Circuitscape.jl

April 8, 2026

Contents

Part I

Home

Chapter 1

Circuitscape Documentation

Circuitscape is an open-source Julia program that uses circuit theory to model connectivity in heterogeneous landscapes. Its most common applications include modeling movement and gene flow of plants and animals, as well as identifying areas important for connectivity conservation. Circuit theory complements commonly-used connectivity models because of its connections to random walk theory and its ability to simultaneously evaluate contributions of multiple dispersal pathways. Landscapes are represented as conductive surfaces, with low resistances assigned to landscape features types that are most permeable to movement or best promote gene flow, and high resistances assigned to movement barriers. Effective resistances, current flow, and voltages calculated across the landscapes can then be related to ecological processes, such as individual movement and gene flow.

More detail about the underlying model, its parameterization, and potential applications in ecology, evolution, and conservation planning can be found in McRae (2006) and McRae et al. (2008).

More detail about implementation can be found in the Circuitscape In Julia paper.

A PDF version of this documentation is also available.

Chapter 2

Quick Start Guide

This is a quick start guide. If you're looking for basics on how to use Circuitscape, refer to the usage guide.

To run Circuitscape, you need to install Julia. At the Julia prompt, install the Circuitscape package by running the following code

```
using Pkg
Pkg.add("Circuitscape")
```

You can also install Circuitscape in a local project as well.

2.1 Running a job

A Circuitscape job is fully described by an INI file. This configuration file consists of file paths to data as well as flag values. A detailed list of all INI arguments can be found in the Inputs, Outputs and Options section. Examples can be found in the test folder. You can also use a built-in terminal UI to build Circuitscape jobs. For more on that, skip ahead to Building a Circuitscape Job.

If you do have your INI file handy, you can run the job by:

```
using Circuitscape
compute("myjob.ini")
```

You can also run Circuitscape programmatically by passing a configuration dictionary:

```
using Circuitscape
cfg = Circuitscape.init_config()
cfg["habitat_file"] = "resistance_map.asc"
cfg["point_file"] = "focal_nodes.asc"
cfg["scenario"] = "pairwise"
cfg["output_file"] = "output/results.out"
compute(cfg)
```

2.2 Building a Circuitscape Job

The builder is kicked off by calling the `start()` function from the Julia prompt. It will build an INI file for you step by step, and either run the job directly or write the final INI file to a specified location. You can exit the builder at any time by hitting Ctrl+C.

Please note that this version of Circuitscape **does not** come with a GUI.

You can also manually write your own INI file by copying and pasting an example from the test folder and changing values as needed.

2.3 Citing Circuitscape

Please use the following to cite Circuitscape:

```
@article{hall2021circuitscape,
  title={Circuitscape in julia: empowering dynamic approaches to connectivity assessment},
  author={Hall, Kimberly R and Anantharaman, Ranjan and Landau, Vincent A and Clark, Melissa and
  ↪  Dickson, Brett G and Jones, Aaron and Platt, Jim and Edelman, Alan and Shah, Viral B},
  journal={Land},
  volume={10},
  number={3},
  pages={301},
  year={2021},
  publisher={Multidisciplinary Digital Publishing Institute}
}
```

2.4 Circuitscape.jl vs Circuitscape 4 (Python)

Circuitscape.jl is built entirely in the Julia language, offering significant performance advantages over the previous Python version (v4.0.5).

Faster and More Scalable

We benchmarked `Circuitscape.jl` with the Python version (v4.0.5) using 16 parallel processes and benchmark problems from the standard Circuitscape benchmark suite.

These benchmarks were run on a Linux (Ubuntu) server machine with the following specs:

- Name: Intel(R) Xeon(R) Silver 4114 CPU
- Clock Speed: 2.20GHz
- Number of cores: 20
- RAM: 384 GB

From the benchmark, Circuitscape.jl is up to *4x faster* on 16 processes. However, the best performing bar in the chart is *Julia-CHOLMOD*, which uses a direct solver.

CHOLMOD Solver

The CHOLMOD solver performs a Cholesky decomposition on the graph constructed, and performs a batched back substitution to compute the voltages. It plugs into the CHOLMOD library, which is part of the SuiteSparse collection of high performance sparse matrix algorithms.

To use this mode, include a line in your Circuitscape INI file:

```
solver = cholmod
```

The Cholesky decomposition is a direct solver method, unlike the algebraic multigrid method used by default. The advantage is that it can be much faster than the iterative solution for smaller problem sizes.

Word of caution: The Cholesky decomposition is not practical to use beyond a certain problem size because of a phenomenon called fill-in, which results in loss of sparsity and large memory consumption.

Pardiso Solver

Circuitscape also supports the Pardiso direct solver as a package extension. Pardiso requires Intel MKL, so it is only available on systems where MKL is installed. To use it, first install Pardiso.jl, then load it before Circuitscape:

```
using Pkg
Pkg.add("Pardiso")
```

```
using Pardiso         # must be loaded before or alongside Circuitscape
using Circuitscape
compute("myjob.ini")  # with solver = pardiso in the INI file
```

And set the solver in your INI file:

```
solver = pardiso
```

Pardiso requires double precision and will automatically switch if single precision is requested. Like CHOLMOD, it is a direct solver best suited for small to medium problem sizes.

Parallel Computing

Circuitscape.jl supports multi-threaded computation on Linux, macOS, and Windows using Julia's native threading. To run with multiple threads, start Julia with the `-t` flag:

```
julia -t 4    # use 4 threads
```

Or set the `JULIA_NUM_THREADS` environment variable before starting Julia.

You can also enable parallelism from the INI file:

```
parallelize = True
```

The AMG solver (default) parallelizes individual pair solves across threads, which can provide significant speedups for pairwise and one-to-all/all-to-one modes with many focal points. The CHOLMOD and Pardiso solvers perform batched direct solves; threading is used for postprocessing (current map accumulation and output writing) in these modes.

Single Precision (Experimental)

Circuitscape.jl supports running problems in single precision as opposed to the standard double precision.

Single precision usually takes much less memory, but trades off against solution accuracy.

Use this new feature by including a line in your config file:

```
precision = single
```

Chapter 3

Related Projects

1. Omniscape.jl - Omnidirectional connectivity analysis built on top of Circuitscape
2. AlgebraicMultigrid.jl - Algebraic Multigrid methods in Julia. This is the default solver used in Circuitscape.

Chapter 4

Further Reading

- Beier, P., W. Spencer, R. Baldwin, and B.H. McRae. 2011. Best science practices for developing regional connectivity maps. Conservation Biology 25(5): 879-892
- Dickson B.G., G.W. Roemer, B.H. McRae, and J.M. Rundall. 2013. Models of regional habitat quality and connectivity for pumas (*Puma concolor*) in the southwestern United States. PLoS ONE 8(12): e81898. doi:10.1371/journal.pone.0081898
- McRae, B.H. 2006. Isolation by resistance. Evolution 60:1551-1561.
- McRae, B.H. and P. Beier. 2007. Circuit theory predicts Gene flow in plant and animal populations. Proceedings of the National Academy of Sciences of the USA 104:19885-19890.
- McRae, B.H., B.G. Dickson, T.H. Keitt, and V.B. Shah. 2008. Using circuit theory to model connectivity in ecology and conservation. Ecology 10: 2712-2724.
- Shah, V.B. 2007. An Interactive System for Combinatorial Scientific Computing with an Emphasis on Programmer Productivity. PhD thesis, University of California, Santa Barbara.
- Shah,V.B. and B.H. McRae. 2008. Circuitscape: a tool for landscape ecology. In: G. Varoquaux, T. Vaught, J. Millman (Eds.). Proceedings of the 7th Python in Science Conference (SciPy 2008), pp. 62-66.
- Spear, S.F., N. Balkenhol, M.-J. Fortin, B.H. McRae and K. Scribner. 2010. Use of resistance surfaces for landscape genetic studies: Considerations of parameterization and analysis. Molecular Ecology 19(17): 3576-3591.
- Zeller K.A., McGarigal K., and Whiteley A.R. 2012. Estimating landscape resistance to movement: a review. Landscape Ecology 27: 777-797.

Part II

User Guide

Chapter 5

How Circuitscape Works

Whatever software you use, connectivity modeling involves a great deal of research, data compilation, GIS analyses, and careful interpretation of results. Defining areas to connect, parameterizing resistance models, and other modeling decisions you will need to make are not trivial. Before diving in, we strongly recommend that users first acquaint themselves with the process and challenges of connectivity modeling by consulting published resources. Good places to start include overviews on the Corridor Design and Connecting Landscapes websites. Spear et al. (2010), Beier et al. (2011) and Zeller et al. (2012) offer helpful advice on resistance mapping and connectivity analysis in general. Before using this software, users should acquaint themselves with the use of circuit theory for modeling connectivity (summarized in McRae et al. 2008).

See the Gnarly Landscape Utilities website for tools that can help to automate resistance and core area modeling.

Lastly, users interested in mapping important connectivity areas may wish to consider Linkage Mapper, which maps least-cost corridors. Linkage Mapper now also hybridizes least-cost corridor modeling with Circuitscape (see the Pinchpoint Mapper tool within the Linkage Mapper toolkit). Links to other connectivity tools can be found on the Corridor Design and Connecting Landscapes websites.

Circuitscape may be called through its own graphical user interface, from the Circuitscape for ArcGIS Toolbox, or from the command line. Users supply Circuitscape with resistance data and the program calculates effective resistances and/or creates maps of current flow and voltages across landscapes and networks.

Two data types: network and raster

Circuitscape reads either a network of nodes connected by links or a raster grid of resistances (Fig. 3). Links and raster cells are attributed with resistance values that reflect the degree to which the landscape facilitates or impedes movement. Networks and raster maps can be coded in resistances (with higher values denoting greater resistance to movement) or conductances (the reciprocal of resistance; higher values indicate greater ease of movement).

Fig. 1. Simple illustrations of network and raster data types used by Circuitscape. The program can operate on networks of nodes (left panel) or raster grids (right panel). Raster grid cells can have any resistance value. Here, cells with zero resistance ("short-circuit regions," which can be used to represent contiguous habitat patches) are shown in white, cells with a resistance of 1 are shown in gray, and a cell with infinite resistance (coded as NODATA) is shown in black.

For rasters, every grid cell with finite resistance is represented as a node in a graph, connected to either its four first-order or eight first- and second-order neighboring cells. Cells with infinite resistance (zero conductance) are dropped. Habitat patches, or collections of cells, can be assigned zero resistance (infinite conductance) using a separate "short-circuit region" file. These collections of cells are collapsed into a single node.

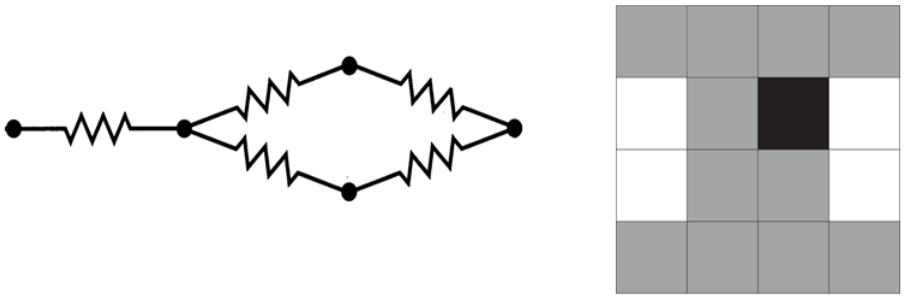

Figure 5.1:

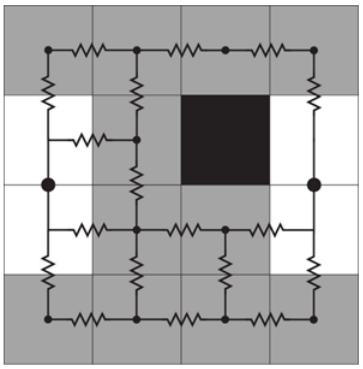

Figure 5.2:

Fig. 2. Raster grids are converted to electrical networks. Each cell becomes a node (represented by a dot), and adjacent cells are connected to their four or eight neighbors by resistors. Here, the two short-circuit regions have each been collapsed into a single node. The infinite resistance cell is dropped entirely from the network.

Calculation modes

Circuitscape operates in one of four modes: **pairwise**, **advanced**, **one-to-all**, and **all-to-one**. Pairwise and advanced modes are available for both raster and network data types. The one-to-all and all-to-one modes are available for raster data only.

In the **pairwise** mode, connectivity is calculated between all pairs of focal nodes (points or regions between which connectivity is to be modeled) supplied to the program in a single input file. For each pair of focal nodes, one node will arbitrarily be connected to a 1-amp current source, while the other will be connected to ground. Effective resistances will be calculated iteratively between all pairs of focal nodes, and, if selected, maps of current and voltage will be produced. If there are n focal nodes, there will be $n(n - 1)/2$ calculations **unless** you're using focal points (only one cell per focal node) and not mapping currents or voltages. In the latter case, we can do it in n calculations **(much faster)**.

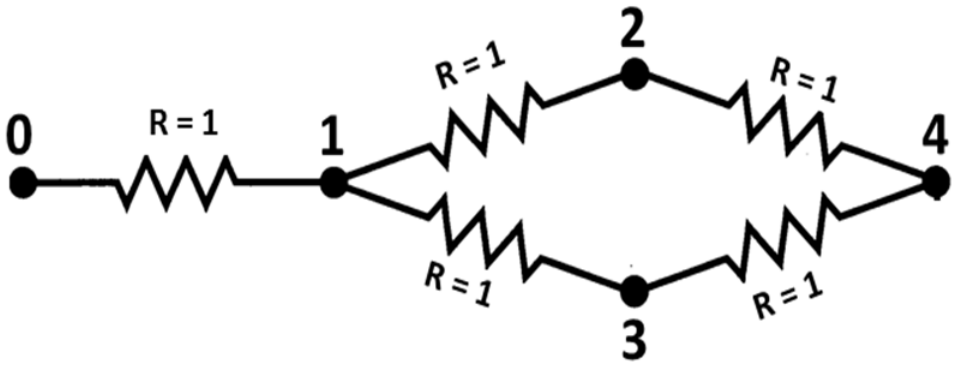

Figure 5.3:

The **advanced** mode offers much greater flexibility in defining sources and targets for current flow. The user defines any number of current sources and any number of grounds in a network or raster landscape, and these are all activated simultaneously. Sources represent points or areas from which current flows, whereas grounds represent nodes where current exits the system.

Source nodes can have different strengths (i.e. inject more or less current into the network or grid), and ground nodes can be tied to ground with any resistance. Current sources and grounds are supplied in separate input files.

Two other modes are available for raster data types only. The **one-to-all** mode is similar to the pairwise mode, and takes the same input files. However, instead of iterating across all focal node *pairs*, this mode iterates across all focal nodes. In each iteration, one focal node is connected to a 1-amp current source, while all remaining focal nodes are connected to ground. If there are *n* focal nodes, there will be *n* calculations.

The **all-to-one** mode is similar to the one-to-all mode, and takes the same input files. However, in this mode Circuitscape connects one focal node to ground and all remaining focal nodes to 1-amp current sources. It then repeats the process for each focal node; if there are *n* focal nodes, there will be *n* calculations.

Circuitscape can generate maps showing the current density and voltage at each node or cell under each configuration (and current flow for each link/resistor in networks). Additionally, Circuitscape writes a file reporting effective resistances between all pairs of focal nodes in the pairwise mode, and between each node and ground in the one-to-all mode. Resistances in the all-to-one mode are undefined, so a file is written with zeros indicating successful solves.

Illustrations of analyses with network data

For network data types, any node can be connected to any other node by a resistor:

Fig. 3. Example network. This network would be input as a **text list** specifying resistances between each pair of connected nodes (0-1, 1-2, 1-3, 2-3, and 2-4; see the *Input file formats* section below).

For **pairwise analysis** we would also supply a list of focal nodes (containing at least two node numbers, but as many as five, the number of nodes in the circuit) between which we want to perform calculations.

Fig 4. In pairwise mode, Circuitscape will iterate across pairs of nodes in a focal node list. If node 0 and node 4 are in the focal node list, then one of the iterations will look like the above, with a 1 amp current source connected to one node and the other grounded. Current will flow across the network from the source to the ground. Branch currents, node currents, node voltages, and effective resistances between node pairs can be written for each iteration.

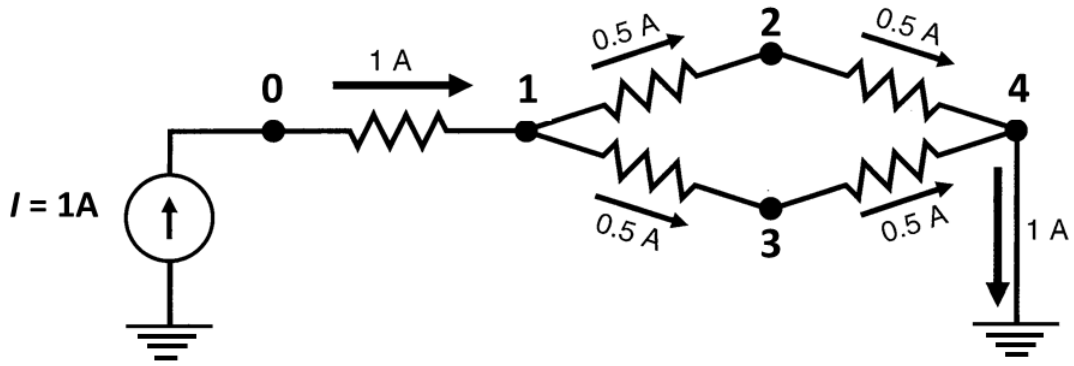

Figure 5.4:

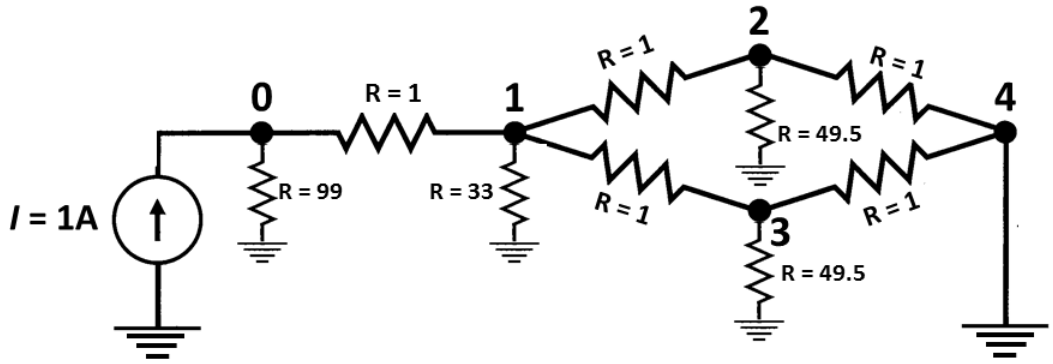

Figure 5.5:

More complexity can be added by running in **advanced mode**, which allows any number of sources and grounds to be activated simultaneously. For example, we could modify the circuit above by adding a single, fixed source at node zero and adding multiple grounds with different resistances. Current sources and grounds are entered in separate files.

Fig. 5. In advanced mode, any node can be tied to a current source or to ground, either directly or via resistors with any value (top panel). Currents passing through all nodes and links can then be calculated (bottom panel), and voltages can be calculated at each node. Circuit above is from McRae et al. (2008).

Illustrations of analyses with raster data

Fig. 6. Example raster input files for **pairwise, one-to-all, and all-to-one modes**. Input files in this example include a **resistance map** specifying per-cell resistances or conductances, a **focal node location file** (with two focal regions and one focal point in this case), and an optional **short-circuit region map**. Focal regions and short-circuit regions represent areas with zero resistance. Cells with the same region ID are considered perfectly connected and are collapsed into a single node, even if they are not contiguous.

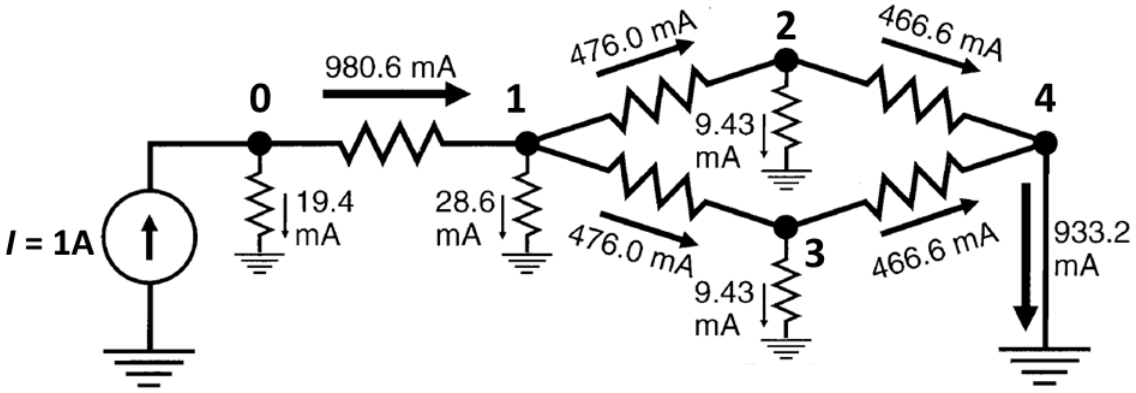

Figure 5.6:

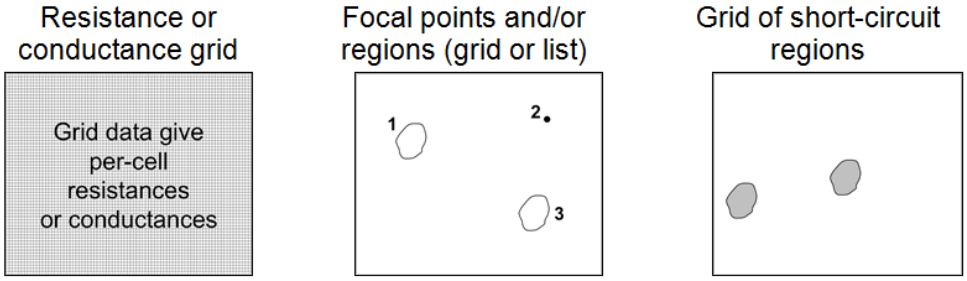

Figure 5.7:

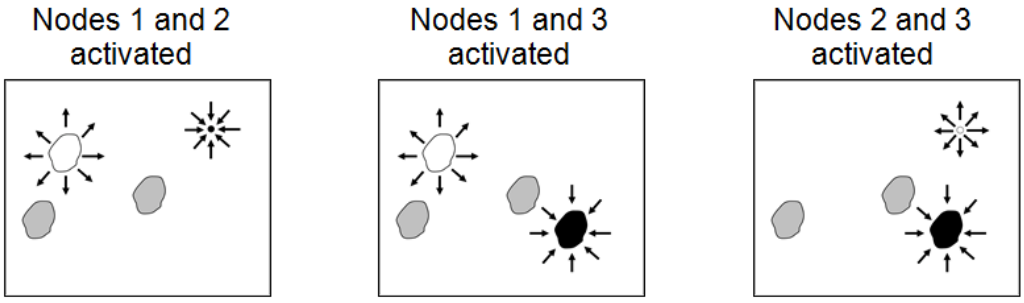

Figure 5.8:

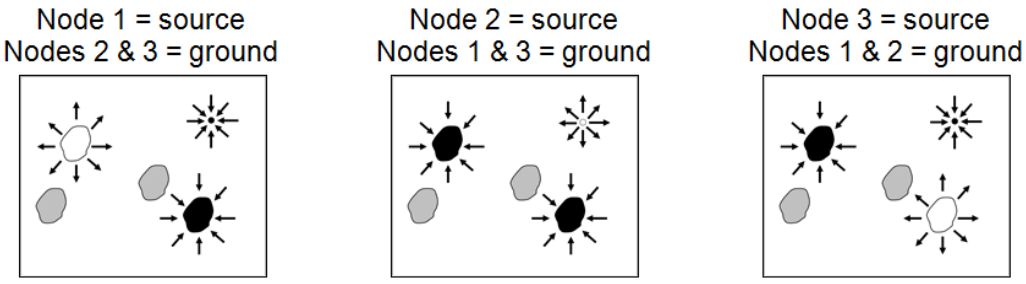

Figure 5.9:

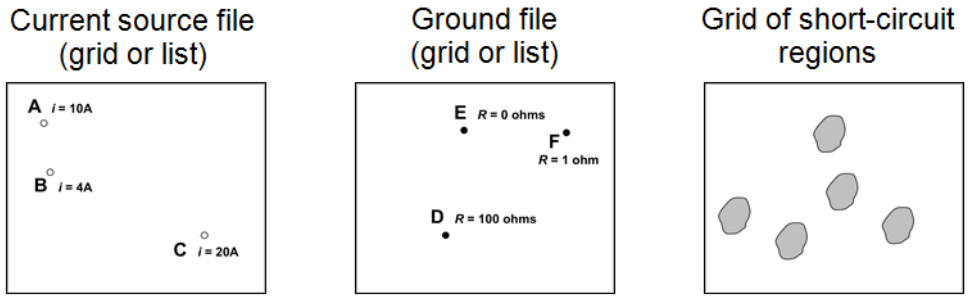

Figure 5.10:

Fig. 7. Schematic describing **pairwise** mode analyses that would result from the input files shown in Fig. 8. Three sets of pairwise calculations, involving focal nodes 1 and 2, nodes 1 and 3, and nodes 2 and 3, would be conducted. For each pair, one node would be connected to a 1-amp current source, and the other to ground. Note that focal region nodes become short-circuit regions when they are activated (e.g., node 1 in scenario 1), but these regions are not present when the nodes are not activated (e.g., node 1 in scenario 3).

Fig. 8. Schematic describing **one-to-all mode** analyses that would result from the input files shown in Fig. 8. Three sets of calculations, involving focal nodes 1, 2, and 3, would be conducted. For each, one node would be connected to a 1-amp current source, and the other two would be connected to ground. The all-to-one mode is similar, with arrow directions reversed; that is, one node is connected to ground while the remaining nodes are connected to 1-amp current sources.

Fig. 9. Example raster input files for **advanced mode**, which requires independent **current source and ground files**. Note that current sources in this example have different "strengths," and ground nodes are connected to ground with differing levels of resistance. This example also includes an optional grid with five short-circuit regions.

Fig. 10. The first two panels show the "effective" configuration resulting from the input files in Fig. 11. Because current source C and grounds D and E overlap with short-circuit regions, these short-circuit regions effectively become sources or grounds themselves. The rightmost panel shows a schematic of the resulting analysis, with all sources (white points and polygons) and grounds (black points and polygons) activated simultaneously. Note that sources may be negative (drawing current out of the system), and ground nodes can actually contribute current to the system when negative sources are present.

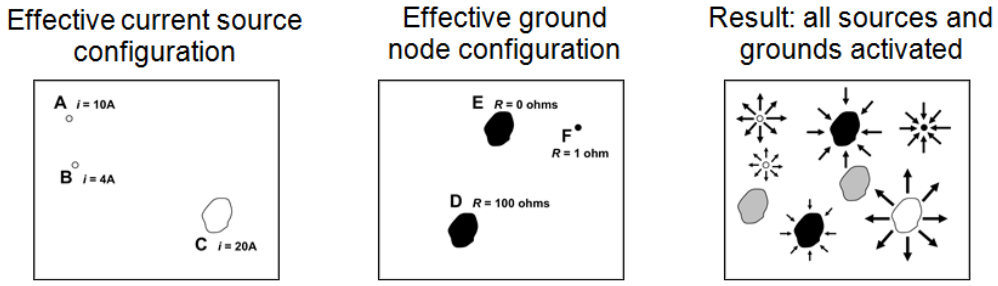
Effective current source configuration
A i = 10A
B i = 4A
C i = 20A
Effective ground node configuration
E R = 0 ohms
F
R = 1 ohm
D R = 100 ohms
Result: all sources and grounds activated

Figure 5.11:

Part III

Inputs, Outputs and Options

Chapter 6

Options and Flags

6.1 INI Argument Reference

All Circuitscape configuration is done through an `.ini` file. Below is a complete reference of all arguments, organized by section, with their types, default values, and descriptions.

Circuitscape Mode

Argument	Type	Default	Description
`data_type`	String	`raster`	Input data type. Values: `raster`, `network`.
`scenario`	String	not entered	Modeling mode. Values: `pairwise`, `advanced`, `one-to-all`, `all-to-one`.

Habitat Raster or Graph

Argument	Type	Default	Description
`habitat_file`	Path	—	Path to the resistance/conductance map file (raster or network).
`habitat_map_is_resistances`	Boolean	`True`	If `True`, habitat map values are resistances. If `False`, values are conductances.

Connection Scheme for Raster Habitat Data

Argument	Type	Default	Description
`connect_four_neighbors_only`	Boolean	`False`	If `True`, connect each cell to 4 cardinal neighbors only. If `False`, connect to all 8 neighbors.
`connect_using_avg_resistances`	Boolean	`False`	If `True`, use average resistance for cell connections. If `False`, use average conductance.

Argument	Type	Default	Description
`use_polygons`	Boolean	`False`	If `True`, read a short-circuit region file. Cells within each region are collapsed into a single node with zero resistance.
`polygon_file`	Path	—	Path to the short-circuit region map. Must have the same cell size and extent as the resistance grid.

Short-Circuit Regions (Polygons)

Options for Advanced Mode

Argument	Type	Default	Description
`source_file`	Path	—	Path to the current source file (raster or text list). Values specify source strength in amps.
`ground_file`	Path	—	Path to the ground point file (raster or text list). Values specify resistance or conductance to ground.
`ground_file_is_resistances`	Boolean	`True`	If `True`, ground file values are resistances. If `False`, conductances. Set to `False` with value 0 to connect directly to ground.
`use_unit_currents`	Boolean	`False`	If `True`, all current sources are set to 1 Amp regardless of values in the source file.
`use_direct_grounds`	Boolean	`False`	If `True`, all ground nodes are tied directly to ground (R=0) regardless of values in the ground file.
`remove_src_or_gnd`	String	`keepall`	When a source and ground are at the same node: `keepall`, `rmvsrc`, `rmvgnd`, or `rmvall`.

Options for Pairwise, One-to-All, and All-to-One Modes

Argument	Type	Default	Description
`point_file`	Path	—	Path to focal node file (raster or text list). Each focal node must have a unique positive integer ID.
`use_included_pairs`	Boolean	`False`	If `True`, only run calculations on a subset of focal node pairs specified in the included pairs file.
`included_pairs_file`	Path	—	Path to file specifying pairs to include or exclude from calculations.

Options for One-to-All and All-to-One Modes

Argument	Type	Default	Description
`use_variable_source_strengths`	Boolean	`False`	If `True`, read per-node source strengths from a file instead of using 1 Amp for all.
`variable_source_file`	Path	`None`	Path to text file with focal node IDs and corresponding source strengths.

Argument	Type	Default	Description
`use_mask`	Boolean	`False`	If `True`, apply a mask to the resistance map. Cells with negative, zero, or NODATA values in the mask are dropped.
`mask_file`	Path	`None`	Path to the raster mask file.

Mask File

Output Options

Argument	Type	Default	Description
`output_file`	Path	—	Base path and filename for all output files.
`write_cur_maps`	Boolean	`False`	If `True`, write current maps for each iteration and a cumulative map.
`write_volt_maps`	Boolean	`False`	If `True`, write voltage maps for each iteration.
`write_cum_cur_map_only`	Boolean	`False`	If `True`, calculate current maps for each iteration but only write the cumulative (summed) map to disk.
`write_max_cur_maps`	Boolean	`False`	If `True`, write a map showing the maximum current value at each cell across all iterations.
`set_null_currents_to_nodata`	Boolean	`False`	If `True`, set cells with zero current to NODATA in output current maps.
`set_null_voltages_to_nodata`	Boolean	`False`	If `True`, set cells with zero voltage to NODATA in output voltage maps.
`set_focal_node_currents_to_zero`	Boolean	`False`	If `True`, set current at focal nodes to zero in output maps. *Note: not yet implemented in Circuitscape 5.*
`compress_grids`	Boolean	`False`	If `True`, compress output ASCII grids using gzip.
`log_transform_maps`	Boolean	`False`	If `True`, log10-transform values in output current maps. Cells with zero current are set to NODATA.
`write_as_tif`	Boolean	`False`	If `True`, write output rasters as GeoTIFF instead of ASCII grid format.

Calculation Options

Argument	Type	Default	Description
`solver`	String	`cg+amg`	Linear solver to use. Values: `cg+amg` (iterative, recommended for large grids), `cholmod` (direct, uses more memory), `accelerate` (direct, macOS only, requires AppleAccelerate.jl), `pardiso` (direct, requires Pardiso.jl).
`precision`	String	`Double`	Floating-point precision. Values: `Double`, `Single`. CHOLMOD and Pardiso require double precision.
`use_64bit_indexing`	Boolean	`True`	If `True`, use 64-bit integer indexing. Required for very large grids.
`cholmod_batch_size`	Integer	`1000`	Number of pairs to solve simultaneously when using CHOLMOD in pairwise mode.
`parallelize`	Boolean	`False`	If `True`, run iterations in parallel using Julia threads. Start Julia with `julia -t N` for N threads.

| `low_memory_mode` | Boolean | `False` | If `True`, reduce memory usage at the cost of computation time. | | `preemptive_memory_release` | Boolean | `False` | If `True`, release memory more aggressively during computation. |

Reclassification

Argument	Type	Default	Description
`use_reclass_table`	Boolean	`False`	If `True`, reclassify resistance values using a lookup table. *Note: not yet implemented in Circuitscape 5.*
`reclass_file`	Path	—	Path to file with reclassification data.

Logging

Argument	Type	Default	Description
`log_file`	Path	`None`	Path to log file. If `None`, no file logging.
`log_level`	String	`INFO`	Logging level. Values: `DEBUG`, `INFO`. When set to `DEBUG`, prints detailed timing summary and per-pair solve progress.
`suppress_messages`	Boolean	`False`	If `True`, suppress all informational messages.
`profiler_log_file`	Path	`None`	Path to profiler log file.

6.2 Data Type

Set `data_type` to `raster` or `network` to choose whether you will be analyzing raster grid or network data.

6.3 Modeling Mode

Circuitscape is run in one of four modes (set via `scenario`). Pairwise and advanced modes are available for both raster and network data types. The one-to-all and all-to-one modes are available for raster data only.

6.4 Resistance Map or Network/Graph

The resistance file (`habitat_file`) specifies the ability of each cell in a landscape or link in a network to carry current. File formats are described in the *Input file formats* section below.

Most users code their data in terms of resistances (with higher values denoting greater resistance to movement). Set `habitat_map_is_resistances = False` to specify conductances instead (conductance is the reciprocal of resistance; higher values indicate greater ease of movement).

Zero and infinite values for conductances and resistances represent special cases. Infinite resistances are coded as NODATA values in input resistance grids, or as zero or NODATA in input conductance grids; these are treated as complete barriers and are disconnected from all other cells. For raster analyses, cells with zero resistance (infinite conductance) can be specified using a separate short-circuit region file as described below.

6.5 Focal Nodes (Pairwise, One-to-All, and All-to-One Modes)

The focal node file (`point_file`) specifies locations of nodes between which effective resistance and current flow are to be calculated. **Each focal node should have a unique positive integer ID.** Files may be text lists specifying coordinates or raster grids. When a grid is used, it must have the same cell size and extent as the resistance grid. Cells that do not contain focal nodes should be coded with NODATA values.

For raster analyses, focal nodes may occur at points (single cells on the resistance grid) or across regions. For the latter, a single ID would occupy more than one cell in a grid or more than one pair of coordinates in a text list (and falling within more than one cell in the underlying resistance grid). Cells within a single region are collapsed into a single node. The difference from short-circuit regions is that a focal region will be "burned in" to the resistance grid only for pairwise calculations that include that focal node. Focal regions need not be contiguous. For large grids, focal regions may require more computation time. When calculating resistances on large raster grids and not creating voltage or current maps, focal points will run much more quickly.

Parallelism

Circuitscape uses Julia's native threading for parallel computation. Set `parallelize = True` and start Julia with `julia -t N` for N threads.

Parallelism is supported in raster **pairwise**, **one-to-all**, and **all-to-one** modes. It is most effective in one-to-all and all-to-one, where each focal point is an independent solve that can be distributed evenly across threads.

In **pairwise** mode, work is distributed by source focal point: the first task solves N-1 pairs, the second solves N-2, and so on. This triangular load distribution means early tasks do significantly more work than later ones, reducing parallel efficiency when the number of threads is large relative to the number of focal points. Use `log_level = DEBUG` to see per-task timing breakdowns.

Network modes and raster advanced mode do not currently support parallelism.

6.6 Advanced Mode Options

Current Source File

The source file (`source_file`) specifies locations and strengths, in amps, of current sources. Either a raster or a text list may be used. Rasters must have the same cell size, projection, and extent as the resistance grid, and cells that do not contain current sources should be coded with NODATA values. Current sources may be positive or negative (i.e., they may inject current into the grid or pull current out). Similarly, grounds may either serve as a sink for current or may contribute current if there are negative current sources in the grid.

Ground Point File

The ground file (`ground_file`) specifies locations of ground nodes and resistances or conductances of resistors tying them to ground. Either a raster or a text list may be used. Rasters must have the same cell size, projection, and extent as the resistance grid, and cells that do not contain grounds should be coded with NODATA values. If a direct (R = 0) ground connection conflicts with a current source, the ground will be removed unless `remove_src_or_gnd` is set to `rmvgnd` or `keepall`.

Set `ground_file_is_resistances = False` if your ground point file specifies connections to ground in terms of conductance instead. To tie cells directly to ground, keep `ground_file_is_resistances = True` and set values in the ground point file to zero.

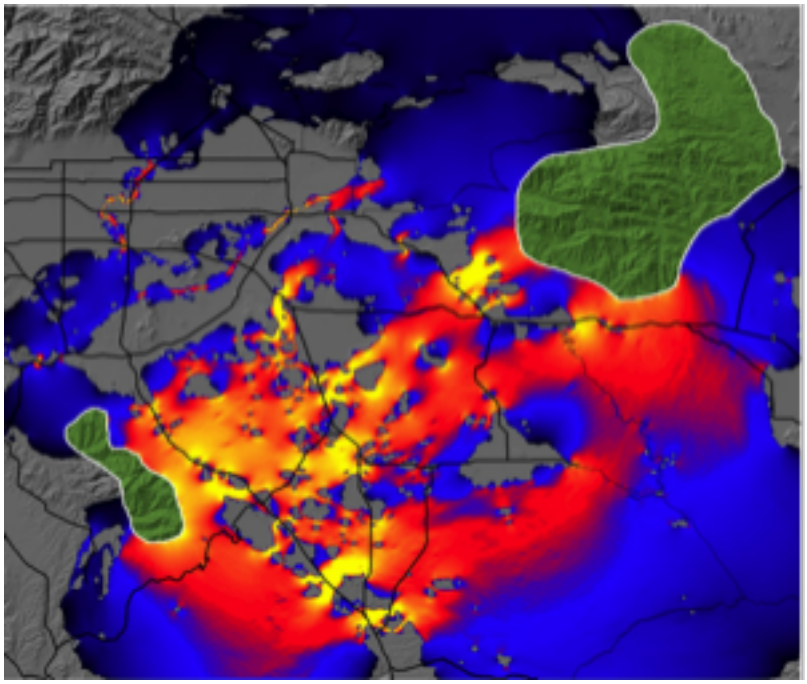

Figure 6.1:

Unit Currents and Direct Grounds

Set `use_unit_currents = True` to force all current sources to 1 Amp, regardless of the value specified in the source file. Set `use_direct_grounds = True` to tie all ground nodes directly to ground (R=0) regardless of the ground file values.

Source/Ground Conflicts

When a cell is connected both to a current source and to ground, `remove_src_or_gnd` determines the behavior: `keepall` (default), `rmvsrc`, `rmvgnd`, or `rmvall`. When using `keepall`, if a source is tied directly to ground (zero resistance), the ground connection will be removed.

6.7 Output Options

Current Maps

When `write_cur_maps = True`, current maps will be generated for every pair of focal nodes in pairwise mode, or for the source/ground configuration in advanced mode. Current maps have the same dimension as the input files, with values at each cell representing the amount of current flowing through the node. In pairwise mode, a current map file will be created for each focal node pair, and a cumulative (additive) map will also be written. (Note that for a given pair of focal nodes, current maps are identical regardless of which node is the source and which is the ground due to symmetry.) For advanced mode, a single map will be written. Such maps can be used to identify areas which contribute most to connectivity between focal points (McRae et al. 2008).

Fig. 1. Current map used to predict important connectivity areas between core habitat patches (green polygons, entered as focal regions) for mountain lions. Warmer colors indicate areas with higher current density. "Pinch points," or areas where connectivity is most tenuous, are shown in yellow. Research Collaborators: Brett Dickson and Rick Hopkins, Live Oak Associates.

Voltage Maps

When `write_volt_maps = True`, voltage maps are written. In pairwise mode, these give node voltages observed for each focal node pair if one node were connected to a 1 amp current source and the other to ground. In advanced mode, voltage maps show voltages resulting from the source and ground configurations.

6.8 Calculation Options

Cell Connectivity

For raster operations, Circuitscape creates a graph by connecting cells to their neighbors. Set `connect_four_neighbors_only = True` for 4 cardinal neighbors only (default is 8, including diagonals).

Average Resistance vs Conductance

Set `connect_using_avg_resistances = True` to connect cells by their average resistance instead of average conductance (the default).

The distinction is particularly important when connecting cells with zero or infinite values. When average resistances are used, first-order neighbors are connected by resistors with resistance: $Rab = (Ra + Rb) / 2$, and second-order (diagonal) neighbors by: $Rab = \text{sqrt}(2) * (Ra + Rb) / 2$. When average conductances are used, first-order neighbors are connected by: $Gab = (Ga + Gb) / 2$, and second-order (diagonal) neighbors by: $Gab = (Ga + Gb) / (2 * \text{sqrt}(2))$.

6.9 Mapping Options

Maximum Current Maps

In pairwise, one-to-all, and all-to-one modes, current maps are created for every iteration. By default, Circuitscape writes a cumulative map showing the sum of values at each node or grid cell across all iterations. Set `write_max_cur_maps = True` to also write a map showing the maximum current value at each cell across iterations.

Cumulative Maps Only

Set `write_cum_cur_map_only = True` to calculate current maps for each iteration but only write the cumulative (and optionally maximum) map to disk. This saves disk space when many iterations are performed.

Compress Output Grids

Set `compress_grids = True` to compress output ASCII grids using gzip. This can be useful when many large maps will be written.

Log-Transform Current Maps

Set `log_transform_maps = True` to apply a log10 transform to current densities in output maps. Cells with zero current will be set to NODATA values.

Set Focal Node Currents to Zero

When `set_focal_node_currents_to_zero = True`, focal nodes will have zero current in output maps when they are activated. For pairwise mode, cumulative maps will still show currents flowing through focal regions from other pairs being activated. This helps show the importance of each focal region for connecting others (see Dickson et al. 2013). *Note: not yet implemented in Circuitscape 5.*

6.10 Optional Input Files

Mask File

Set `use_mask = True` and provide `mask_file` to apply a raster mask. Cells with negative, zero, or NODATA values in the mask will be dropped from the resistance map (treated as complete barriers). Positive integer cells will be retained. File should only contain integers and be in raster format.

Short-Circuit Region Map

Set `use_polygons = True` and provide `polygon_file` to load short-circuit regions. These act as areas of zero resistance, providing patches through which current gets a "free ride." Each region should have a unique positive integer identifier; cells within each region are merged into a single node, including non-adjacent cells (regions need not be contiguous). Non-region areas should be stored as NODATA values. The file must have the same cell size and extent as the resistance grid.

Variable Source Strengths

In one-to-all and all-to-one modes, set `use_variable_source_strengths = True` and provide `variable_source_file` with focal node IDs and corresponding source strengths. The file should be a text list with two columns (ID followed by source strength). All nodes not in the list will default to 1 Amp.

Include/Exclude Focal Node Pairs

Set `use_included_pairs = True` and provide `included_pairs_file` to restrict calculations to a subset of focal node pairs. Users can either identify pairs to include or pairs to exclude, as specified in the first line of the file. This affects all modes except advanced mode. Files should be in tab-delimited text with a .txt extension.

6.11 Input Raster Format

Raster input maps should be stored in Arc/Info ASCII grid or GeoTIFF format, as exported by standard GIS packages. Set `write_as_tif = True` to produce GeoTIFF output instead of ASCII grids. For focal nodes, the value stored in each grid location refers to the focal node ID, and a single ID can occupy more than one cell (IDs must be positive integers). For current sources, the grid value specifies the source strength in amps. For grounds, the grid value specifies either the resistance or conductance of the resistor tying each ground node to ground.

The ASCII raster format is as follows:

Header:

```
ncols        <Number of columns>
nrows        <Number of rows>
xllcorner    <X coordinate of lower left corner>
yllcorner    <Y coordinate of lower left corner>
cellsize     <size of each cell>
NODATA_value <Code for cells with no habitat, focal nodes, sources or grounds>
```

Body (grid data):

Numeric data only. Columns are delimited with tabs and rows are delimited with new line characters.

Examples

Below is a 10 x 10 resistance map. Cells with infinite resistance are assigned NODATA values (-9999):

```
ncols         10
nrows         10
xllcorner     1
yllcorner     1
cellsize      1
NODATA_value  -9999
130    168    153    -9999  14     12    13     107    140    171
104    3      2      -9999  13     158   12     14     13     114
124    2      2      12     -9999  -9999 13     161    4      5
184    5      4      14     13     14    -9999  13     4      4
105    143    103    169    -9999  115   10     -9999  166    14
187    1      163    188    121    142   14     175    -9999  10
198    11     110    115    149    2     2      164    3      -9999
100    11     193    14     12     4     2      1      11     13
-9999  11     12     11     10     12    167    157    181    157
-9999  -9999  122    134    12     157   192    184    190    172
```

Below is a 10 x 10 focal region map. Groups of cells have been coded as focal regions that will be treated as "core area polygons" to be connected in circuit analyses. All cells within each focal region will be collapsed into a single node (even the non-contiguous cell in region #1) when that region is activated in pairwise, one-to-all, or all-to-one analyses. This format is identical to the short-circuit region file format.

```
ncols                10
nrows                10
xllcorner            1
yllcorner            1
cellsize             1
NODATA_value -9999
-9999  -9999  -9999  -9999  -9999  -9999  -9999  -9999  -9999  -9999
-9999  1      1      -9999  -9999  -9999  -9999  -9999  -9999  -9999
-9999  1      1      -9999  -9999  -9999  -9999  -9999  3      3
-9999  1      1      -9999  -9999  -9999  -9999  -9999  3      3
-9999  -9999  -9999  -9999  -9999  -9999  -9999  -9999  -9999  -9999
-9999  1      -9999  -9999  -9999  -9999  -9999  -9999  -9999  -9999
-9999  -9999  -9999  -9999   2      2     -9999  -9999  -9999  -9999
-9999  -9999  -9999  -9999   2      2      2     -9999  -9999  -9999
-9999  -9999  -9999  -9999  -9999  -9999  -9999  -9999  -9999  -9999
-9999  -9999  -9999  -9999  -9999  -9999  -9999  -9999  -9999   -9999
```

Note that regions 1 and 2 are well-connected by a low-resistance corridor in the resistance map above. Region 3 is connected to the other two regions only if cells are connected to their eight neighbors. In the four-neighbor case, region 3 would be completely isolated.

6.12 Text List File Format

For network/graph operations, resistor networks, focal nodes, current sources, and grounds should be stored as text lists (saved with a ".txt" extension). To specify a network of resistors, three columns are used. The first and second columns give the node IDs being connected by a resistor, and the third column gives the resistance value. For example, the simple circuit:

can be defined by the following text list:

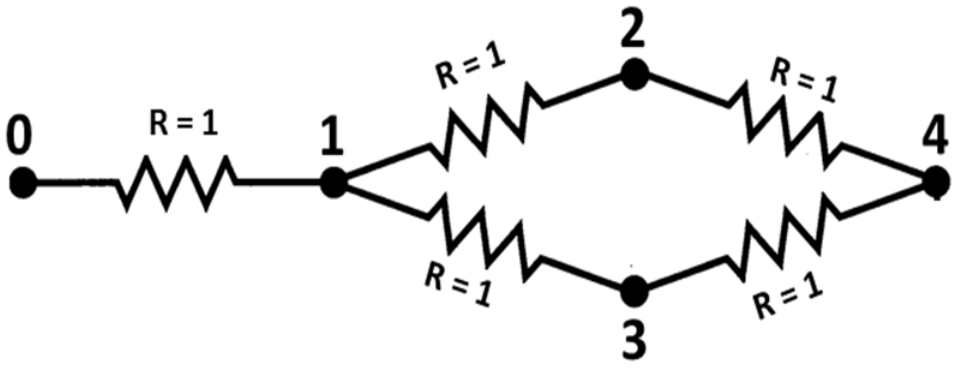

Figure 6.2:

```
0    1    1
1    2    1
1    3    1
2    4    1
3    4    1
```

Please note: typically, there should just be one entry for each pair of connected nodes. If there are two entries for a single pair in the form of (node1, node2, value1) and (node2, node1, value2), these will be considered parallel resistors and their conductances will be summed. For example, if the above text list had an extra entry for node pair (4, 3) like this:

```
0    1    1
1    2    1
1    3    1
2    4    1
3    4    1
4    3    1
```

then the resistance between nodes 3 and 4 in the resulting graph would be 1/2 ohm.

For advanced mode, current sources and grounds are also stored as text lists. The above circuit can be expanded to include a current source and grounds with two extra input files. For example, we can add a 1 Amp current source at node 0 with a file that looks like this:

```
0    1
```

To tie node 4 directly to ground (i.e. to connect it to ground with a wire that has a resistance of 0 Ohms) and connect the remaining nodes to ground with resistors, we can use a file that looks like this:

```
0    99
1    33
2    49.5
3    49.5
4    0
```

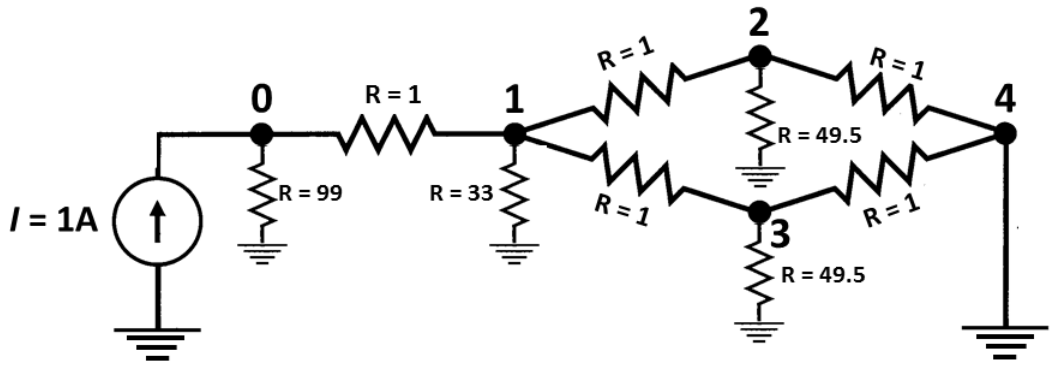

Figure 6.3:

The resulting circuit would look like this (from McRae et al. 2008):

For **raster** operations, you can also store focal nodes, current sources, and grounds as text lists (saved with a ".txt" extension). For each node referenced in a text list, a value and X and Y coordinates are specified as shown below.

```
Value1 X1 Y1
Value2 X2 Y2
…
```

Note: X and Y are geographical coordinates, not row and column numbers.

Example text list (a partial list of the cell locations in the focal region map above; coordinates are for cell centroids):

```
1    2.5    9.5
1    3.5    9.5
1    2.5    8.5
1    3.5    8.5
1    2.5    7.5
1    3.5    7.5
1    2.5    5.5
2    6.5    4.5
...
```

For focal nodes, the value field references the focal node ID; values must be positive integers, and a single ID can occupy more than one pair of coordinates (and more than one cell in the underlying resistance grid). For current sources, the value field references the source strength in amps. For grounds, the value field references either the resistance or conductance of the resistor tying each ground node to ground.

6.13 Include/Exclude File Format

This file is used when `use_included_pairs = True`, and affects all modes except advanced mode. There are two file formats that can be used. The first is the simplest, and gives a list of pairs to include in calculations,

or pairs to exclude, as specified in the first line of the file. For example, if there are five focal nodes, numbered 1-5, and the following list is entered, only pairs (1,2), (1,3), and (1,5) will be analyzed:

```
mode    include
1        2
1        3
1        5
```

Similarly, if the first line in the above file read:

```
mode     exclude
```

all pairs except (1,2), (1,3), and (1,5) would be analyzed.

The second method uses a matrix identifying which pairs of focal nodes to connect. The file specifies minimum and maximum values in the matrix to consider a pair connected. This method can be useful when used with a distance matrix to only run analyses between points separated by a minimum distance, or by a distance equal to or less than a maximum distance. Note: any focal node not in the matrix will be dropped from analyses. Entries on the diagonal are ignored. For example, in the following matrix, only pairs with entries between 2 and 50 are connected. Pairs (1,2), (2,4), and (3,4) will not be analyzed. Focal node 5 will be dropped entirely:

```
min    2
max    50
0     1     2     3     4     5
1     0     100   6.67  7     1
2     100   0     11    1     60
3     6.67  11    0     -1    100
4     7     1     -1    0     0
5     1     60     100  0     0
```

Make sure to include a zero in the upper-left corner of the matrix.

Files should be in tab-delimited text with a .txt extension.

6.14 Output Files

Current and Voltage Data

Current and voltage data for networks are written in text list formats. Raster voltage and current maps are written in ASCII raster format (or GeoTIFF if `write_as_tif = True`).

Resistance Files

Resistance data are written in both matrix and 3-column formats.

Here are pairwise resistances written to the output directory for the eight neighbor case (using per-cell resistances and average resistances for cell connection calculations). The first row and column contain the focal node IDs:

```
  0          1            2            3
  1          0            11.93688471  15.03634473
  2          11.93688471  0            11.57640568
  3          15.03634473  11.57640568  0
```

Here are pairwise resistances written to the output directory for the four neighbor case, in which focal node 3 was completely isolated (-1 indicates infinite resistance):

```
0          1            2            3
1          0            33.55792693  -1
2          33.55792693  0            -1
3          -1           -1           0
```

For convenience, resistances are also written to a separate file in a 3-column format, e.g.:

```
1      2       33.55792693
1      3       -1
2      3       -1
```

Part IV

On Solvers and Computation Time

Chapter 7

Computational Limitations, Speed, and Landscape Size

We have tested this code on landscapes with up to 437 million cells. Increasing numbers of connections using diagonal (eight neighbor) connections will decrease the size of landscapes that can be analyzed. Also, increasing landscape size or numbers of focal nodes will increase computation time. Note that due to the matrix algebra involved with solving many pairs of focal nodes, Circuitscape will run much faster when focal points (each focal node falls within only one grid cell), rather than focal regions (at least one focal node occupies multiple grid cells), are used.

7.1 Memory Limitations

There are several ways to increase the solvable grid size:

- Set impermeable areas of your resistance map to NODATA
- Use focal points instead of regions in pairwise mode
- Connect cells to their four neighbors only (`connect_four_neighbors_only = True`)
- Disable current and voltage maps (`write_cur_maps = False`, `write_volt_maps = False`)
- Use the one-to-all or all-to-one modes, which typically use less memory and run more quickly than pairwise mode
- Use the `cg+amg` solver instead of `cholmod`, `accelerate`, or `pardiso` for large grids (direct solvers use significantly more memory)
- Coarsen your grids (use larger cell sizes) – this often produces qualitatively similar results (see McRae et al. 2008)

The all-to-one mode can be an alternative to pairwise mode when the goal is to produce a cumulative map of important connectivity areas among multiple source/target patches.

7.2 Multi-threading

Circuitscape uses Julia's native threading for parallel computation. Start Julia with multiple threads to take advantage of this:

```
julia -t 4    # use 4 threads
```

The `cg+amg` solver benefits most from threading — each focal point pair is solved independently on a separate thread. For problems with many focal points, this can provide significant speedups.

The `cholmod`, `accelerate`, and `pardiso` solvers perform batched direct solves. Threading in these modes parallelizes the postprocessing step (current map accumulation and output writing).

7.3 Default Solvers: CG+AMG and CHOLMOD

Circuitscape ships with two solvers that work out of the box with no additional packages:

- **CG+AMG** (`solver = cg+amg`, the default) — an iterative solver using conjugate gradient with an algebraic multigrid preconditioner. This is the best choice for large grids because memory usage scales well with problem size. It also parallelizes individual pair solves across threads.
- **CHOLMOD** (`solver = cholmod`) — a direct solver using Cholesky factorization from SuiteSparse. It can be significantly faster than CG+AMG for small to medium problems, but memory usage grows quickly due to fill-in, making it impractical for very large grids. In pairwise mode it performs batched solves controlled by the `cholmod_batch_size` parameter.

7.4 Pardiso Solver

Circuitscape supports the Pardiso direct solver as a package extension. Pardiso uses Intel MKL's sparse direct solver and can offer excellent performance on Intel hardware. To use it, first install Pardiso.jl:

```
using Pkg
Pkg.add("Pardiso")
```

Then load it before (or alongside) Circuitscape:

```
using Pardiso
using Circuitscape
compute("config.ini")  # with solver = pardiso in the INI file
```

Pardiso requires double precision and will automatically switch if single precision is requested. Like CHOLMOD, it is a direct solver best suited for small to medium problem sizes, and uses the same batched solve strategy in pairwise mode.

7.5 Apple Accelerate Solver

On macOS (13.4 or later), Circuitscape can use Apple's Accelerate framework for sparse Cholesky factorization. This is a direct solver that can provide high performance on Apple Silicon hardware. To use it:

```
using AppleAccelerate
using Circuitscape
compute("config.ini")  # with solver = accelerate in the INI file
```

Or install `AppleAccelerate` first:

```
using Pkg
Pkg.add("AppleAccelerate")
```

The Accelerate solver supports both single and double precision, and uses the same batched solve strategy as the CHOLMOD and Pardiso solvers.

Note

Circuitscape sets BLAS to single-threaded at startup. Its workload is predominantly sparse matrix operations which do not benefit from multi-threaded BLAS.

Part V

Calling Circuitscape from other Programs

Chapter 8

Calling Circuitscape from Other Programs

8.1 From Julia

Circuitscape can be called directly from Julia:

```
using Circuitscape
result = compute("myconfig.ini")
```

You can also pass a configuration dictionary directly:

```
using Circuitscape
cfg = Circuitscape.init_config()
cfg["habitat_file"] = "resistance_map.asc"
cfg["point_file"] = "focal_nodes.asc"
cfg["scenario"] = "pairwise"
cfg["output_file"] = "output/results.out"
result = compute(cfg)
```

8.2 From R

You can call Circuitscape from R using the JuliaCall R package.

8.3 From Python

You can call Circuitscape from Python using the juliacall Python package.

Part VI

Logging Options

Chapter 9

Logging Options

Circuitscape uses a custom `CSLogger` built on Julia's `AbstractLogger` interface. The logger is configured automatically when you call `compute()`, based on settings in your INI file.

9.1 Suppressing Messages

Set `suppress_messages = True` in your INI file to suppress informational messages during computation. Warnings will still be displayed.

9.2 Logging to File

Set `log_file` in your INI file to write log messages to a file:

```
log_file = /path/to/logfile.log
```

When a log file is set, messages are written to both the console and the file.

9.3 Disabling Log Output

To disable all informational log messages from Julia's side, use the built-in logging system:

```
using Logging
Logging.disable_logging(Logging.Info)
```

To re-enable:

```
Logging.disable_logging(Logging.Debug)
```

9.4 UI Callback

Circuitscape exposes a `ui_interface` callback for GUI integration. This is a `Ref{Function}` that receives `(message, level)` for every log event, where `level` is `:info` or `:warn`. This is used by downstream packages like Omniscape to integrate Circuitscape's logging into their own UI.