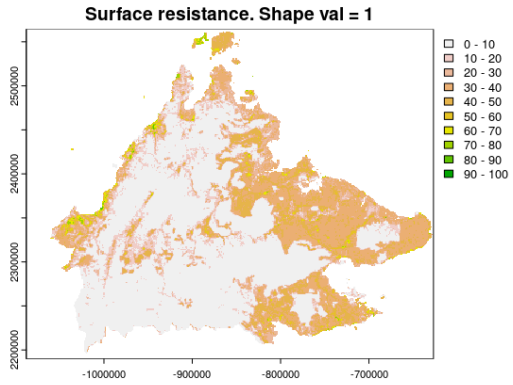
shape: 1

cmd:

```
0 1 100 1 -9999 None
```

R:

```
minval = 0, maxval = 1, maxout = 100,
shape = 1, nodata = NULL, prj = 'None'
```



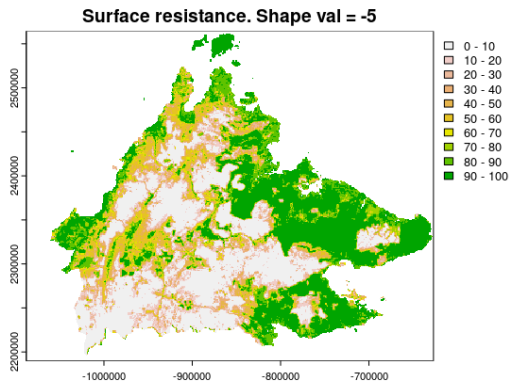
shape: -5

cmd:

```
0 1 100 -5 -9999 None
```

R:

```
minval = 0, maxval = 1, maxout = 100,
shape = -5, nodata = NULL, prj =
'None'
```



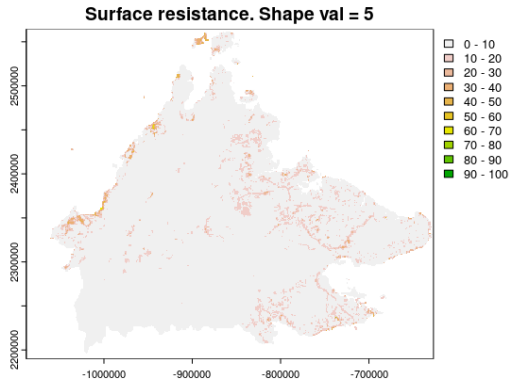
shape: 5

cmd:

```
0 1 100 5 -9999 None
```

R:

```
minval = 0, maxval = 1, maxout = 100,
shape = 5, nodata = NULL, prj = 'None'
```



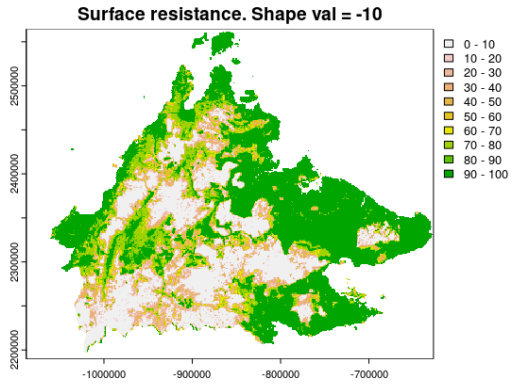
shape: -10

cmd:

```
0 1 100 -10 -9999 None
```

R:

```
minval = 0, maxval = 1, maxout = 100,
shape = -10, nodata = NULL, prj =
'None'
```



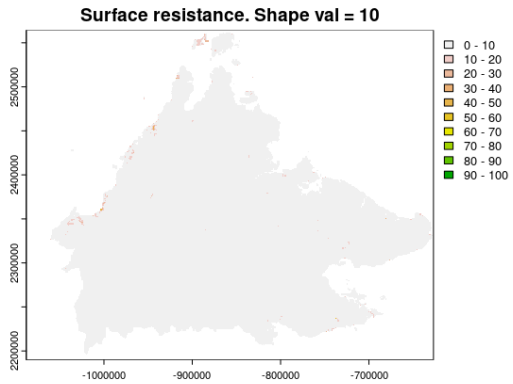
Parameter 6: 10

cmd:

```
0 1 100 10 -9999 None
```

R:

```
minval = 0, maxval = 1, maxout = 100,
shape = 10, nodata = NULL, prj =
'None'
```



Common errors

- Using negative values for the shape (parameter 5) parameter
- Using values for the minimum (parameter 3) and maximum (parameter 5) out of the real input raster range
- Using zero as shape parameter

> Simulate source points

Create a set of points given a range of values and a raster layer. The script generates points over the pixels inside the numerical range provided as an augment. You can use it by `create_source_points.py` in the command line or `points_py( )` in R. The script samples the pixels, i.e., more likely to sample higher value pixels.

Parameters

- **[1] Surface (intif):** String. File path to the input raster. Requires a GeoTIFF file with square pixels and cost units as distance units.
- **[2] Output point layer (outtif):** String. File path to the output point layer. Written in ESRI Shapefile format.
- **[3] Minimum value (minval):** Numeric. The lower value of the pixels in the raster to consider to simulate the points.
- **[4] Maximum value (maxval):** Numeric. The upper value of the pixels in the raster to consider to simulate the points.
- **[5] Number of points (npoints):** Numeric. Number of points to simulate.
- **[6] Is it suitable? (issuit).** String. 'Yes' (default) or 'No'. Indicates if the provided raster [1] is suitability. If so, the script will likely sample higher value pixels. If 'No', will
- **[7] Update CRS (upcrs):** String. Projection information in the case the input raster [1] has no spatial projection. For GeoTiffs, this is automatically determined. For text-based files like ASCII or RSG rasters, this must be input by the user. Provide it as EPSG or ESRI string e.g. "ESRI:102028". Default value is 'None'.

Output

Creates an ESRI Shapefile vector shapefile. The internal R object is a list of two slots. The first one contains the path of the created shapefile, if any, and the second slot includes any function message or log, if any. The shapefile attribute table has two columns: X and Y coordinates fields.

Usage

In the command line:

```
/path/to/python /path/to/create_source_points.py inputHS.tif out_pts.shp
0.2 0.9 50 Yes None
```

In R:

```
library(cola)
library(terra)
wd <- 'cola_examples'
dir.create(wd)
setwd(wd)

hs_path <- system.file(package = 'cola', 'sampledata/sampleTif.tif')
# hs_path <- 'C:/path/to/raster.tif'
points_path <- system.file(package = 'cola', 'sampledata/samplePoints.shp')
# points_path <- 'C:/path/to/points.shp'

pts_result <- points_py(intif = hs_path, outshp = 'out_pts.shp', minval = 0.2,
maxval = 0.9, npoints = 50, issuit = 'Yes', upcrs = 'None')
```

Common errors

- Using a raster with geographical or no coordinate system
- Raster with pixels that are not square

Examples

Considering having the sample data set (REF HERE):

```
library(cola)
library(terra)
wd <- 'cola_examples'
dir.create(wd)
setwd(wd)

hs_path <- system.file(package = 'cola', 'sampledata/sampleTif.tif')

hs_rast <- terra::rast(hs)
plot(hs_rast, main = 'Habitat suitability')

out_pts <- points_py(intif = hs, outshp = '50_points_02_095.shp',
                     minval = 0.20, maxval = 0.95, npoints = 50)
```

Parameters	Result

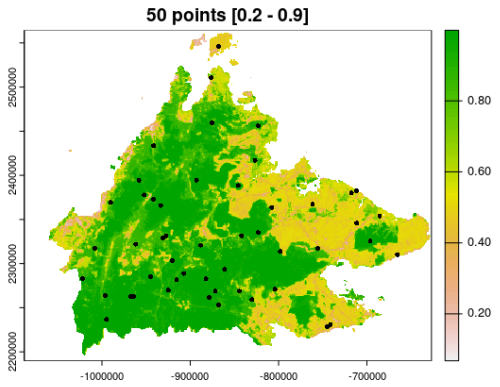
minval: 0.2
maxval: 0.9
npoints: 50

cmd:

```
0.2 0.9 50
```

R:

```
minval = 0.2, maxval = 0.9, npoints =
50
```



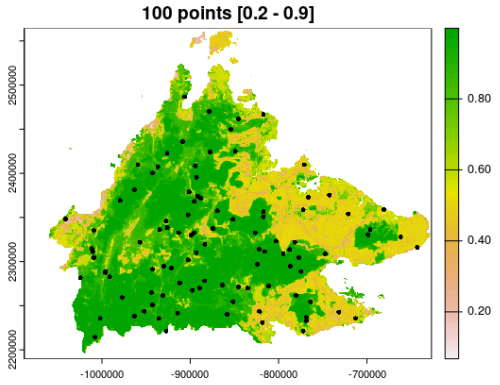
minval: 0.2
maxval: 0.9
npoints: 100

cmd:

```
0.2 0.9 100
```

R:

```
minval = 0.2, maxval = 0.9, npoints =
100
```



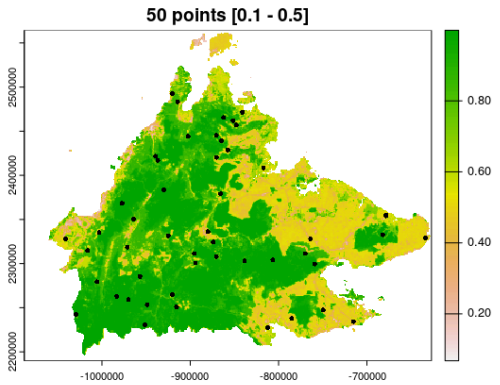
minval: 0.1
maxval: 0.5
npoints: 50

cmd:

```
0.1 0.5 50
```

R:

```
minval = 0.1, maxval = 0.5, npoints =
50
```



> Cumulative resistant kernels

Creates a continuous surface of cumulative paths that connects a set of points. `crk.py` in the command line, or `crk_py( )` in R.

Parameters

- [**1**] **Source points (inshp):** String. file path to the point layer. Spatial point layer (any ORG driver), CSV (X, Y files), or *.xy file
- [**2**] **Surface resistance (intif):** String. File path to the input raster. Requires a GeoTIFF file with square pixels and cost units as distance units
- **[3] Output raster file name (outtif):** String. File path of the output surface resistance
- **[4] Max. dispersal distance in cost units (maxdist):** Numeric. This is the maximum distance to consider when calculating kernels and should correspond to the maximum dispersal distance of the focal species. Values greater than this will be converted to 0 before summing kernels. For example, if the maximum dispersal distance of the focal species is 10 km, set this value to 10000.
- **[5] Kernel shape (shape)**: String. This determines how the probability of dispersal declines with distance from the focal point. `'linear'` implements the function `1 - (1/dThreshold) * d` where dThreshold is the specified distance threshold and d is the distance from the focal point. `'gaussian'` implements the function `exp(-1*((d**2)/(2*(dispScale**2))))` where d is the distance from the focal point and dispScale is equal to dThreshold/4. In future versions, users will be able to specify dispScale.
- **[7] Use transformation? (transf):** String. 'yes' or 'no'. If it is set to 'no', then no kernel volume transformation is applied and the next argument [8] is ignored. If it is set to 'yes', then the value in argument `volume` [8] is used to transform the kernel volume.
- **[8] Kernel volume (volume):** Numeric. If 1, the default, the resistant kernel value at the origin is 1 and no kernel volume transformation is applied. If > 1, the parameter value is used to scale distance values by a constant that is determined by the equation `kVol * 3/(pi*dThreshold**2)` where `kVol` is the kernel volume parameter, dThreshold is the specified distance threshold, and pi is the mathematical constant pi. The constant is then multiplied by the distances to the focal point resulting in a scaled kernel volume.
- **[9] Number of cores (ncores):** Numeric. Number of CPU cores to run the analysis. Default value is 1.

- **[10] Projection string (crs):** String. Projection information in the case the input raster [2] has no spatial projection. Provide it as EPSG or ESRI string e.g. "ESRI:102028". Default value is 'None'.

Output

Creates a raster layer: output_crk.tif a raster layer with a minimum value of 1 and maximum value given the parameter 5. The internal R object is a list of two slots. The first one contains the path of the created raster, if any, and the second slot includes any function message or log, if any.

Usage

In the command line:

```
/path/to/python /path/to/crk.py points.shp inputHS.tif out_crk.tif 125000
linear 1 1 None
```

In R:

```
library(cola)
library(terra)
wd <- 'cola_examples'
dir.create(wd)
setwd(wd)

hs_path <- system.file(package = 'cola', 'sampledata/sampleTif.tif')
points_path <- system.file(package = 'cola', 'sampledata/samplePoints.shp')

crk_result <- crk_py(inshp = points_path, intif = hs_path, outtif = 'out_crk.tif',
                     maxdist = 125000, shape = 'linear', volume = 1)
```

Examples

Considering having the sample data set (REF HERE):

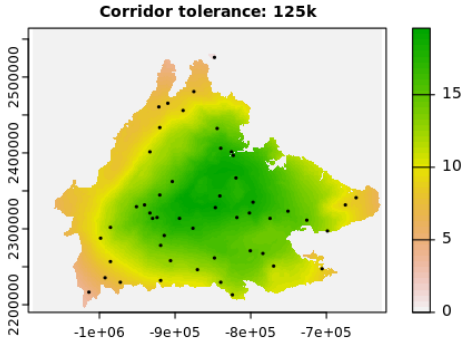
```
library(cola)
library(terra)
```

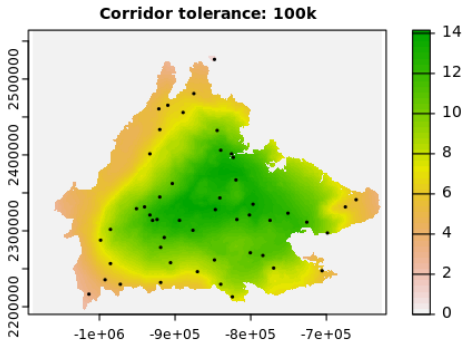
```
wd <- 'cola_examples'
dir.create(wd)
setwd(wd)

hs_path <- system.file(package = 'cola', 'sampledata/sampleTif.tif')
points_path <- system.file(package = 'cola', 'sampledata/samplePoints.shp')

hs_rast <- terra::rast(hs_path)
plot(hs_rast, main = 'Habitat suitability')
plot(terra::vect(points_path), add = TRUE)
```

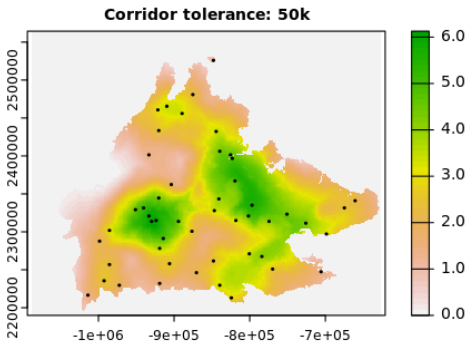
Parameter 4: Max. dispersal distance (cost units)

Defines how far the animals can move through the landscape. The higher the value, the more and farther routes can be established in the landscape.

Parameters	Result

maxdist: 125000
 	
cmd:
`125000 linear 1 1 None`
 	
R:
`maxdist = 125000, shape = 'linear', volume = 1`
	

maxdist: 100000
 	
cmd:
`100000 linear 1 1 None`
 	
R:
`maxdist = 125000, shape = 'linear', volume = 1`
	

maxdist: 50000
 	
cmd:
`50000 linear 1 1 None`
 	
R:
`maxdist = 50000, shape = 'linear', volume = 1`
	

Parameter 5: Kernel shape

Defines the shape of the kernel around the point. The following figure illustrates how the linear shape reaches further than the gaussian one.

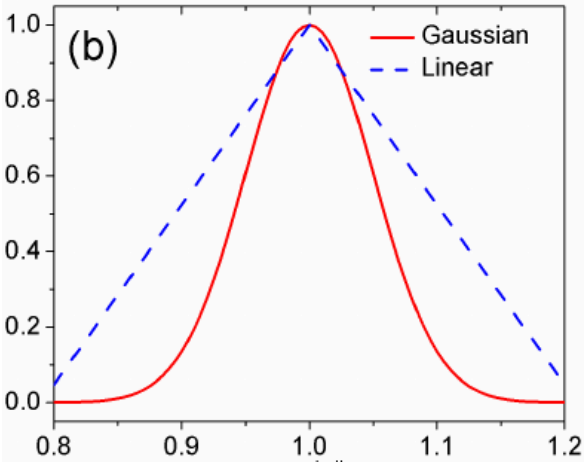
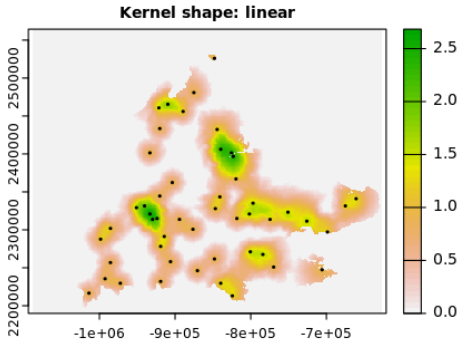
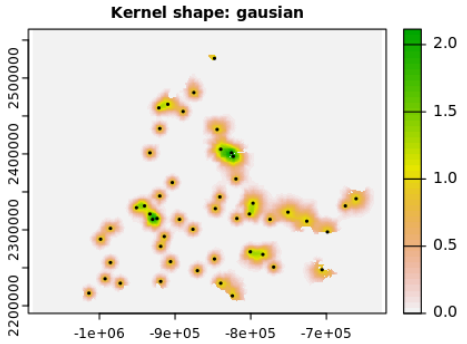


Figure 19. Taken from researchgate.net

Parameters	Result

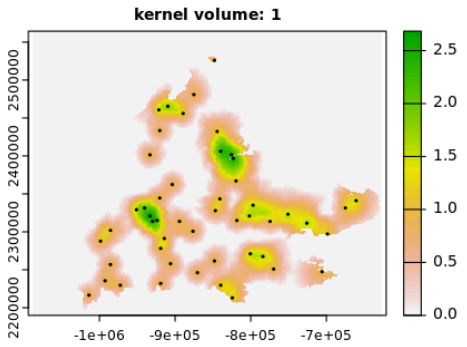
shape: linear
 	
cmd:
`20000 linear 1 1 None`
 	
R:
`maxdist = 20000, shape = 'linear', volume = 1`
	

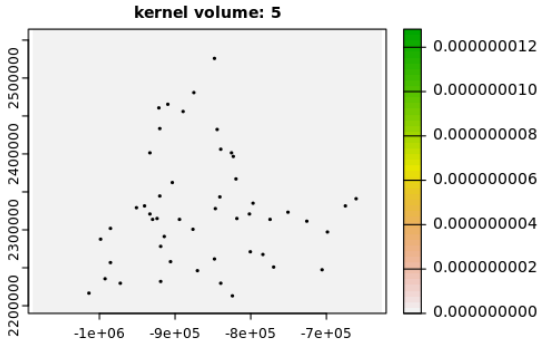
shape: gaussian
 	
cmd:
`20000 gaussian 1 1 None`
 	
R:
`maxdist = 20000, shape = 'gaussian', volume = 1`
	

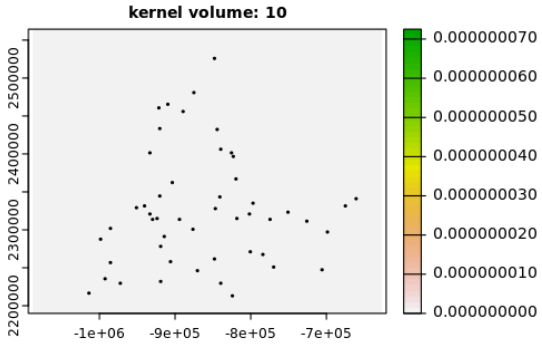
Parameter 6: Kernel volume

This description is adapted from the UNICOR User Manual - A constant kernel volume could be used if you are comparing different species. For example, a mobile species can travel farther producing a larger resistant kernel than a less-mobile species. If a constant kernel volume is enforced, then the volume of the kernel is essentially population size and species that have different mobility can be modeled at the same population size. When transform kernel volume is “yes”, then the kernel volume parameter is used on the transformed kernel resistant distance using the following equation - kernel volume * 3/(math.pi*kernel resistant distances^2). When transform kernel volume is no, then no volume transformation is applied. In this case, the center pixel of each kernel has a value of 1 (corresponding to the probability that an individual traverses that pixel). The probability of traversing surrounding pixels decays to zero at the maximum dispersal distance.

Parameters	Result

volume: 1
 	
cmd:
`20000 linear 1 1 None`
 	
R:
`maxdist = 20000, shape = 'linear', volume = 1`
	

volume: 5
 	
cmd:
`20000 linear 5 1 None`
 	
R:
`maxdist = 20000, shape = 'linear', volume = 5`
	

volume: 10
 	
cmd:
`20000 linear 10 1 None`
 	
R:
`maxdist = 20000, shape = 'linear', volume = 10`
	

Common errors

- Using negative values for the shape (parameter 5) parameter
- Using values for the minimum (parameter 3) and maximum (parameter 5) out of the real input raster range
- Using zero as shape parameter

- Using a multiband raster
- Using a raster with an unknown “NoData” value
- Using a raster with geographical or no coordinate system
- Raster with no squared pixels

> Corridors

Creates a continuous surface of cumulative paths that connects a set of points. `lcc.py` in the command line or `lcc_py( )` in R. Also, you can use two extra versions of this function to generate equivalent results. The first is a memory-safe version for big files: `lccHeavy_py( )` and `lcc_heavy.py`. The second one is for parallel computing: `lccJoblib_py( )` and `lcc_joblib.py`

Parameters

- [**1] Source points (inshp):** String. File path to the point layer. Spatial point layer (any ORG driver), CSV (X, Y files), or *.xy file
- [**2] Surface resistance (intif):** String. File path to the input raster. Requires a GeoTIFF file with square pixels and cost units as distance uni
- **[3] Output raster file name (outtif):** String. File path of the output surface resistance
- **[4] Max. dispersal distance in cost units (maxdist):** Numeric. This is the maximum distance to consider when calculating corridors and should correspond to the maximum dispersal distance of the focal species. For example, if the maximum dispersal distance of the focal species is 10 km, set this value to 10000. Values greater than this will be converted to 0 before summing corridors.
- **[5] Corridor smoothing factor (smooth)**: Numeric. The width of the window, in the number of cells, is used to smooth the output corridor surface. If no smoothing is desired, set it to 0. This parameter allows backward compatibility with the original UNICOR functionality, which runs a smoothing window over the least-cost path surface.
- [6] **Corridor tolerance in cost units (tolerance):** Numeric. This is the distance beyond the least-cost path that an animal might traverse when moving between source points. Larger values result in wider corridors.
- **[7] Number of cores (ncores):** Numeric. Number of CPU cores to run the analysis. The default value is 1.
- **[8] Projection string (crs):** String. Projection information in the case of the input raster [2] has no spatial projection. Provide it as EPSG or ESRI string e.g. "ESRI:102028". The default value is ‘None’.
- **[9] First temporal h5 file (not used in R):** String. Not used in R, but required in Python. Created as a temporal file internally in R but required in the Python command line. This file is created to store temporal structures in disk and avoid allocate the information on the RAM. This argument is used only in the R and Python version of `lcc_heavy( )` and `lcc_joblib( )`.

- **[10] Second temporal h5 file (not used in R):** String. Not used in R, but required in Python. Created as a temporal file internally in R but required in the Python command line. This file is created to store temporal structures in disk and avoid allocate the information on the RAM. This argument is used only in the R and Python version of `lcc_heavy( )` and `lcc_joblib( )`.
- **[11] Max GB ram allowed (maxram):** Numeric. Maximum gibabytes to use in your machine. The default value is 6.

Output

Creates a raster layer: output_lcc.tif a raster layer with a minimum value of 1 and maximum value given the parameter 5. The internal R object is a list of two slots. The first one contains the path of the created raster, if any, and the second slot includes any function message or log, if any.

Usage

In the command line:

```
/path/to/python /path/to/lcc.py points.shp inputHS.tif out_lcc.tif 150000 0
0 1 None
/path/to/python /path/to/lcc_heavy.py points.shp inputHS.tif out_lcc.tif
150000 0 0 1 None tempA.h5 tempB.h5 6
/path/to/python /path/to/lcc_joblib.py points.shp inputHS.tif out_lcc.tif
150000 0 0 1 None tempA.h5 tempB.h5 6
```

In R:

```
library(cola)
library(terra)
wd <- 'cola_examples'
dir.create(wd)
setwd(wd)

hs_path <- system.file(package = 'cola', 'sampledata/sampleTif.tif')
# hs_path <- 'C:/path/to/raster.tif'
points_path <- system.file(package = 'cola', 'sampledata/samplePoints.shp')
# points_path <- 'C:/path/to/points.shp'

lcc_result <- lcc_py(inshp = points_path, intif = hs_path, outtif = 'out_lcc.tif',
                  maxdist = 150000, smooth = 0, tolerance = 0)
```

Examples

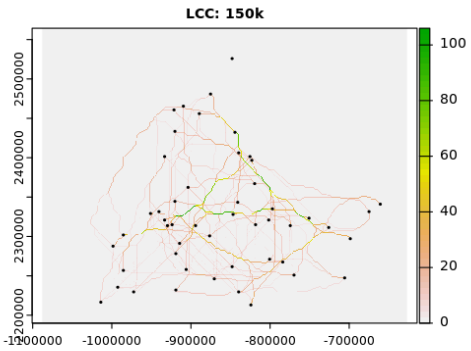
Considering having the sample data set (REF HERE):

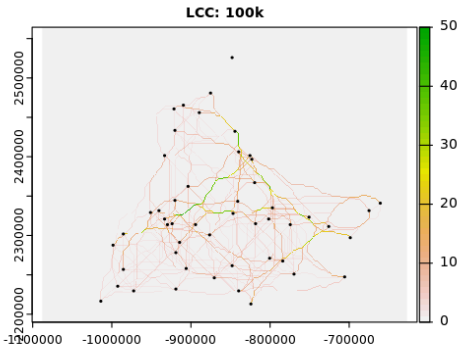
```
library(cola)
library(terra)
wd <- 'cola_examples'
dir.create(wd)
setwd(wd)

hs_path <- system.file(package = 'cola', 'sampledata/sampleTif.tif')
points_path <- system.file(package = 'cola', 'sampledata/samplePoints.shp')

hs_rast <- terra::rast(hs)
plot(hs_rast, main = 'Habitat suitability')
plot(terra::vect(points_path), add = TRUE)
```

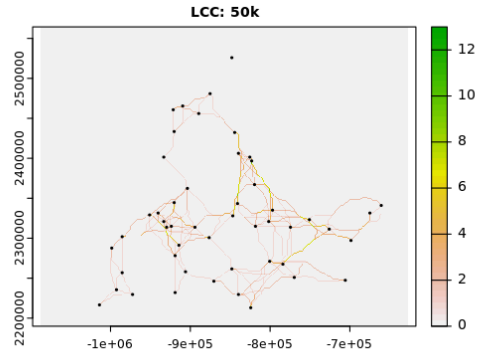
Parameter 4: Max. dispersal distance (cost units)

Defines how far the animals can move through the landscape. The higher the value, the more and farther routes can be established in the landscape.

Parameters	Result

maxdist: 150000
 	
cmd:
`150000 0 0 1 None`
 	
R:
`maxdist = 150000, smooth = 0, tolerance = 0`
	

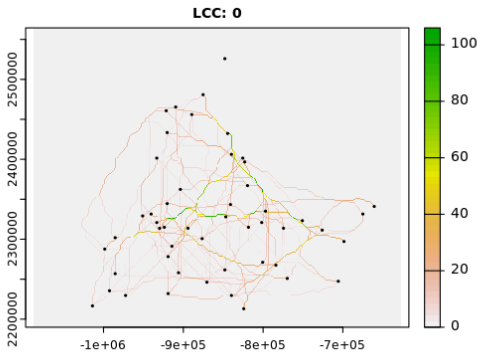
maxdist: 100000
 	
cmd:
`100000 0 0 1 None`
 	
R:
`maxdist = 100000, smooth = 0, tolerance = 0`
	

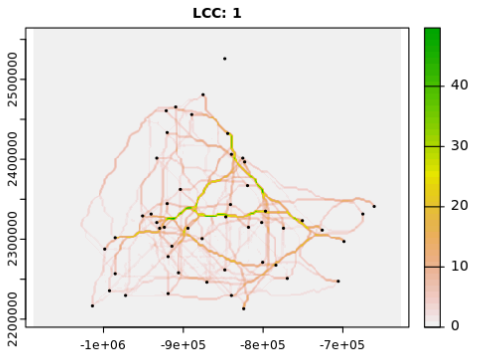
maxdist: 50000
 	
cmd:
`50000 0 0 1 None`
 	
R:
`maxdist = 50000, smooth = 0, tolerance = 0`
	

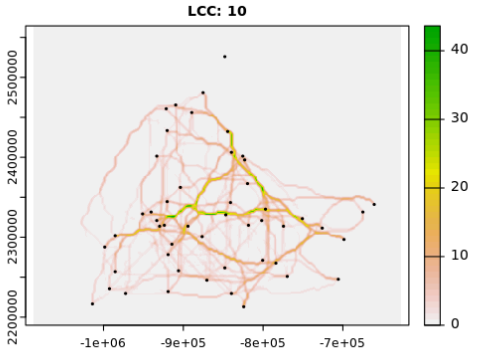
Parameter 5: Corridor smoothing factor

Define a moving window gaussian smoother. Lower values average corridor values over a smaller window, higher values average corridor values over a larger window. Provided primarily for backwards compatibility with UNICOR.

Parameters	Result

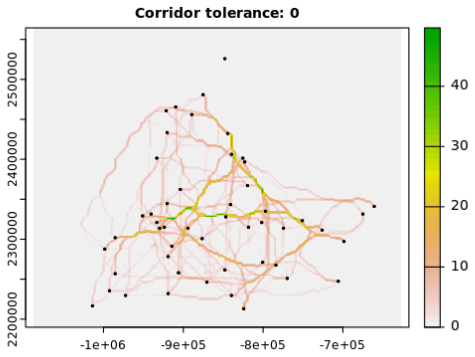
smooth: 0
 	
cmd:
`150000 0 0 1 None`
 	
R:
`maxdist = 150000, smooth = 1, tolerance = 0`
	

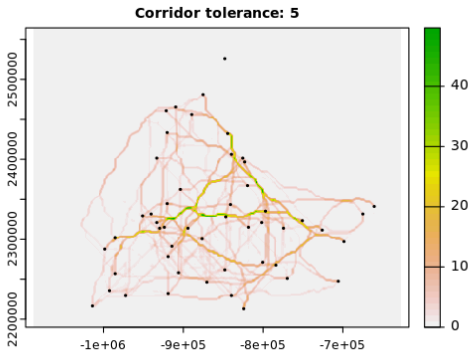
smooth: 1
 	
cmd:
`150000 1 0 1 None`
 	
R:
`maxdist = 150000, smooth = 1, tolerance = 0`
	

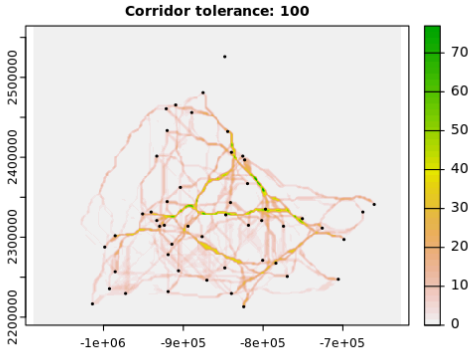
smooth: 10
 	
cmd:
`150000 10 0 1 None`
 	
R:
`maxdist = 150000, smooth = 10, tolerance = 0`
	

Parameter 6: Corridor tolerance (cost units)

Parameters	Result

tolerance: 0
 	
cmd:
`150000 1 0 1 None`
 	
R:
`maxdist = 150000, smooth = 1, tolerance = 0`
	

tolerance: 5
 	
cmd:
`150000 1 5 1 None`
 	
R:
`maxdist = 150000, smooth = 5, tolerance = 5`
	

tolerance: 10
 	
cmd:
`150000 1 10 1 None`
 	
R:
`maxdist = 150000, smooth = 10, tolerance = 10`
	

Common errors

- Using negative values for the shape (parameter 5) parameter
- Using values for the minimum (parameter 3) and maximum (parameter 5) out of the real input raster range
- Using a multiband raster

- Using a raster with an unknown “NoData” value
- Using a raster with geographical or no coordinate system
- Raster with no squared pixels

> Two way interaction matrix path

Creates a dispersal resistance matrix among a set of points. `create_cdmat.py` in the command line or `cdmat_py( )` in R.

Parameters

- [**1**] **Source points (inshp):** String. File path to the point layer. Spatial point layer (any ORG driver), CSV (X, Y files), or *.xy file
- [**2**] **Surface resistance (intif):** String. File path to the input raster. Requires a GeoTIFF file with square pixels and cost units as distance uni
- **[3] Output csv file name (outcsv):** String. Path of the output csv matrix
- **[4] Max. dispersal distance in cost units (maxdist):** Numeric. This is the maximum distance to consider when calculating kernels and should correspond to the maximum dispersal distance of the focal species. Values greater than this will be converted to 0 before summing kernels. For example, if the maximum dispersal distance of the focal species is 10 km, set this value to 10000.
- **[5] Number of cores (ncores):** Numeric. Number of CPU cores to run the analysis
- **[6] Projection string (crs):** String. Projection information in the case the input raster [2] has no spatial projection. Provide it as EPSG or ESRI string e.g. "ESRI:102028". Default value is 'None'.

Output

Creates a CSV matrix. The internal R object is a list of two slots. The first one contains the path of the created matrix, if any, and the second slot includes any function message or log, if any.

Usage

In the command line:

```
/path/to/python /path/to/create_cdmat.py points.shp inputHS.tif out_mat.csv
125000 1 None
```

In R:

```
library(cola)
library(terra)
```

```
wd <- 'cola_examples'
dir.create(wd)
setwd(wd)

hs_path <- system.file(package = 'cola', 'sampledata/sampleTif.tif')
# hs_path <- 'C:/path/to/raster.tif'
points_path <- system.file(package = 'cola', 'sampledata/samplePoints.shp')
# points_path <- 'C:/path/to/points.shp'

mat_result <- cdmat_py(inshp = points_path, intif = hs_path, outtif =
'out_mat.csv)
```

Common errors

- Using a multiband raster
- Using a raster with an unknown “NoData” value
- Using a raster with geographical or no coordinate system
- Raster with no squared pixels
- Using negative or zero distance

> Prioritize

Identify and rank existing corridors, assigning a higher rank to the shorter links that connect the bigger nodes. You can use it by `prioritize_core_conn.py` in the command line or `pri_py( )` in R.

Parameters

- **[1] Surface resistance (tif):** String. File path to the input raster. Requires a GeoTIFF file with square pixels and cost units as distance units
- **[2] Kernels (incrk):** String. File path to the input kernel GeoTiff raster file.
- **[3] Corridors (inlcc):** String. File path to the input corridors GeoTiff raster file.
- **[4] Masked output raster (maskedcsname):** String. The output raster file path of the resulting nodes, derived from the kernels raster and the quantile ([10] **threshold**)
- **[5] Output point layer (outshppoint):** String. The output point layer file path of the resulting corridor centroids in ESRI Shapefile format.
- **[6] Output polygon layer (outshppol):** String. The output polygon layer file path of the resulting corridors in ESRI Shapefile format.
- **[7] Output patch polygon layer (outshppatch):** String. The output polygon layer file path of the resulting patches in ESRI Shapefile format.
- **[8] Output patch raster layer (outtifpatch):** String. The output raster layer file path of the resulting patches in GeoTIFF format.
- **[9] Output corridor raster layer (outtif):** String. The output raster layer file path of the resulting corridors in GeoTIFF format.
- **[10] Threshold quantile (threshold):** Decimal. Numeric value between zero and one (0 - 1) to convert continuous kernels into discrete patches.
- **[11] Maximum distance (maxdist):** Numeric. This is the maximum distance to consider when calculating kernels and should correspond to the maximum dispersal distance of the focal species. Values greater than this will be converted to 0 before summing kernels. For example, if the maximum dispersal distance of the focal species is 10 km, set this value to 10000. This should be the same value used for the parameter [4], Max. dispersal distance in cost units in the least cost path function.

Output

Creates a set of files. The internal R object is a list of three. The first one contains the path of the prioritized GeoTIFF raster, the second one the centroids point vectorial layer, and the second slot includes any function message or log, if any.

Usage

In the command line:

```
/path/to/python /path/to/prioritize_core_conn.py inputHS.tif out_crk.tif
out_lcc.tif tempMask.tif out_pri_shp_pts.shp out_pri_shp_pol.shp
out_pri_shp_patch.shp out_pri_tif_patch.tif out_pri_tif_corridors.tif 0.5
150000
```

In R:

```
library(cola)
library(terra)
wd <- 'cola_examples'
dir.create(wd)
setwd(wd)

hs_path <- system.file(package = 'cola', 'sampledata/sampleTif.tif')
# hs_path <- 'C:/path/to/raster.tif'
points_path <- system.file(package = 'cola', 'sampledata/samplePoints.shp')
# points_path <- 'C:/path/to/points.shp'

out_pts <- points_py(intif = hs_path, outshp = pts_path,
                     Smin = 0.20, smax = 0.95, npoints = 50)

out_crk <- crk_py(inshp = pts_path, intif = hs_path, outtif = 'out_crk.tif',
                  maxdist = 20000, shape = 'linear', volume = 10)
out_lcc <- lcc_py(inshp = pts_path, intif = hs_path, outtif = 'out_lcc.tif',
                maxdist = 150000, smooth = 0, tolerance = 0)

## Run Prioritization
out_pri <- pri_py(tif = hs_path, incrk = out_crk$file, inlcc = out_lcc$file,
                  maskedcsname = 'temp_mask.tif',
                  outshppoint = 'out_pri_shp_pts.shp',
                  outshppol = 'out_pri_shp_pol.shp',
```

```
                  outshppatch = 'out_pri_shp_patch.shp',
                  outtifpatch = 'out_pri_tif_patch.tif',
                  outtif = 'out_pri_tif_corridors.tif',
                  threshold = 0.5,
                  maxdist = 150000)
```

Common errors

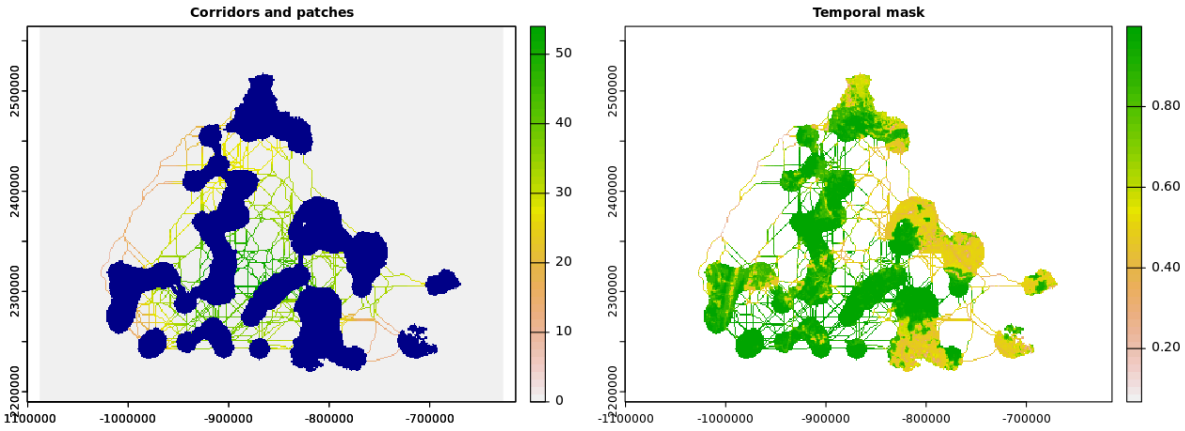
- Using a raster with geographical or no coordinate system.
- Raster with no squared pixels.
- Threshold value to convert kernel layer into paths is too high and results in a single patch.

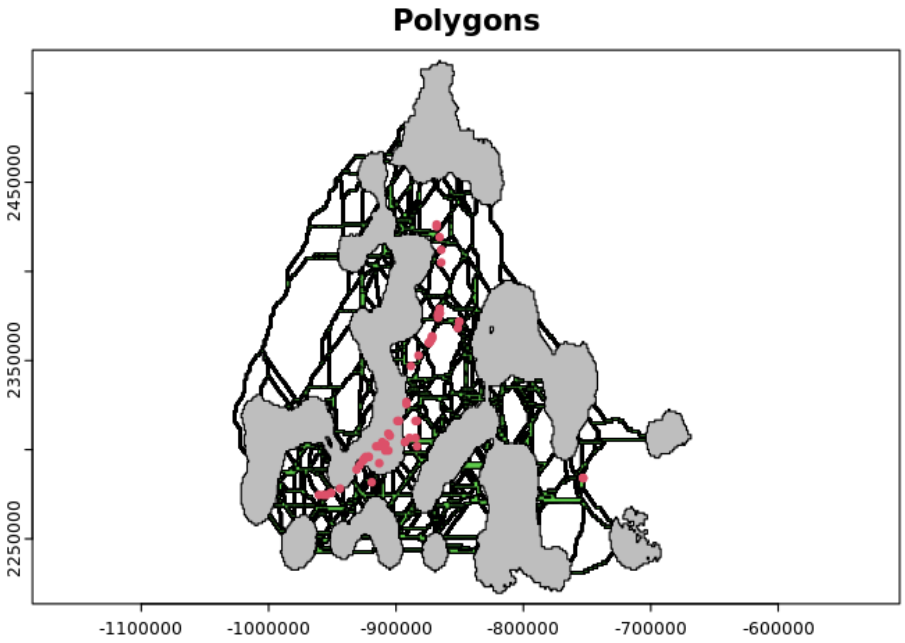
Examples

Considering having the sample data set (REF HERE):

```
library(cola)
library(terra)
wd <- 'cola_examples'
dir.create(wd)
setwd(wd)

hs_path <- system.file(package = 'cola', 'sampledata/sampleTif.tif')
pts_path <- 'pts50.shp'

out_pts <- points_py(intif = hs_path, outshp = pts_path,
                     Smin = 0.20, smax = 0.95, npoints = 50)

out_crk <- crk_py(inshp = pts_path, intif = hs_path, outtif = 'out_crk.tif',
                   maxdist = 20000, shape = 'linear', volume = 10)
out_lcc <- lcc_py(inshp = pts_path, intif = hs_path, outtif = 'out_lcc.tif',
                maxdist = 150000, smooth = 0, tolerance = 0)

## Run Prioritization
out_pri <- pri_py(tif = hs_path, incrk = out_crk$file, inlcc = out_lcc$file,
                  maskedcsname = 'temp_mask.tif',
                  outshppoint = 'out_pri_shp_pts.shp',
                  outshppol = 'out_pri_shp_pol.shp',
                  outshppatch = 'out_pri_shp_patch.shp',
                  outtifpatch = 'out_pri_tif_patch.tif',
                  outtif = 'out_pri_tif_corridors.tif',
```

```
                  threshold = 0.5,
                  maxdist = 150000)

tif_patch <- terra::rast('out_pri_tif_patch.tif')
tif_corri <- terra::rast('out_pri_tif_corridors.tif')
tif_mask <- terra::rast('temp_mask.tif')

plot(tif_corri, main = 'Corridors and patches')
plot(tif_patch, range = c(1, max(tif_patch[])), col = 'darkblue',
     add = TRUE, legend = FALSE)

plot(tif_mask, main = 'Temporal mask')

vec_pts <- terra::vect('out_pri_shp_pts.shp')
vec_pol <- terra::vect('out_pri_shp_pol.shp')
vec_pat <- terra::vect('out_pri_shp_patch.shp')

plot(vec_pat, main = 'Polygons', col = 'grey')
plot(vec_pol, add = TRUE, col = 3)
plot(vec_pts, add = TRUE, col = 2)
```

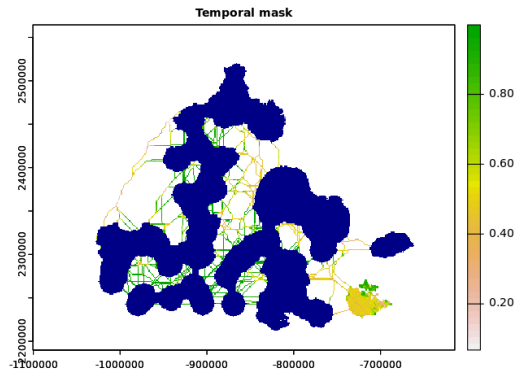


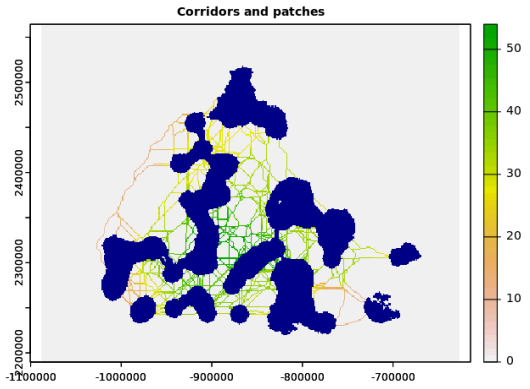


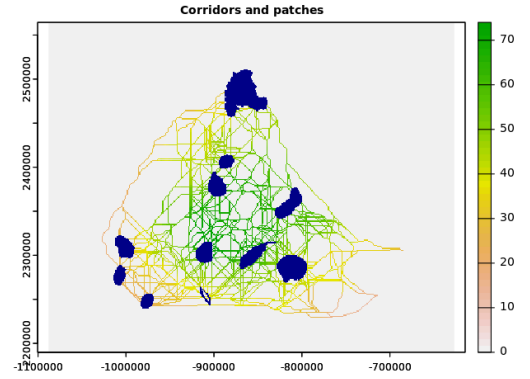
Parameter 7: Threshold quantile

Defines the percentile to use for converting continuous kernels into disrecte patches or nodes. The higher the value, the less percentage of pixels converted into patches, resulting in smaller areas as well.

Parameters	Result

Parameter 7: 0.3
 	
cmd:
`0.3 150000`
 	
R:
`threshold = 0.3, maxdist =150000`
	

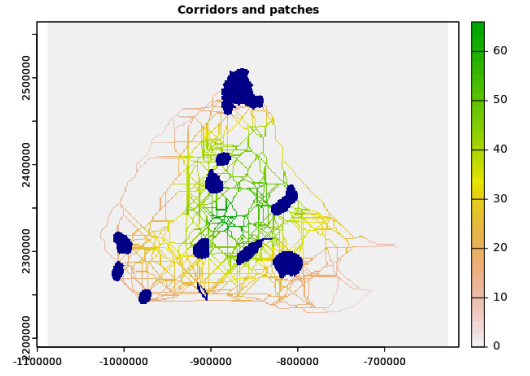
Parameter 7: 0.5
 	
cmd:
`0.5 150000`
 	
R:
`threshold = 0.5, maxdist =150000`
	

Parameter 7: 0.9
 	
cmd:
`0.9 150000`
 	
R:
`threshold = 0.9, maxdist =150000`
	

Parameter 8: Maximum distance

Defines how far can nodes reach others in the landscape. The higher the value, the more likely to reach other nodes.

Parameters	Result

Parameter 8: 100000
 	
cmd:
`0.9 100000`
 	
R:
`threshold = 0.9, maxdist = 100000`
	

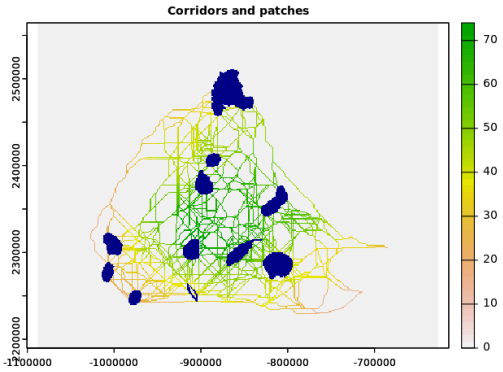
Parameter 8: 150000

cmd:

```
0.9 150000
```

R:

```
threshold = 0.9, maxdist = 150000
```



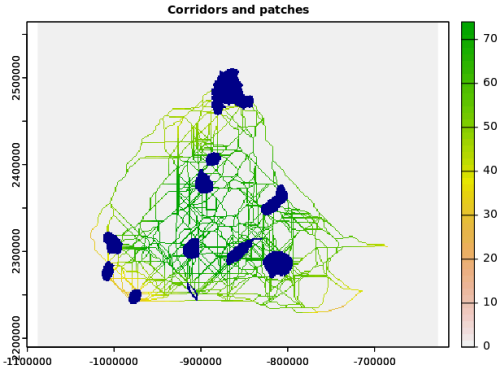
Parameter 8: 200000

cmd:

```
0.9 200000
```

R:

```
threshold = 0.9, maxdist = 200000
```



> Edit rasters (scenarios)

Edits a raster by adding or replacing values on the overlapping pixels given by a features geometry. There are two options: “adding” and “replacing” values. The “add” option (`burnShp()`) sums a given value to the pixels, and is generated by the gdal_rasterize function, with the parameter -add (or add = TRUE in R). “Replace” (`replacePixels()`) assigns a value to the pixels. Both functions rasterize the vectorial layer.

Parameters

- **[1] Feature layer (polPath):** String. File path to the feature layer to be rasterized. Must have the same coordinate system as the input raster [4].
- **[2] Value to burn (burnval):** String or number. Can be the numeric value to use or the name of an attribute (column name) with numeric values on the feature layer.
- **[3] Value [2] is a column name? (colu):** Logical. Is `burnval[2]` the name of a column? Default FALSE
- **[4] Raster layer path (rastPath):** String. File path of the raster layer to be modified.
- **[5] Rasterize all touched pixels? (att):** Logical. File path of the table to be written that contains the relative values of the layers.
- **[6] Use GDAL (only for `replaceRastShp`)?:** Logical. If your system had access to gdal_calc.py then use that script to execute the values replacement. Otherwise it uses `terra` R library.

Output

Returns the path of the edited GeoTIFF raster layer with the prefix “_rasterized.tif” for `burnshape( )` and “_replaced.tif” for `replacePixels( )`.

Usage

In R:

```
library(cola)
library(terra)
wd <- 'cola_examples'
dir.create(wd)
```

```
setwd(wd)

sr_path <- system.file(package = 'cola', 'sampledata/sampleTif.tif')
# sr_path <- 'C:/path/to/tif.tif'
pol_path <- system.file(package = 'cola', 'sampledata/samplePolygon.shp')
# pol_path <- 'C:/path/to/polygon.shp'

addedTif <- burnShp(polPath = pol_path, burnval = 10, colu = FALSE, rastPath =
sr_path, att = FALSE)
plot(rast(addedTif))


replacedTif <- replacePixels(polPath = pol_path, burnval = 10, colu = FALSE,
rastPath = sr_path, gdal = FALSE, att = FALSE)
plot(rast(replacedTif))
```

Common errors

- Using a raster with geographical or no coordinate system
- Polygon with a different coordinate system than the raster
- Having negative values in the resulting raster, if it's going to use for running kernels or corridors

Examples

Considering having the sample data set (REF HERE):

```
library(cola)
library(terra)
wd <- 'cola_examples'
dir.create(wd)
setwd(wd)

sr_path <- system.file(package = 'cola', 'sampledata/sampleTif.tif')
# sr_path <- 'C:/path/to/tif.tif'
pol_path <- system.file(package = 'cola', 'sampledata/samplePolygon.shp')
# pol_path <- 'C:/path/to/polygon.shp'

addedTif <- burnShp(polPath = pol_path, burnval = 10, colu = FALSE, rastPath =
sr_path, att = FALSE)
plot(rast(addedTif))
```

```
replacedTif <- replacePixels(polPath = pol_path, burnval = 10, colu = FALSE,
rastPath = sr_path, gdal = FALSE, att = FALSE)
plot(rast(replacedTif))
```

> Compare scenarios

Compares kernels (crk) or corridors (lcc) baseline layers with other scenarios. You can use them by `crk_compare.py` and `lcc_compare.py` in the command line or `crk_compare_py( )` and `lcc_compare_py( )` in R.

Parameters

- **[1] Baseline raster (intif):** String. File path to the input baseline kernel or corridor raster. Requires a GeoTIFF file with square pixels and cost units as distance units
- **[2] Layers to compare (intifs):** String of raster layer paths to compare. Requires to be passed as a comma separated single string as “layer1.tif, layer2.tif,layer3.tif,...”. Accepts relative or full paths. Pass the baseline raster again [1] as the first raster.
- **[3] Absolute difference out table (outcsvabs):** String. File path of the table to be written that contains the absolute values of the layers. Returns one value for argument [1] plus one of each of the layers passed on the argument [2]
- **[4] Relative difference out table (outcsvrel):** String. File path of the table to be written that contains the relative values of the layers. Returns one value for each of the layers passed on the argument [2] relative to the argument [1]
- **[5] Absolute difference out png (outpngabs):** String. File path of the barplot image to be written that contains the absolute values of the layers. Returns one value for argument [1] plus one of each of the layers passed on the argument [2]
- **[6] Relative difference out png(outpngrel):** String. File path of the barplot image to be written that contains the relative values of the layers. Returns one value for each of the layers passed on the argument [2] relative to the argument [1]
- **[7] Out folder path (outfolder):** String. Data folder path where the comparison rasters will be saved. Writes as many layers as passed in the argument [2]
- **[8] Cutline vectorial layer path (inshp):** String. File path to a vectorial layer used to cutline the comparison. Default value is 'None' for ignoring this parameter. Accept
- **[9] Cutline vectorial layer field (shpfield):** String. Attribute or column name of the vectorial layer provided if you want to regionalize the comparison. By providing a valid column name the csv table will have a second dimension of the length of the unique values of the column. Default value is 'None'.

Output

Creates 5 outputs:

1. CSV table with the absolute values.
2. CSV table with the relative values.
3. PNG barplot with absolute values for all the layers.
4. PNG barplot with relative values for the layers relative to the first layer.
5. GeoTIFF difference rasters layers relative to the first layer.

Usage

In the command line:

```
/path/to/python /path/to/crk_compare.py.py out_crkA.tif
out_crkA.tif,out_crkB.tif,out_crkC.tif out_csv_abs.csv out_csv_rel.csv
out_png_abs.png out_png_rel.png out_folder None None
```

In R:

```
library(cola)
crk_comp <- crk_compare_py(intif = 'out_crkA.tif',
                           intifs = 'out_crkB.tif,out_crkC.tif',
                           outcsvabs = 'out_csv_abs.csv',
                           outcsvrel = 'out_csv_rel.csv',
                           outpngabs = 'out_png_abs.png',
                           outpngrel = 'out_png_rel.png',
                           outfolder = 'out_folder',
                           inshp = 'None', shpfield = 'None')
```

Common errors

- Using a raster with geographical or no coordinate system
- Raster with no squared pixels

Examples

Considering having the sample data set (REF HERE):

```
library(cola)
library(terra)
wd <- 'cola_examples'
dir.create(wd)
setwd(wd)

hs_path <- system.file(package = 'cola', 'sampledata/sampleTif.tif')
pts_path <- 'pts50.shp'

out_pts <- points_py(intif = hs_path, outshp = pts_path,
                     smin = 0.20, smax = 0.95, npoints = 50)

out_crkA <- crk_py(inshp = pts_path, intif = hs_path, outtif = 'out_crkA.tif',
                   maxdist = 10000, shape = 'linear', volume = 1)
out_crkB <- crk_py(inshp = pts_path, intif = hs_path, outtif = 'out_crkB.tif',
                   maxdist = 15000, shape = 'linear', volume = 1)
out_crkC <- crk_py(inshp = pts_path, intif = hs_path, outtif = 'out_crkC.tif',
                   maxdist = 20000, shape = 'linear', volume = 1)

plot(terra::rast('out_crkA.tif'), main = 'Baseline')
plot(terra::rast('out_crkB.tif'), main = 'Scen. B')
plot(terra::rast('out_crkC.tif'), main = 'Scen. C')

crk_comp <- crk_compare_py(intif = 'out_crkA.tif',
                           intifs = 'out_crkA.tif,out_crkB.tif,out_crkC.tif',
                           outcsvabs = 'out_csv_abs.csv',
                           outcsvrel = 'out_csv_rel.csv',
                           outpngabs = 'out_png_abs.png',
                           outpngrel = 'out_png_rel.png',
                           outfolder = 'out_folder',
                           inshp = 'None', shpfield = 'None')

plot(terra::rast('out_folder/s2_crk_comp.tif'),
     main = 'Comp. baseline (A) and Scen. 1 (B)')
plot(terra::rast('out_folder/s1_crk_comp.tif'),
     main = 'Comp. baseline (A) and Scen. 2 (C)')
```

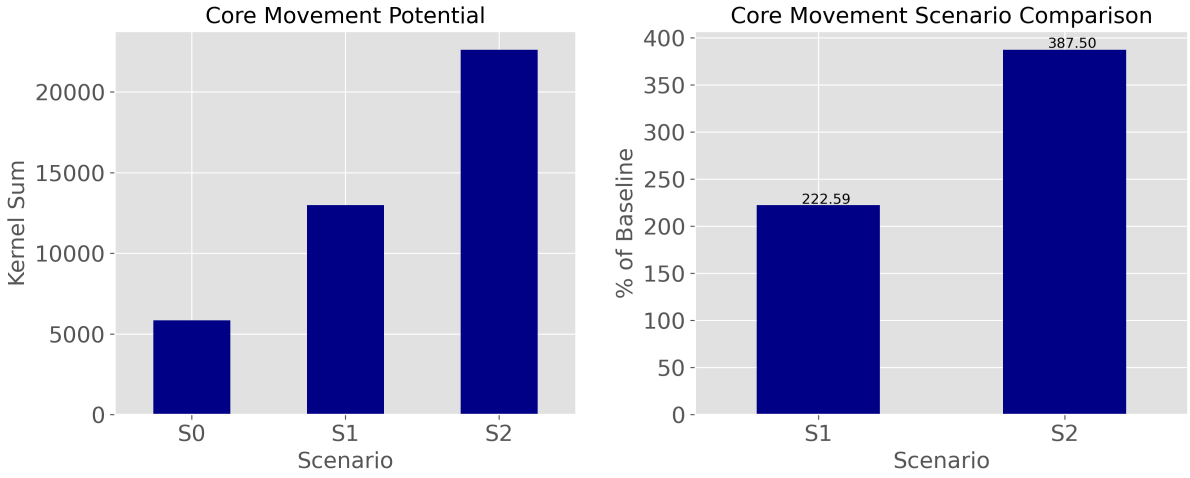


Figure 19. Barplot outputs. Left: Absolute values. Right: Relative values against the baseline

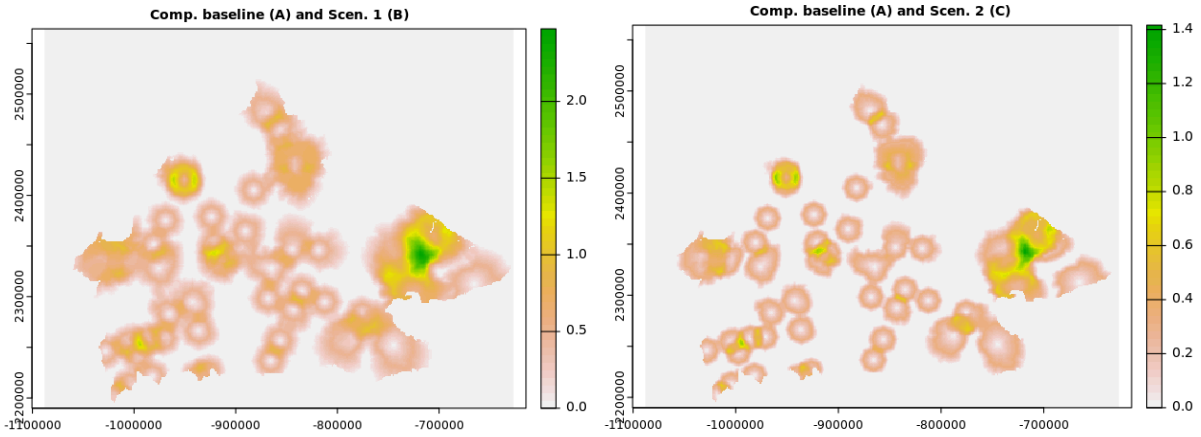


Figure 19. GeoTIFF outputs. Left: Difference from scenarios 0 and 1. Right: Difference from scenarios 0 and 1

Decision support system (DSS), dashboard, or front-end

The decision support system (DSS) is the third module of the software. The DSS, also called dashboard or front-end, intends to provide a graphical user interface that allows users to run the functions and visualize the results without interacting with the command line.

All the functions presented in the previous title are incorporated into the DSS. There are also some exclusive functionalities available in the DSS, meant for non-coder users.

Launch the DSS by typing in R:

```
library(cola)
cola::cola_dss()
```

The next figure shows in purple the vertical disposition of the different tools, inside the purple box. That lateral bar contains all the available modules we developed. Each of them can work as independent analysis, but also you can nest your products.

All the tabs provide the widgets required to upload the input files and customize your analysis through parameters. Also each module had a “download” button.

The DSS can allocate the following type of variables under the tabs:

- HS: Raster of habitat suitability
    - Input tabs: Habitat suitability <> resistance surface

- SR: Raster of surface resistance (input and output)
    - Input tabs: Customize resistance surface, Create source points, Cost distance matrix, Connectivity - kernels, Connectivity - corridors
    - Output tab: Habitat suitability <> resistance surface
- Points of individuals locations (input and output)
    - Input tabs: Cost distance matrix, Connectivity - kernels, Connectivity - corridors
    - Output tab: Create source points
- Polygon for editing the SR (input)
    - Input tabs: Customize resistance surface
- 2-way interaction matrix table (output)
    - Output tab: Cost distance matrix
- LCC: Raster of corridors (output):
    - Output tab: Connectivity - corridors
- CRK: Raster of cumulative resistance kernels (output)
    - Output tab: Connectivity - kernels
- Corridor prioritization: Prioritized Corridors shapefiles and rasters
    - Output tab: Connectivity - prioritization
- Comparisons (LCC or CRK): Set of difference rasters relative to a baseline
    - Output tab: Compare results
-

Most of the tools tabs have the following components:

- Purple: Toolbar
- Green: Log box. Report your session ID. Save this code for debugging. It also shows important messages in your session. Also there's a "load sample HS" button. Only for the "Habitat suitability to Surface resistance" tab.
- Yellow: Input widgets where user can upload the laters
- Orange: Parameters section when users can customize the analysis
- Gray: Input-readiness label, indicating if inputs and outputs (only for "Cost distance matrix" tab)
- Magenta / Pink: "Run" and "Download results" button.
- Blue: Geographical canvas where spatial results are shown. Users can also click on the map and inspect the raster values.
- Red: Map widgets and utilities, including zoom buttons, legend, layer and basemap manager, distance+area measurement tool, minimap, and drawing tools (only available for Editing scenarios tab).

Figure 20. General DSS modules. Each color

Keep this color code in mind so the next sections will refer to them when explaining the tool usage.

When the raster layers are uploaded to the DSS the back-end might not display the original dataset. The DSS generates two independent procedures in order to verify the raster quality and optimize the system performance.

The two process that occurring at the DSS are:

- Verify and fix raster conditions. The kernels and corridors algorithms require squared pixels and a specific NODATA value. CoLa resample your raster to the higher pixel side size (width or height) to ensure squared pixels. In that sense, the backend will produce a new “[ORIGINAL_FILE]_fixed.tif” layer with squared pixels and -9999 as NODATA
- Resample for visualization. Since the raster display might require several resources from your machine that we want to save for computing capacity, CoLa assesses if the raster contains more than 1000000 pixels (default configuration value). If so, the raster will be resampled to 1000 x 1000 pixels to be displayed on the canvas. The original (maybe fixed) file will remain in the back end and will be used for all the analysis. This resampling action will also apply to any raster produced in the back-end. The original “high-res” raster can be downloaded at any time. The value of 1000000 can be modified by each user. The resampled raster layer is named “[ORIGINAL_FILE]_resampled.tif”.

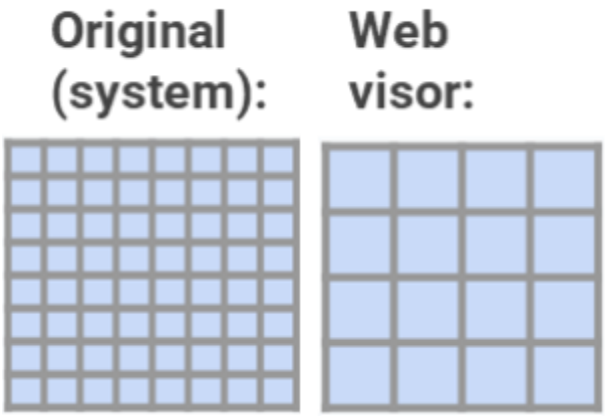


Figure 21. Raster resampling schema used for visualize the layers in the DSS

> Habitat suitability <> resistance surface

Creates a surface resistance (SR) from habitat suitability (HS) raster as explained in the back-end title.

Usage

For using the tool on the DSS, follow these steps:

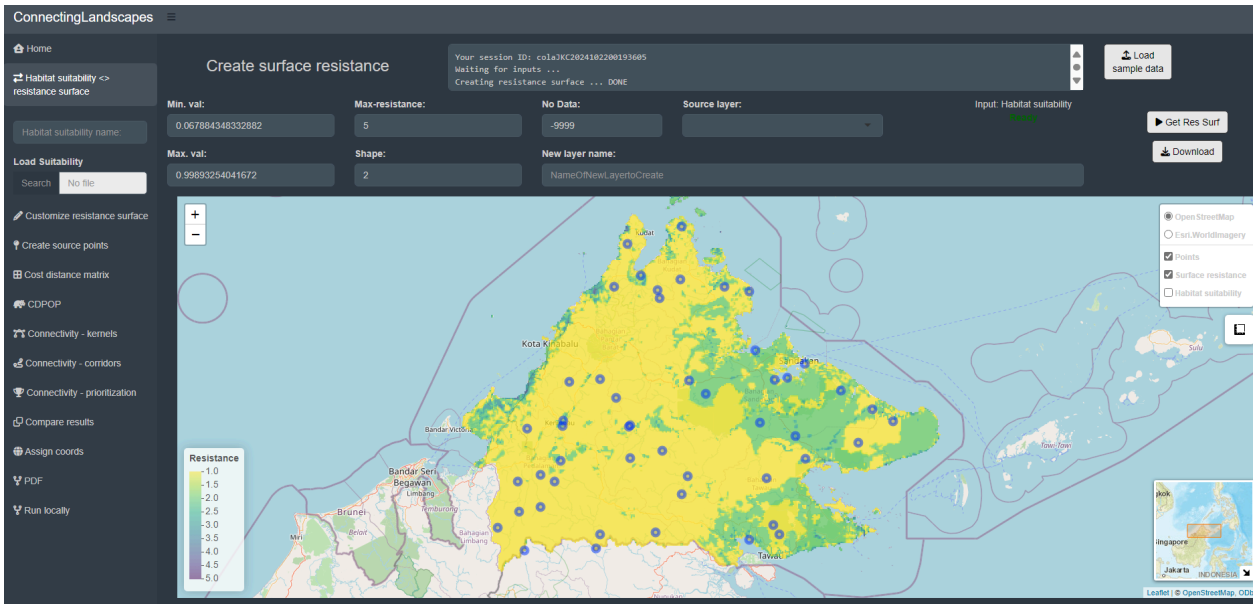
- Upload the SR on the yellow box, or load the sample dataset. Be sure your raster has a valid NODATA value and has a projected coordinates reference system.
- Internally the DSS will convert any valid NODATA value to -9999
- If the raster is a valid file, the map will be displayed. The input layer has a color ramp from yellow (low) to blue (high).
- Wait for a message in the (green) log box and an “Input: Ready” message (gray).
- Specify the parameters, avoiding using zero as “shape”.
- Hit the “run” button. The map will be updated. Some layers will be hidden.
- Download the resulting TIF layer.

Output

Creates a new surface resistance layer

Common errors

- Using 0 as shape parameter
- Using a raster without a NODATA value
- Using a raster with non or a geographical reference coordinates system


ConnectingLandscapes
Home
Habitat suitability <> resistance surface
Habitat suitability name:
Load Suitability
Search
No file
Customize resistance surface
Create source points
Cost distance matrix
CDPOP
Connectivity - kernels
Connectivity - corridors
Connectivity - prioritization
Compare results
Assign coords
PDF
Run locally
Create surface resistance
Your session ID: colaJKC202410220019360S
Waiting for inputs ...
Creating resistance surface ... DONE
Load sample data
Min. val:
0.067884348332882
Max-resistance:
5
No Data:
-9999
Source layer:
Input: Habitat suitability
Get Res Surf
Download
Max. val:
0.998932540416 72
Shape:
2
New layer name:
NameOfNewLayertoCreate
Resistance
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
Kota Kinabalu
Sandakan
Bandar Seri Begawan
Brunei
Tawau
Jakarta
INDONESIA
OpenStreetMap
Esri.WorldImagery
Points
Surface resistance
Habitat suitability
Leaflet | © OpenStreetMap, ODb

> Customize resistance surface

Digitalize and rasterize features drawn or uploaded on the DSS. This utility is only enabled on the front end. On the canvas, the user can download different feature types and use that coverage to be added to replace the underlying raster values.

Parameters

- **[1] Features:** Set of polygons drawn on the canvas or uploaded on the input widget. Uploaded features must have the same (projected) coordinate system as the underlying surface resistance.
- **[2] Value to add or replace:** Numeric. Value to add or replace to the surface resistance. Zero is not allowed for the “Add value” option.
- **[3] Pixel width:** The number of pixels to use as buffer width to the lines geometries draw, if any. A value of 1 will return a line of 2 pixels in width. Only applies for drawn geometries. Uploaded geometries will not be buffered.
- **[4] All touched pixels checkbox:** Logical. Passed internally as the `-at` argument to `gdal rasterize` function. Checking this box is recommended for line geometries especially to avoid situations where features are converted to rasters with only the corners of pixels touching, allowing least cost pathways to “jump” over the feature.
- **[5] Action button:** Hit on “Add value” or “Replace values”.
- **[6] Download button:** Hit on the button to download the updated raster file

Usage

For using the tool on the DSS, follow these steps:

- Upload the SR on the yellow box, if you haven't uploaded a SR before. Be sure your raster has a valid NODATA value and has a projected coordinates reference system.
- Wait for a message in the (green) log box and an “Input: Ready” message (gray).
- If the raster is a valid file, the map will be displayed. The input layer has a color ramp from yellow (low) to blue (high).
- Upload the geometry layer. If the files are valid, the features will be displayed on the map.
- Use the draw tool (red box) to create several features. Multiple geometries allowed. Be sure to close or terminate each feature by double clicking at the end of each part.
- If both updated polygon and drawn geometries are valid, only the first features source will be used.

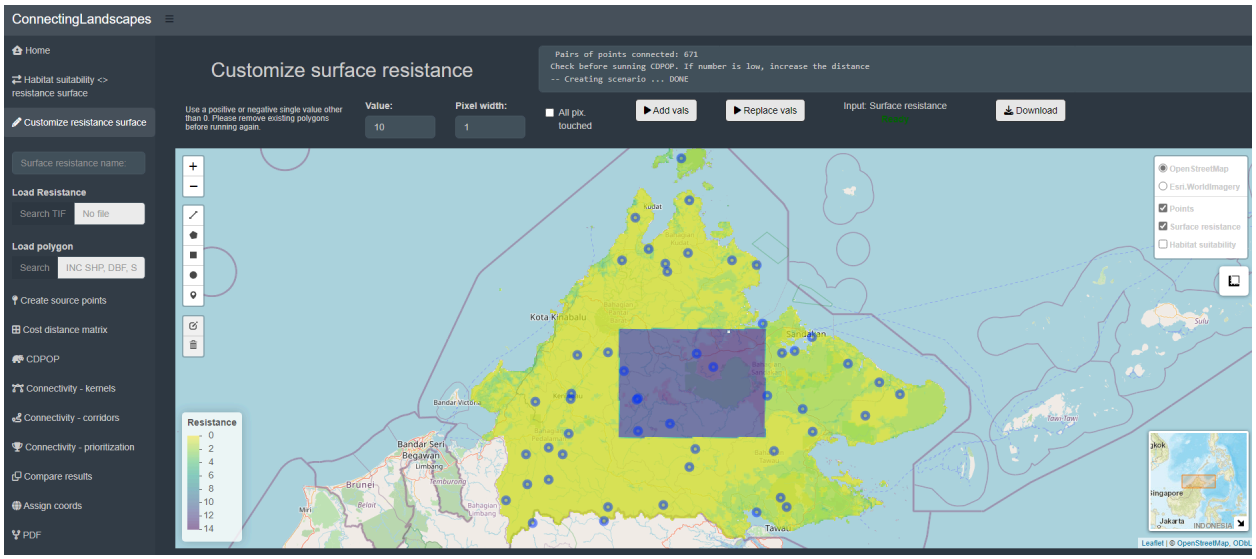
- Specify the parameters, avoiding using zero as “shape”.
- Hit the “run” button. The map will be updated. Some layers will be hidden.
- Download the resulting TIF layer.

Output

Creates a new surface resistance raster layer with the replaced or added values. The new raster will be displayed on the front end. The new layer can be downloaded from the DSS using the Download button.

Common errors

- Using a raster with geographical or no coordinate system
- Raster with no squared pixels



> Cost distance matrix

Creates a two-way interaction CSV matrix as explained in the back-end title.

Usage

For using the tool on the DSS, follow these steps:

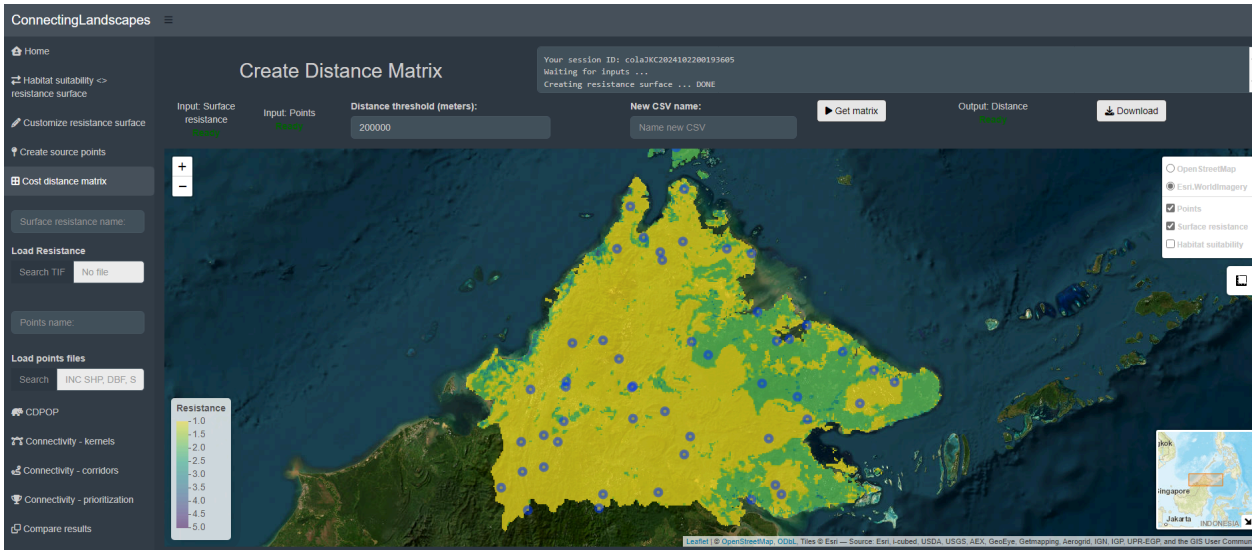
- First upload the SR on the yellow box, in this or other tool tabs. Be sure your raster has a valid NODATA value and has a projected coordinates reference system.
- If the raster is a valid file, the map will be displayed. The input layer has a color ramp from yellow (low) to blue (high).
- Then upload the point layer. Be sure to include all files if the layer is an ESRI Shapefile format. The layers are required to have the same projected coordinate system.
- If the layer is valid the points will be displayed in blue.
- Wait for a message in the (green) log box and a green text “Input: Ready” message (gray box) after uploading valid inputs.
- Specify the cost-distance parameter, avoiding using negative values.
- Hit the “run” button. The map will be updated. Some layers will be hidden.
- Download the resulting CSV matrix.

Output

Creates a new 2-way cost-distance CSV matrix. This output will not be displayed, but can be downloaded. The “Output” label will turn color green with the text “Ready”

Common errors

- Using negative parameters
- Using a raster without a NODATA value
- Using a raster with non or a geographical reference coordinates system
- Using a points layer with a different coordinate system
- Trying to upload the point layer before uploading the SR layer


ConnectingLandscapes
Home
Habitat suitability <> resistance surface
Customize resistance surface
Create source points
Cost distance matrix
Surface resistance name:
Load Resistance
Search TIF
No file
Points name:
Load points files
Search
INC SHP, DBF, S
CDPOP
Connectivity - kernels
Connectivity - corridors
Connectivity - prioritization
Compare results
Create Distance Matrix
Your session ID: colaJKC202410220193605
Waiting for inputs ...
Creating resistance surface ... DONE
Input: Surface resistance
Ready
Input: Points
Ready
Distance threshold (meters):
200000
New CSV name:
Name new CSV
Get matrix
Output: Distance
Ready
Download
OpenStreetMap
Esri.WorldImagery
Points
Surface resistance
Habitat suitability
Resistance
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
Singapore
Jakarta
INDONESIA
Leaflet | © OpenStreetMap, ODbL, Tiles © Esri — Source: Esri, i-cubed, USDA, USGS, AEX, GeoEye, Getmapping, Aerogrid, IGN, IGP, UPR-EGP, and the GIS User Commun

> CDPOP

Runs CDPOP simulation with the inputs available in the DSS. For further details check the CDPOP webpage.

Parameters

- **[1] Features:** Layer of points simulated inside the DSS or uploaded. Converted internally to match CDPOP coordinates file format.
- **[2] 2-way matrix:** Matrix generated inside the DSS.
- **[3] Mortality from resistance?:** Populate the mortality column of the coordinates files with the extracted values from the SR raster layer. If not checked, mortality is zero for all points. Resistance will be rescaled to range from 0-100.

Usage

For using the tool on the DSS, follow these steps:

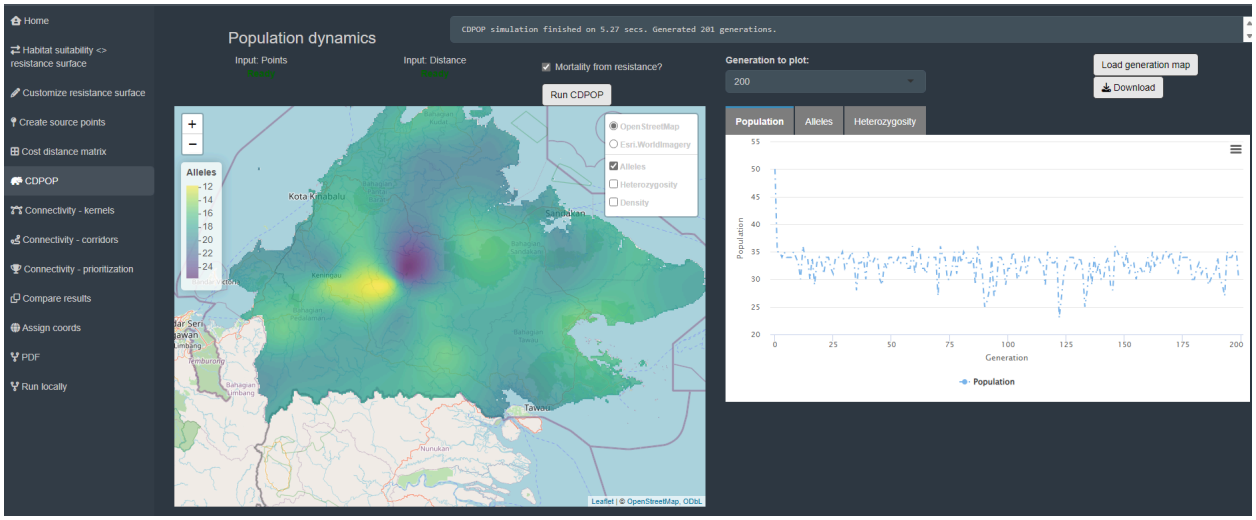
- Having a valid 2-way CSV matrix. For this, a valid SR raster and points layer.
- Decide to check or not the “use mortality from resistance” box.
- Hit the “run” button.
- A time series will appear with some genetic metrics
- The map viewer of the interpolated genetic metrics will display the genetic map of last available year of the simulation
- Download the resulting text and spatial files on a ZIP file.

Output

Creates a set of files from CDPOP simulation. There will be as many files as years of the simulation with at least one surviving individual. Also a spatial interpolation of the last available year. Files are downloaded on a zip file.

Common errors

- Using a small distance parameter when creating the 2-way matrix that returns a population who dies at the first simulation


Home
Habitat suitability <> resistance surface
Customize resistance surface
Create source points
Cost distance matrix
CDPOP
Connectivity - kernels
Connectivity - corridors
Connectivity - prioritization
Compare results
Assign coords
PDF
Run locally
Population dynamics
CDPOP simulation finished on 5.27 secs. Generated 201 generations.
Input: Points
Input: Distance
Mortality from resistance?
Run CDPOP
Generation to plot:
200
Load generation map
Download
Population
Alleles
Heterozygosity
Alleles
Kota Kinabalu
Sandakan
Tawau
Nunukan
OpenStreetMap
Esri.WorldImagery
Alleles
Heterozygosity
Density
Leaflet | © OpenStreetMap, ODbL
Population
Generation
Population

> Connectivity - kernels

Creates a cumulative kernels raster TIF as explained in the back-end title.

Usage

For using the tool on the DSS, follow these steps:

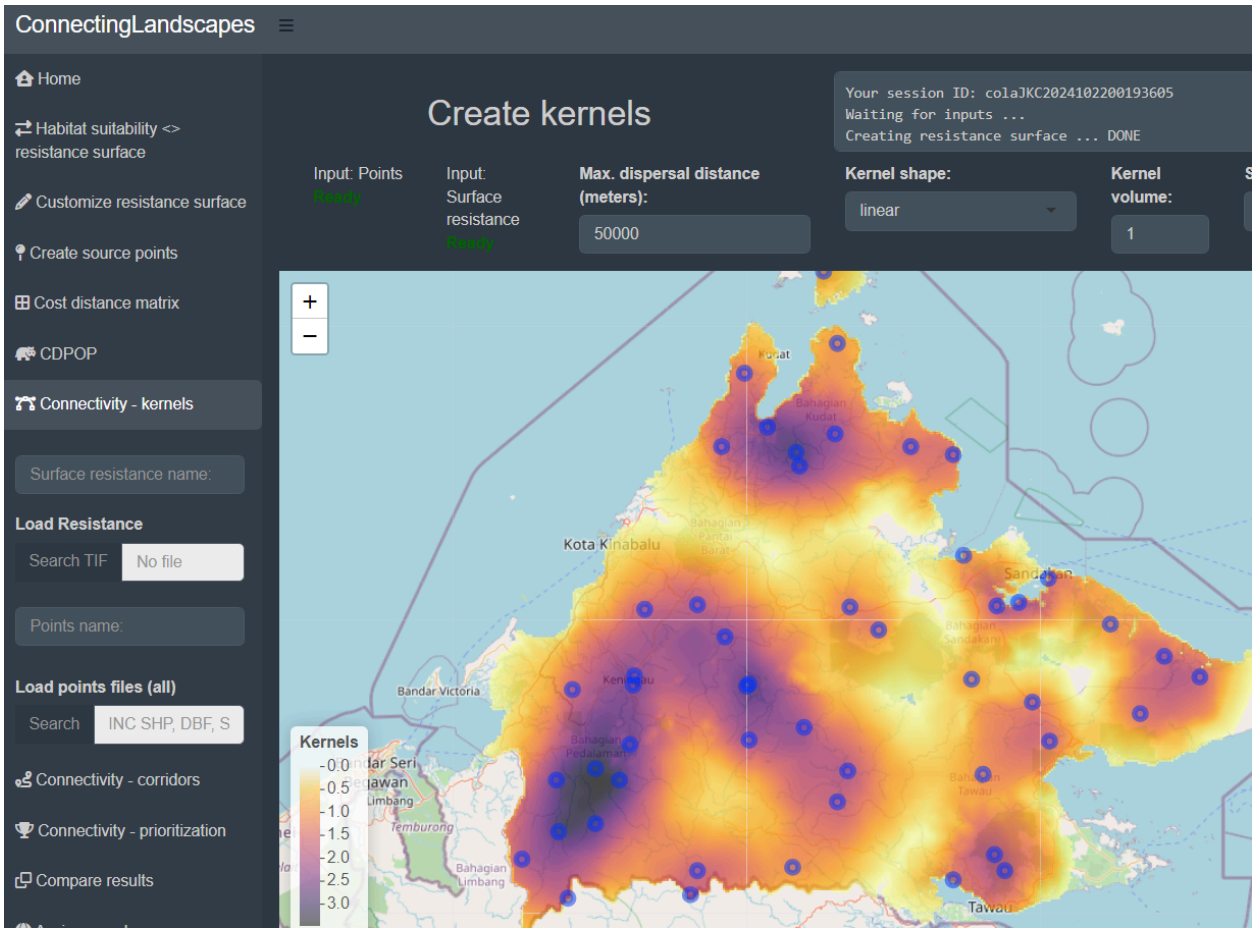
- First upload the SR on the yellow box, in this or other tool tabs. Be sure your raster has a valid NODATA value and has a projected coordinates reference system.
- If the raster is a valid file, the map will be displayed. The input layer has a color ramp from yellow (low) to blue (high).
- Then upload the point layer. Be sure to include all files if the layer is an ESRI Shapefile format. The layers are required to have the same projected coordinate system.
- If the layer is valid the points will be displayed in blue.
- Wait for a message in the (green) log box and a green text “Input: Ready” message (gray box) after uploading valid inputs.
- Specify the cost-distance, kernel volume and shape parameter, avoiding using negative values.
- Hit the “run” button. The map will be updated. Some layers will be hidden.
- Download the resulting TIF raster.

Output

Creates a kernels TIF raster layer. This output will be displayed, and can be downloaded.

Common errors

- Using negative parameters
- Using a raster without a NODATA value
- Using a raster with non or a geographical reference coordinates system
- Using a points layer with a different coordinate system
- Trying to upload the point layer before uploading the SR layer


ConnectingLandscapes
Home
Habitat suitability <> resistance surface
Customize resistance surface
Create source points
Cost distance matrix
CDPOP
Connectivity - kernels
Surface resistance name:
Load Resistance
Search TIF
No file
Points name:
Load points files (all)
Search
INC SHP, DBF, S
Connectivity - corridors
Connectivity - prioritization
Compare results
Create kernels
Your session ID: colaJKC2024102200193605
Waiting for inputs ...
Creating resistance surface ... DONE
Input: Points
Ready
Input: Surface resistance
Ready
Max. dispersal distance (meters):
50000
Kernel shape:
linear
Kernel volume:
1
Kota Kinabalu
Bandar Victoria
Kudat
Sandakan
Tawau
Kernels
0.0
0.5
1.0
1.5
2.0
2.5
3.0

> Connectivity - corridors

Creates a cumulative kernels raster TIF as explained in the back-end title.

Usage

For using the tool on the DSS, follow these steps:

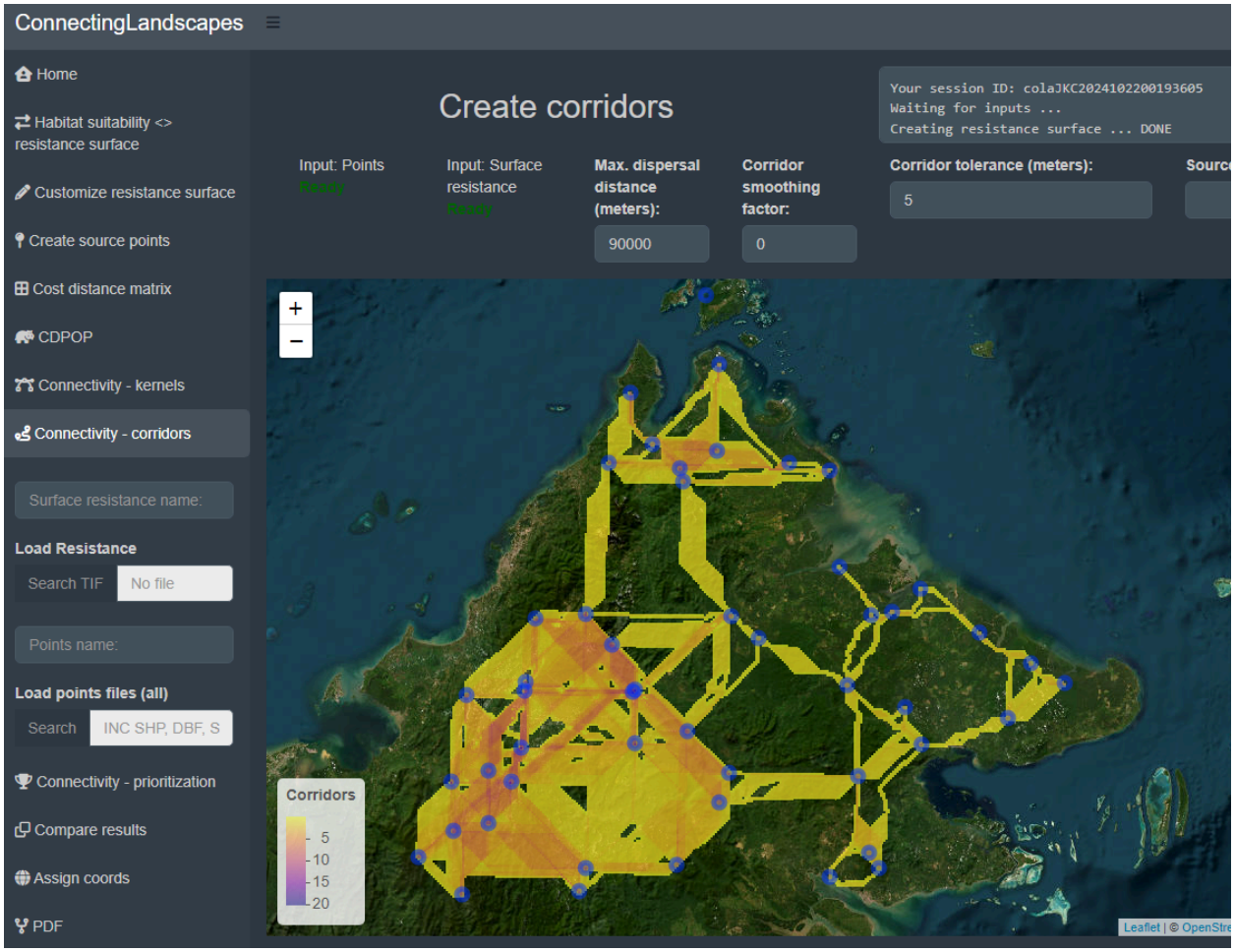
- First upload the SR on the yellow box, in this or other tool tabs. Be sure your raster has a valid NODATA value and has a projected coordinates reference system.
- If the raster is a valid file, the map will be displayed. The input layer has a color ramp from yellow (low) to blue (high).
- Then upload the point layer. Be sure to include all files if the layer is an ESRI Shapefile format. The layers are required to have the same projected coordinate system.
- If the layer is valid the points will be displayed in blue.
- Wait for a message in the (green) log box and a green text “Input: Ready” message (gray box) after uploading valid inputs.
- Specify the cost-distance, smooth factor, and corridor tolerance parameter, avoiding using negative values.
- Hit the “run” button. The map will be updated. Some layers will be hidden.
- Download the resulting TIF raster.

Output

Creates a corridor TIF raster layer. This output will be displayed, and can be downloaded.

Common errors

- Using negative parameters
- Using a raster without a NODATA value
- Using a raster with non or a geographical reference coordinates system
- Using a points layer with a different coordinate system
- Trying to upload the point layer before uploading the SR layer


ConnectingLandscapes
Home
Habitat suitability <> resistance surface
Customize resistance surface
Create source points
Cost distance matrix
CDPOP
Connectivity - kernels
Connectivity - corridors
Surface resistance name:
Load Resistance
Search TIF
No file
Points name:
Load points files (all)
Search
INC SHP, DBF, S
Connectivity - prioritization
Compare results
Assign coords
PDF
Create corridors
Your session ID: colaJKC2024102200193605
Waiting for inputs ...
Creating resistance surface ... DONE
Input: Points
Ready
Input: Surface resistance
Ready
Max. dispersal distance (meters):
90000
Corridor smoothing factor:
0
Corridor tolerance (meters):
5
Corridors
5
10
15
20
Leaflet | © OpenStre

> Connectivity - prioritization

Creates a cumulative kernels raster TIF as explained in the back-end title.

Usage

For using the tool on the DSS, follow these steps:

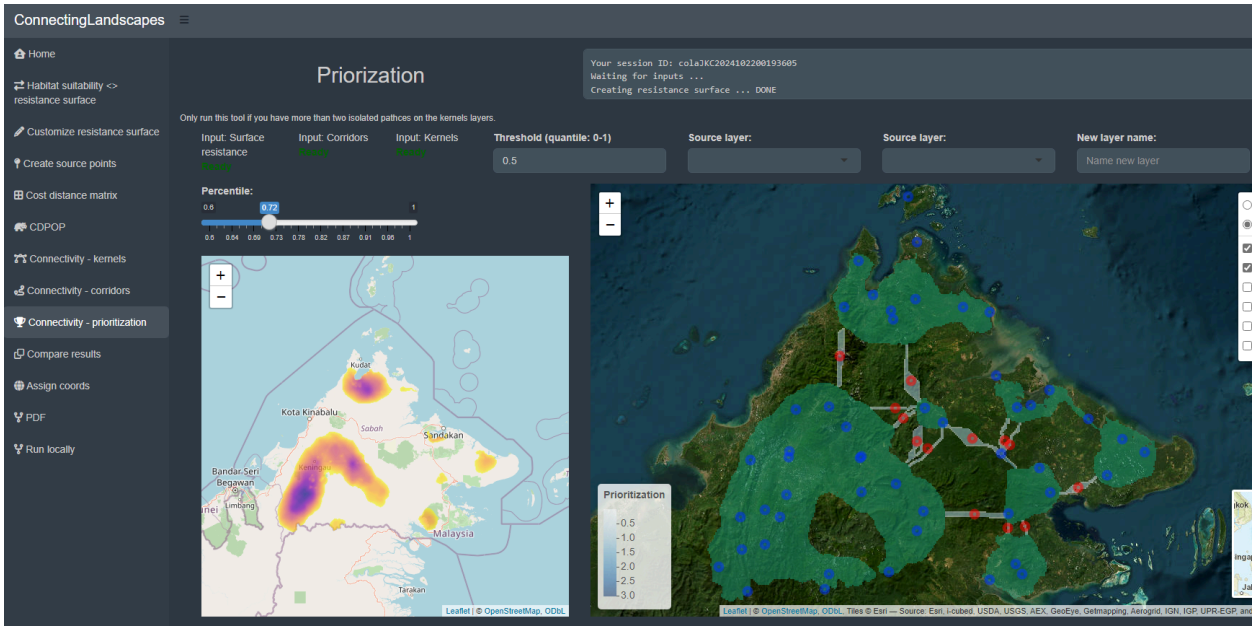
- It is required to have a valid SR, kernels and corridors layers active in the DSS.
- Use the slider on the left to determine the percentile that cuts the continuous kernels maps into discrete patches. Then define the threshold parameter.
- Be sure to use a threshold that generates more than one patch. The function requires more than one patch.
- Hit the “run” button. The map will be updated. Some layers will be hidden.
- Download the resulting TIF raster and SHP layers in a ZIP file.

Output

Creates a prioritized corridors TIF, patches TIF, and vectorial prioritized corridor layer, and corridors centroids. This output will be displayed, and can be downloaded.

Common errors

- Using negative parameters
- Use a threshold that creates a single patch from the kernels layer


ConnectingLandscapes
Home
Habitat suitability <> resistance surface
Customize resistance surface
Create source points
Cost distance matrix
CDPOP
Connectivity - kernels
Connectivity - corridors
Connectivity - prioritization
Compare results
Assign coords
PDF
Run locally
Priorization
Your session ID: colaJKC2024102200193605
Waiting for inputs ...
Creating resistance surface ... DONE
Only run this tool if you have more than two isolated pathces on the kernels layers.
Input: Surface resistance
Input: Corridors
Input: Kernels
Threshold (quantile: 0-1)
0.5
Source layer:
Source layer:
New layer name:
Name new layer
Percentile:
Kudat
Kota Kinabalu
Sabah
Sandakan
Bandar Seri Begawan
Malaysia
Tarakan
Leaflet | © OpenStreetMap, ODbL
Prioritization
-0.5
-1.0
-1.5
-2.0
-2.5
-3.0
Leaflet | © OpenStreetMap, ODbL, Tiles © Esri — Source: Esri, i-cubed, USDA, USGS, AEX, GeoEye, Getmapping, Aerogrid, IGN, IGP, UPR-EGP, and

> Compare results

Compare the first available raster layer of the kernels or corridors in the DSS against the other layers of the same kind. The comparison is made only among corridors or kernels values.

Usage

For using the tool on the DSS, follow these steps:

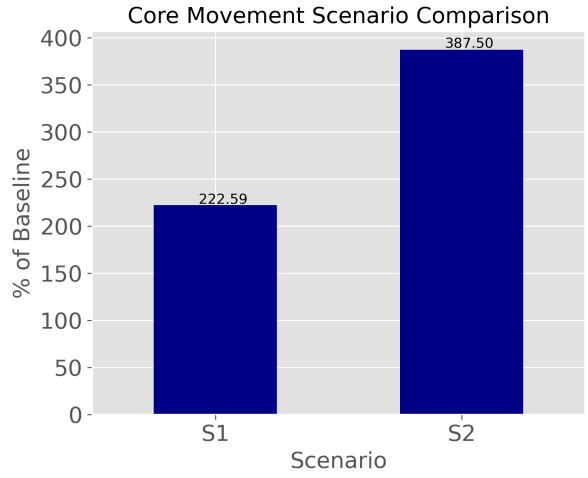
- Having at least two available layers of the type you want to compare.
- Select on the top box the type of layer to compare. Be sure to have produced at least 2 of them in the DSS session. The log box will indicate how many layers are available.
- Hit the “Compare” button.
- The compared layers will be displayed. Will show all the possible differences between the first available layer against the remaining one.
- Two bar plots will be show the absolute and relative (difference against the reference) values of the available layers
- Download the resulting spatial difference TIFF rasters, a summary CSV table and PNG plots.

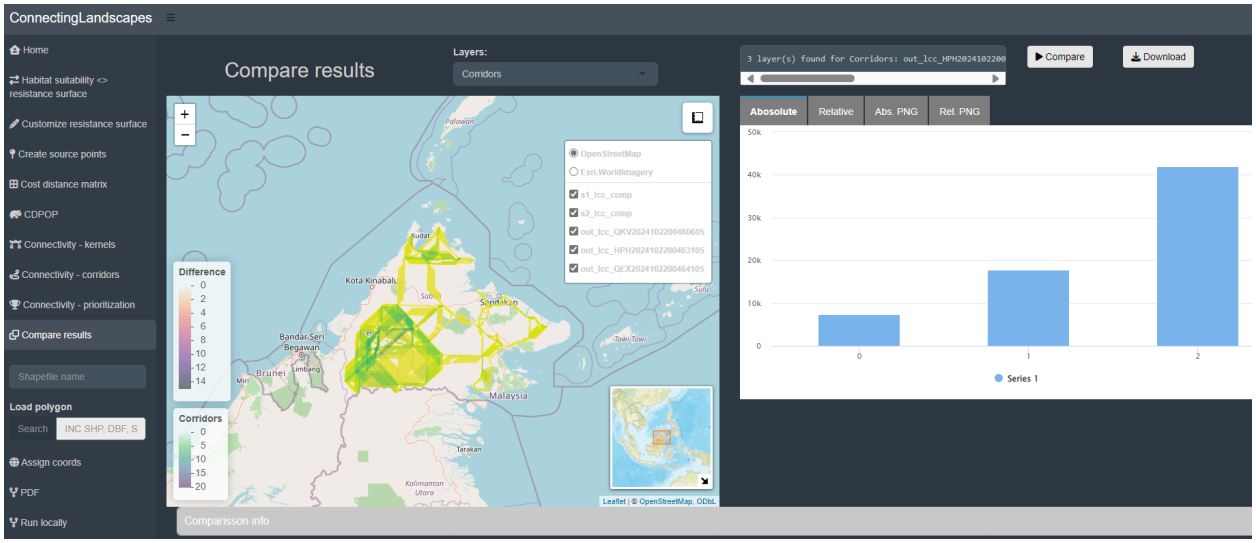
Output

Creates a set of spatial TIF raster difference layers. Also a CSV table that summarizes the spatial layers, and PNG boxplots made with the CSV values. Files are downloaded on a zip file.

Common errors

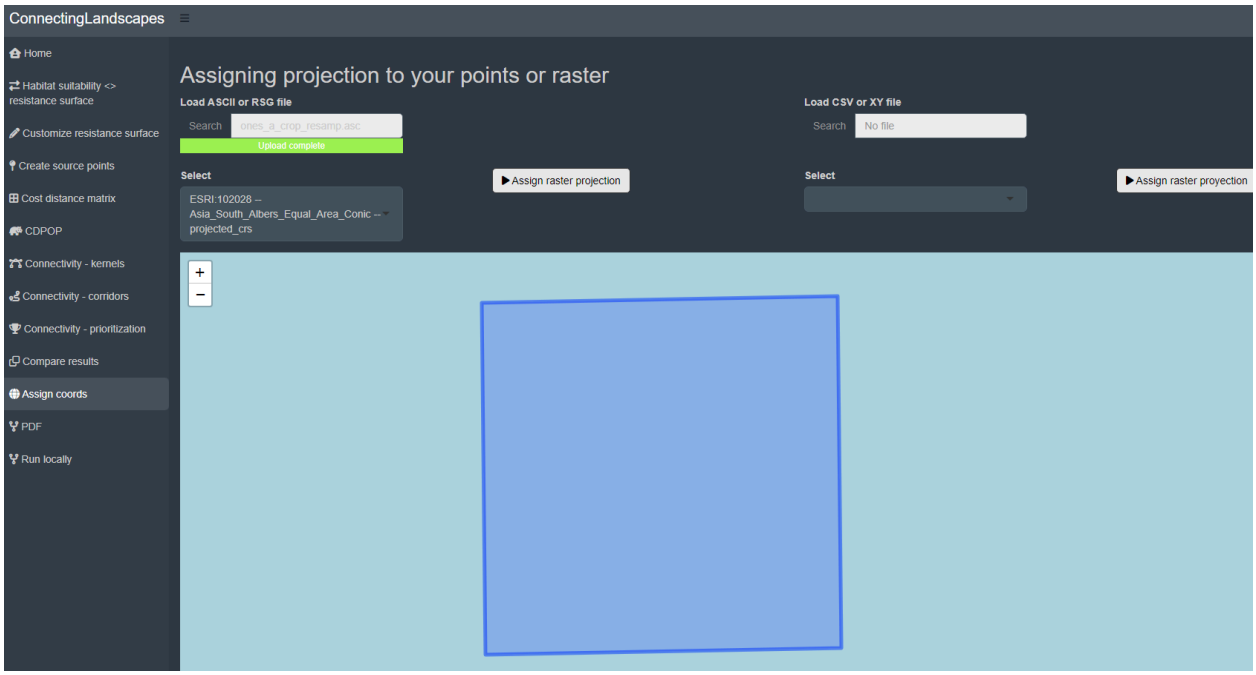
- Only having one available layer of corridors or kernels


Core Movement Scenario Comparison
400
350
300
250
200
150
100
50
0
% of Baseline
387.50
222.59
S1
S2
Scenario


ConnectingLandscapes
Home
Habitat suitability <> resistance surface
Customize resistance surface
Create source points
Cost distance matrix
CDPOP
Connectivity - kernels
Connectivity - corridors
Connectivity - prioritization
Compare results
Shapefile name
Load polygon
Search
INC SHP, DBF, S
Assign coords
PDF
Run locally
Compare results
Layers:
Corridors
Difference
Corridors
Comparisson info
3 layer(s) found for Corridors: out_lcc_HPH2024102200
Compare
Download
Abosolute
Relative
Abs. PNG
Rel. PNG
Series 1

> Assign coords

This tool allows you to define the coordinate reference system of your non-georeferenced files. For raster layers you can include .RSG or .ASC files. For the points layer you can use .XY or .CSV files. The original rasters are not displayed, but the extent square derived from the layer limits.



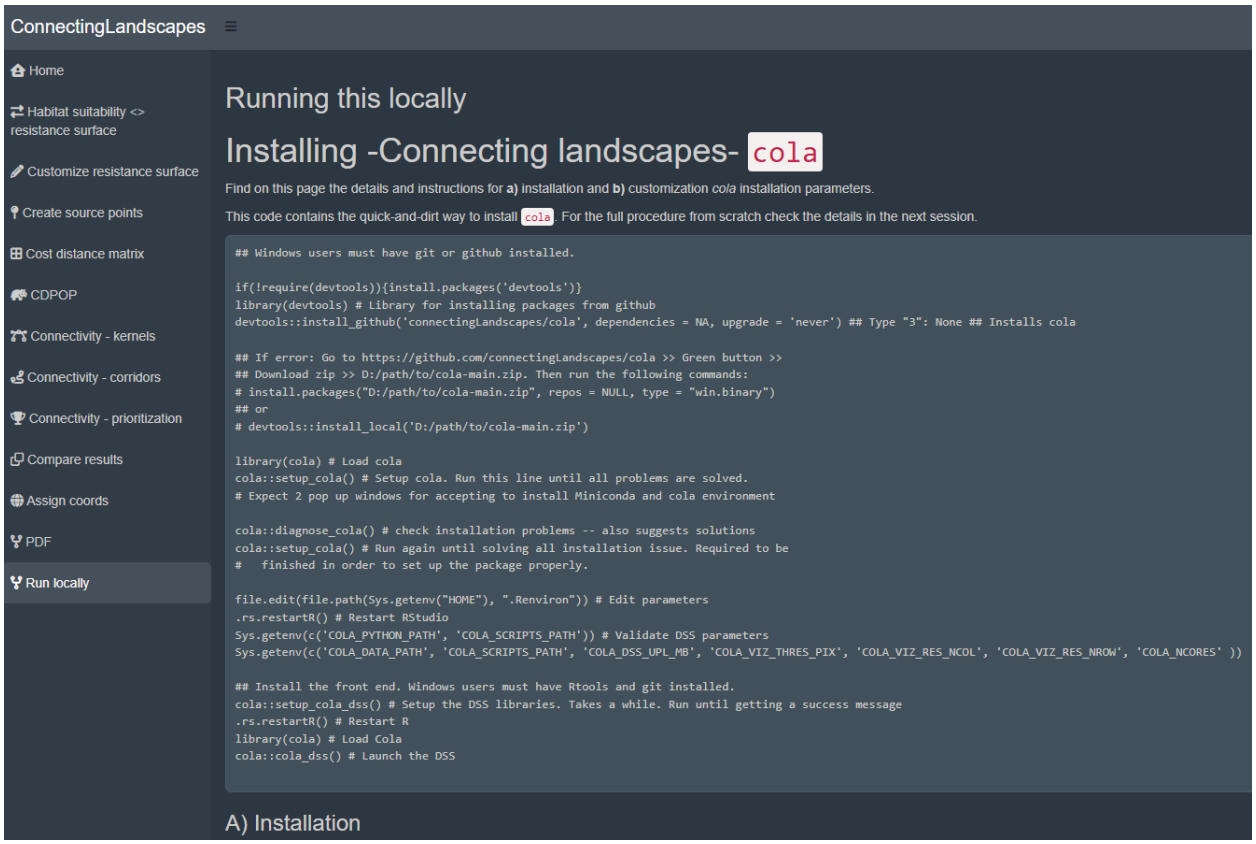
> PDF

Graphical DSS tutorial using the sample raster


ConnectingLandscapes
Home
Habitat suitability <> resistance surface
Customize resistance surface
Create source points
Cost distance matrix
CDPOP
Connectivity - kernels
Connectivity - corridors
Connectivity - prioritization
Compare results
Assign coords
PDF
Run locally
Running this locally
Installing -Connecting landscapes- cola
Find on this page the details and instructions for a) installation and b) customization cola installation parameters.
This code contains the quick-and-dirt way to install cola. For the full procedure from scratch check the details in the next session.
Windows users must have git or github installed.
if(!require(devtools)){install.packages('devtools')}
library(devtools) # Library for installing packages from github
devtools::install_github('connectingLandscapes/cola', dependencies = NA, upgrade = 'never') ## Type "3": None ## Installs cola
If error: Go to https://github.com/connectingLandscapes/cola >> Green button >>
Download zip >> D:/path/to/cola-main.zip. Then run the following commands:
install.packages("D:/path/to/cola-main.zip", repos = NULL, type = "win.binary")
or
devtools::install_local('D:/path/to/cola-main.zip')
library(cola) # Load cola
cola::setup_cola() # Setup cola. Run this line until all problems are solved.
Expect 2 pop up windows for accepting to install Miniconda and cola environment
cola::diagnose_cola() # check installation problems -- also suggests solutions
cola::setup_cola() # Run again until solving all installation issue. Required to be
finished in order to set up the package properly.
file.edit(file.path(Sys.getenv("HOME"), ".Renviron")) # Edit parameters
.rs.restartR() # Restart RStudio
Sys.getenv(c('COLA_PYTHON_PATH', 'COLA_SCRIPTS_PATH')) # Validate DSS parameters
Sys.getenv(c('COLA_DATA_PATH', 'COLA_SCRIPTS_PATH', 'COLA_DSS_UPL_MB', 'COLA_VIZ_THRES_PIX', 'COLA_VIZ_RES_NCOL', 'COLA_VIZ_RES_NROW', 'COLA_NCORES'))
Install the front end. Windows users must have Rtools and git installed.
cola::setup_cola_dss() # Setup the DSS libraries. Takes a while. Run until getting a success message
.rs.restartR() # Restart R
library(cola) # Load Cola
cola::cola_dss() # Launch the DSS
A) Installation

> Run locally

Instruction for installing the DSS in your local machine



Figu

Appendix

Web-version / server

This DSS is hosted on web servers for specific workshops. The DSS uRL might change. For the latest URL check the GitHub web page.

Known issues

As we are developing thi software, the inclusion of new features might face some incompatibilities or issues in different platforms or particular study cases. This manual contains the usage instructions, however, please refer to the CoLa Github repo for the latest updates regarding installation issues.

- General package information: GitHub - connectingLandscapes/cola
- Installation details: GitHub - md_cola_install.md at main · connectingLandscapes/cola
- Installation issues: GitHub - known_issues.md at main · connectingLandscapes/cola

UNICOR

UNICOR users can see the following examples as the translation for CoLa. You can reproduce UNICOR results in CoLa with the following functions.

Kernels - CRK

The RIP file used (kernels.rip) to generate a kernels layer:

UNICOR Param	Value	COLA param
Session_label	case1A	NA
Grid_Filename	surface.rsg (.tif in CoLa)	intif
XY_Filename	coordinates.xy (.shp in CoLa)	outtif
Use_Direction	FALSE	NA
Type_Direction	FlowAcc	NA

Use_Resistance	TRUE	NA
Barrier_or_U_Filename	completebarr_test2.txt	NA
Direction_or_V_Filename	flowaccu_test2.txt	NA
Speed_To_Resistance_Scale	0;10	NA
Use_ED_threshold	False	NA
ED_Distance	10000	NA
Edge_Type	all_paths	NA
Transform_function	linear	shape
Const_kernal_vol	False	1
Kernel_volume	10000	NA
Edge_Distance	1000000	maxdist
Number_of_Processes	4	4
KDE_Function	Gaussian	NA
KDE_GridSize	2	NA
Number_of_Categories	5	NA
Save_Path_Output	TRUE	NA
Save_IndividualPaths_Output	FALSE	NA
Save_GraphMetrics_Output	FALSE	NA
Save_KDE_Output	FALSE	NA
Save_Category_Output	FALSE	NA
Save_CDmatrix_Output	FALSE	NA
NA	‘yes’	transf
NA	result.tif	outtif

The command line for UNICOR:

```
/path/to/python /path/to/UNICOR.py kernels.rip
```

The command line for CoLa:

```
/path/to/python /path/to/crk.py coordinates.shp surface.tif result.tif
1000000 linear 1 4 None
```

The CoLa R function:

```
library(cola)
crk_result <- crk_py(inshp = 'coordinates.shp', intif = 'surface.tif', outtif =
'result.tif', maxdist = 1000000, shape = 'linear', volume = 1, ncores = 4, crs =
'None')
```

Corridors - LCC

The RIP file used (corridor.rip) to generate a corridor layer:

UNICOR Param	Value	COLA param
Session_label	case1A	NA
Grid_Filename	Surface.rsg (.tif in CoLa)	intif
XY_Filename	coordinates.xy (.shp in CoLa)	inshp
Use_Direction	FALSE	NA
Type_Direction	FlowAcc	NA
Use_Resistance	TRUE	NA
Barrier_or_U_Filename	completebarr_test2.txt	NA
Direction_or_V_Filename	flowaccu_test2.txt	NA
Speed_To_Resistance_Scale	0;10	NA
Use_ED_threshold	False	NA
ED_Distance	10000	NA
Edge_Type	threshold	NA
Transform_function	linear	NA

Const_kernal_vol	True	NA
Kernel_volume	10000	NA
Edge_Distance	1000000	maxdist
Number_of_Processes	4	ncores
KDE_Function	Gaussian	NA
KDE_GridSize	2	smooth
Number_of_Categories	5	NA
Save_Path_Output	TRUE	NA
Save_IndividualPaths_Output	FALSE	NA
Save_GraphMetrics_Output	FALSE	NA
Save_KDE_Output	TRUE	NA
Save_Category_Output	FALSE	NA
Save_CDmatrix_Output	FALSE	NA
NA	0	tolerance
NA	result.tif	outtif

The command line for UNICOR:

```
/path/to/python /path/to/UNICOR.py corridors.rip
```

The command line for CoLa:

```
/path/to/python /path/to/lcc.py coordinates.shp surface.tif result.tif
1000000 2 0 4 None
```

The CoLa R function:

```
library(cola)
crk_result <- lcc_py(inshp = 'coordinates.shp', intif = 'surface.tif', outtif =
'result.tif', maxdist = 1000000, smooth = 2, tolerance = 0, ncores = 4, crs =
'None')
```

Distance matrix - CDMAT

The RIP file used (distance.rip) to generate a cost-distance matrix:

UNICOR Param	Value	COLA param
Session_label	case1A	NA
Grid_Filename	size1_side3px_10totpix.rsg	intif
XY_Filename	sabah_10.xy	inshp
Use_Direction	FALSE	NA
Type_Direction	FlowAcc	NA
Use_Resistance	TRUE	NA
Barrier_or_U_Filename	completebarr_test2.txt	NA
Direction_or_V_Filename	flowaccu_test2.txt	NA
Speed_To_Resistance_Scale	0;10	NA
Use_ED_threshold	False	NA
ED_Distance	10000	NA
Edge_Type	threshold	NA
Transform_function	linear	NA
Const_kernal_vol	False	NA
Kernel_volume	10000	NA
Edge_Distance	1000000	maxdist
Number_of_Processes	4	NA
KDE_Function	Gaussian	NA
KDE_GridSize	2	NA

Number_of_Categories	5	NA
Save_Path_Output	TRUE	NA
Save_IndividualPaths_Output	FALSE	NA
Save_GraphMetrics_Output	FALSE	NA
Save_KDE_Output	FALSE	NA
Save_Category_Output	FALSE	NA
Save_CDmatrix_Output	TRUE	NA
NA	result.csv	outcsv

The command line for UNICOR:

```
/path/to/python /path/to/UNICOR.py distance.rip
```

The command line for CoLa:

```
/path/to/python /path/to/create_cdmat.py coordinates.shp surface.tif
result.csv 150000 1 None
```

The CoLa R function:

```
library(cola)
crk_result <- cdmat_py(inshp = 'coordinates.shp', intif = 'surface.tif', outcsv =
'result.csv', maxdist = 125000, ncores = 1, crs = 'none')
```