Circuitscape in Julia: High Performance Connectivity Modelling to Support Conservation Decisions

Ranjan Anantharaman[1], Kimberly Hall[2], Viral B. Shah[3], and Alan Edelman[1]

[1]Massachusetts Institute of Technology
[2]The Nature Conservancy
[3]Julia Computing Inc.

ABSTRACT

Connectivity across landscapes influences a wide range of conservation-relevant ecological processes, including species movements, gene flow, and the spread of wildfire, pests, and diseases. The computational demands of the next generation of connectivity models and the availability of increasingly fine grained remote sensing data drive the need for faster software for connectivity modelling. To address this, we upgraded the widely-used Circuitscape connectivity package to the Julia programming language. The Julia package, Circuitscape.jl, can now solve much larger problems up to an order of magnitude faster, with improved solvers and parallel computing features. We demonstrate scaling up to problems of size 437 million grid cells, with speedups of up to 1800% over the previous version. These improvements increase the pace of interaction between scientists and key stakeholders, facilitating faster policy decisions.

Keywords

landscape ecology , computational ecology, julia programming language

1. Introduction

Connectivity models provide important insights into ecological processes that involve variation in movement or flow patterns across heterogeneous environments [Crooks and Sanjayan, 2006]. In applied conservation, connectivity maps are incorporated into a wide range of resource evaluations and risk assessments, informing decisions on how to sustain population dynamics and genetic diversity in plant and animal populations [Kareiva and Wennergren, 1995], how to most effectively prevent infectious disease spread or reduce risks from wildfire [Gray and Dickson, 2016], and choices of where to invest in land protection or restoration to help support species range shifts under a changing climate [Heller and Zavaleta, 2009, Littlefield et al., 2017, Keeley et al., 2017].
A common requirement for modeling connectivity is a gridded depiction of the landscape in which values for each cell represent some relative value of "resistance" to movement. These resistance grids are developed through several different methods, often involving iterative processes for categorizing resistance weights for different types of barriers based on expert opinion and information on species' life histories and movement behaviors [Spear et al., 2010, Zeller et al., 2012]. This grid can then be abstracted as a graph [Urban and Keitt, 2001], providing a way to quantify ecological distance measures via graph-theoretic metrics.
The range of mathematical approaches and software tools used for modeling connectivity reflect differences in theoretical approaches, and in the underlying assumptions about how movement proceeds. The classical isolation by distance model (IBD) posits that the least cost distance across the landscape graph acts as a good proxy for ecological distance [Wright, 1943]. Tools based on this approach typically identify a single "best" route between focal locations, an important result, but one that is highly dependent upon the choice of start and end points, and has limited application if one's goal is to compare potential options for restoration or protection across a landscape with multiple habitat patches and pathways. As reviewed in [Dickson et al., 2019], from 2006-08, three seminal papers by the late Brad McRae built upon earlier work by [Doyle and Snell, 1984] to demonstrate that isolation by resistance (IBR)[McRae, 2006], which operationalizes the potential for genes and individuals to follow "random walks" across multiple pathways in the same way that electrical current can spread across multiple resistors, can provide an effective tool for considering connectivity potential across landscapes. These insights and McRae's interest in informing conservation applications inspired the Circuitscape software package [McRae et al., 2008], which calculates effective resistance and "current flow," a measure of net movement probability, across heterogeneous landscapes [Dickson et al., 2019]. This approach has evolved into a very flexible set of tools that allow users to vary the resistance grid and identification of what can be connected, enabling modelers to address a wide range of questions related to structural (ability of a landscape to support movement) and functional (modeling that is tailored to species-specific traits) connectivity. Over the past decade Circuitscape has emerged as the most cited landscape connectivity tool in the world [Dickson et al., 2019]. In the last year alone, Circuitscape has been cited 129 times, and in the past three years, 480 times. Dickson et al. (2019) reviews 277 applications of the software, in fields ranging from gene flow and animal movement to fire, water, and disease spread, and also describes how outputs are being used to inform conservation decisions.

2. Circuitscape in Julia

The goal of increasing the computational power of Circuitscape has been addressed multiple times by developers Brad McRae and Viral Shah. McRae's first version was written in Java, before be-

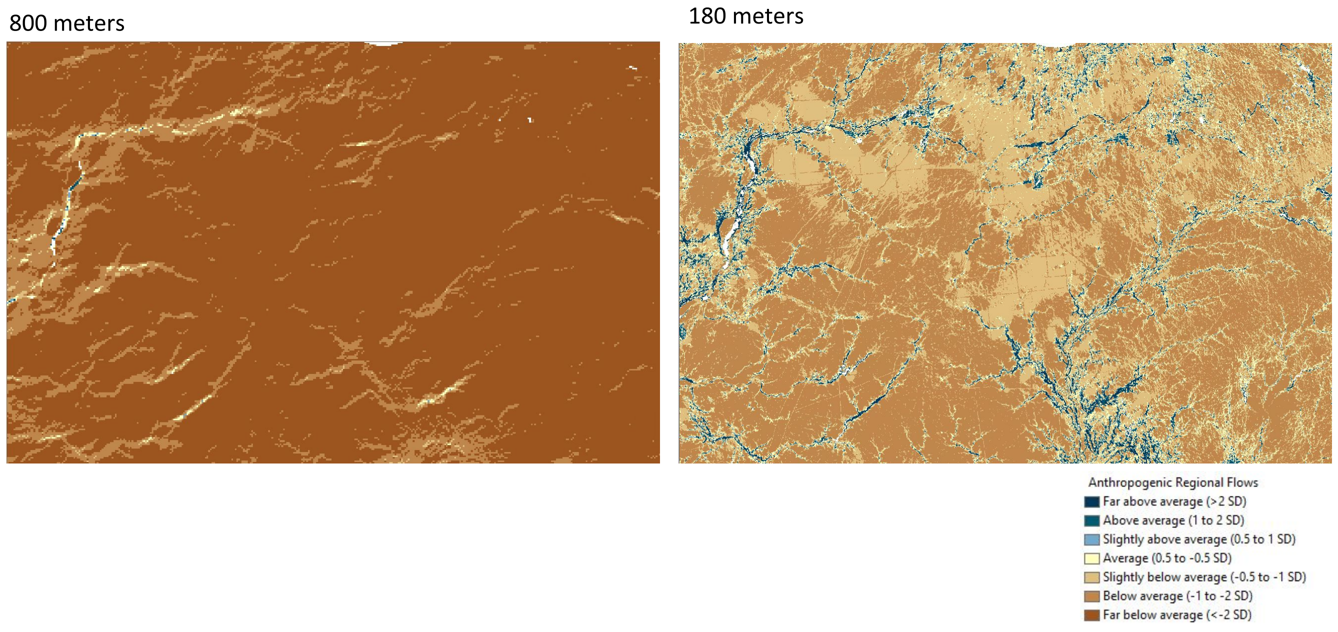
800 meters
180 meters
Anthropogenic Regional Flows
Far above average (>2 SD)
Above average (1 to 2 SD)
Slightly above average (0.5 to 1 SD)
Average (0.5 to -0.5 SD)
Slightly below average (-0.5 to -1 SD)
Below average (-1 to -2 SD)
Far below average (<-2 SD)

Fig. 1. **An illustration of the impact of coarsening the resistance grid to address computational limitations.** For this agricultural area along the border of Illinois and Indiana, USA, we show current flow derived from a resistance grid coarsened to 900m (left), compared to a higher-resolution map (180m cells, right). Both resistance grids were derived from the 2011 NLCD (30m), and run with a "wall-to-wall" application of Circuitscape to evaluate landscape structural connectivity. All "natural" terrestrial land cover types in the NLCD were assigned low resistance values, with pasture, roads, row crops, etc., assigned higher resistances. The 900m resolution map misses many narrow bands of natural land cover with high current flow (dark blue) indicating high connectivity value along streams and rivers. Analyses and maps by Melissa Clark, to support work in [Baldwin et al., 2018].

ing ported to MATLAB, to improve ease of development. Then, in collaboration with Tanmay Mohapatra, it was translated to Python for flexible scripting, platform independence and release under an open source license. This package makes heavy use of the numpy [Van Der Walt et al., 2011], scipy [Jones et al., 2014] and pyamg [Bell et al., 2015] numerical libraries and solves problems of up to tens of millions of nodes, but connectivity analysis increasingly finer resolution databases often scaled to hundreds of millions of nodes and dozens of focal node pairs, a workload size well beyond its capacity. Circuitscape needed to be upgraded to support the newer demands of user community.

To address these demands, McRae and Shah collaborated on a project called GFLOW [Leonard et al., 2017], a software tool for conducting circuit-theory based analyses on supercomputers. Written in the C programmming language and making use of solver libraries such as PetSc [Balay et al., 2004] and Boomer-AMG [Yang et al., 2002], GFLOW yields state of the art performance on large clusters and supercomputers. However, this performance comes at the cost of accessibility (does not work on the Windows operating system, which the majority of Circuitscape users use), composability (ability to integrate with other software libraries) as well as a tangible difficulty in shipping binaries because of their complicated build processes. For many practitioners and researchers, these constraints are a key barrier, as they prevent GFLOW from being supported on the Windows operating system, and limit integration into a decision support tool or workflow with other tools.

As the maintainers of the Circuitscape, we saw the need for an open-source implementation that is easy to maintain, is high performance, is accessible to our user base and works well on every platform. We decided to use the Julia programming language [Bezanson et al., 2017]. Julia is an open source dynamic programming language which combines the readability of scripting languages such as R or Python, with the performance of a statically compiled language such as C or Fortran. With relatively little development effort via a straight reimplementation of the algorithm, Julia allowed us to not only significantly improve the package's computational capacity, but provide new user-facing features for free. For example, Julia's first class sparse matrix library and factorization support allowed us to support multiple solvers. Its modern Just in Time (JIT) compiler enables programmers to write generic code, and then generate specialized code for desired precision, index types and hardware platform. This makes maintenance of the code base simple. Our upgrade, "Circuitscape.jl", is a registered Julia package and is already being used by the community to solve the next generation of connectivity problems.

The rest of this paper is organized as follows: we present an overview of the algorithm in Section 3, before describing the numerical methods and software packages in Section 4. Section 5 then briefly describe the new features introduced in the upgrade and

Algorithm 1: Circuitscape - Pairwise Mode

```
Result: Current (probability) maps $C$, nodal voltages $V$, pairwise
        resistances
1  Read input raster grid ;
2  Read focal nodes ;
3  Construct undirected, weighted graph connecting neighboring cells;
4  Compute graph Laplacian $G$ ;
5  for all pairs of focal nodes do
6      Set up $I$;
7      Solve linear system $GV = I$ ;
8      Compute effective resistance $r$;
9      Write nodal voltages $V$ to disk;
10     Compute nodal currents from $I$ and write current map C to disk
11 end
```

then present benchmarks both on synthetic problems as well as real user data. We then round off this work with a discussion on how speed improvements enable increased collaboration between computer scientists and ecologists enable improved collaboration between ecologists and conservation managers and other stakeholders.

3. The Algorithm

Circuitscape takes as input a raster grid, and a list of points which are potential starting and ending points for animal migration. Selecting these points (or focal nodes) on the landscape is often context dependent. For example, they could be two different national parks, or points on the landscape which share important climate properties. The raster grid is a spatial discretization of a heterogeneous landscape where each cell is assigned a value. Each value represents varying qualities of habitat, dispersal routes, or movement barriers. These "resistance" values are often problem specific, and are either empirically derived based on movement or genetic data, or derived via expert opinion. The following content has been summarized in Figure 2. This grid can then be represented as a graph, with nodes representing grid cells and edge weights proportional to the movement probabilities or number of migrants exchanged. Other inputs include a list of focal points, which represent points on the landscape of interest to the practitioner. These include habitat patches, protected areas, or populations to connect. These can be specified either through a raster grid with numerical labels for each focal point, or a text file with a list of coordinates. Once the graph is constructed, we then compute its laplacian, which is an alternate representation of the landscape. We then solve for Ohm's law using the computed graph laplacian as the conductance matrix and the node locations of the focal points as current sources and sinks. On solving the system, we obtain a list of nodal voltages, which are then used to compute branch currents, which represent movement probabilities along branches of the graph. The ecological significance of these various quantities are summarized in related works [McRae et al., 2008]. The branch currents are then accumulated on each node as nodal currents, which are then written to disk as a raster ASCII grid, for easy import into a geographic information system (GIS).

4. Numerical Methods

The vast majority of the computation in Circuitscape is applying Ohm's law and Kirchoff's law over very large resistive grids, and solving a large sparse linear system:

$$GV = I$$

where G is the graph laplacian representation of the landscape is stored as a large sparse matrix, I are the current sources, and V are the nodal voltages. We provide two solver options to the user: one based on the choleksy factorization and a preconditioned iterative method. The Cholesky factorization [Higham, 2009] is efficient for applications with smaller study areas and a large number of focal nodes. The sparse matrix is factored once and the solution to multiple pairs is computed via back substitution. Since the cost for backsubstitution is polynomially smaller than the cost of factorization, problems with a large number of focal pairs scale efficiently. The solver that scales to large problems is a preconditioned conjugate gradient (PCG) [Trefethen and Bau III, 1997], with an algebraic multigrid preconditioner (AMG) [Vaněk et al., 1996]. We leverage Julia implementations of these methods from open source packages IterativeSolvers.jl and AlgebraicMultigrid.jl.

4.1 Preconditioner Implementation

Multigrid preconditioners are often used in problems solved across large spatial domains. In general, they take the original grid and generate a hierarchy of coarser grids, via predefined or algebraically derived restriction and interpolation operators. Large problems are solved by restricting quantities down to the coarsest level, obtaining a fast solution, and then interpolating the solution back to the original size. This procedure is often referred to as a "V" cycle. The efficacy of a good multigrid preconditioner is to with obtaining good restriction and interpolation operators. Usually, these operators are derived from the structure of the underlying problem. However, one can also estimate these operators algebraically, from the numerical values at each grid cell. This procedure is often referred to as Algebraic Multigrid (AMG). There are several variants of AMG, which are summarized in [Stüben, 2001]. We use a variant called the Smoothed Aggregation AMG, which is known to work well for solving matrices generated by elliptic partial differential equations, such as Laplace's equation. Laplacian matrices are structurally identical to the laplacians of planar graphs that are generated in Circuitscape.

5. New Features

5.1 New Solver Mode

Circuitscape now supports a new solver based on the Cholesky factorization of the graph laplacian representation of the landscape. We use the SuperNodal sparse cholesky factorization implemented in the CHOLMOD library [Chen et al., 2008]. The Julia programming language has first class sparse matrix library and matrix factorization support, which makes it easy to use this solver with a single function call.This solver mode also enables new approaches that divide the landscape into tiles and process them independently [Pelletier et al., 2014]or analyses that move sliding windows across the landscape [McRae et al., 2016], both of which yield fine scale and localized results. The obvious limitation of this solver, as is of most sparse direct solvers, is that they work very well for relatively small matrices and run out of memory for larger ones. This exponential growth in memory is a by product of the factorization itself, as factors are often less sparse than the original matrix.

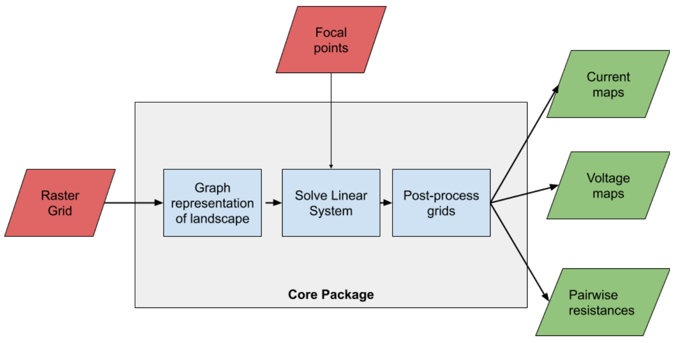

Fig. 2. **Stages of computation and inputs/outputs**. The input raster grid to is usually assembled and produced using a GIS software package, and the output current maps are often exported to a GIS software package for postprocessing.

5.2 Generic Programming

Julia supports the development of generic code while relying on the compiler to generate specialized code for different platforms and precision required. Generic programs are powerful tools for our users, and can be made to adapt to their requirements. For example, the python package had a 32-bit integer type for indexing matrices hardcoded throughout the package, which limited the size of the computation the package ran. This led to crashes when running simulations for hundreds of millions of nodes. One such dataset studying the Mojave desert tortoise [Gray and Chang, 2019] used a resistance surface of 437 million pixels. This crashed the old version of Circuitscape, while the upgrade ran smoothly to completion. Circuitscape.jl defaults to a 32-bit integer type for cache efficiency, but provide users with a runtime flag to switch to 64-bit integer to index sparse matrices on the order of hundreds of millions. Since our Julia code was written generically, and this feature came for free without any additional programming effort. Upgrading the python package to support this feature would require significant effort.

5.3 Improved Parallel Computations

Parallel computing in Circuitscape.jl is inherently faster, because the preconditioner can be serialized as a byte stream and sent over the network to other processes. This is not possible in the python code, as the python multiprocessing module internally uses the package pickle to serialize objects, and pyamg objects, implemented in C++, are not "pickle-able" [Van Rossum and Drake, 2009]. The ability to serialize native Julia objects significantly reduced the amount of effort to parallelize Circuitscape. In addition to faster parallel processing, we also extend parallelism support for more problem types in Circuitscape. In the benchmarks section, a dataset which received speed improvements from these new features. The Julia version also lets the user call Circuitscape itself in parallel. As users aim to run Circuitscape over entire countries or continents, landscape are often divided into multiple tiles [Pelletier et al., 2014]. This feature allows users to process (a batch of) these tiles by running different Circuitscape problem instances on each tile in parallel. These parallel features work on all platforms: Linux, MacOS and Windows.

6. Benchmarks

6.1 Experimental Setup

We conducted our experiments on an Intel(R) Xeon(R) Silver 4114 CPU @ 2.20GHz with 384 GB of RAM. We used Julia v1.1.0 and Circuitscape v5.5.0, and compared against Circuitscape v4.0.5

6.2 Standard Benchmarks

We benchmarked Circuitscape on standard synthetic problems. These benchmarks represent the most commonly used configuration and mode (Raster Pairwise) but with different problem sizes. These datasets can be found at `https://github.com/Circuitscape/BigTests/`. The benchmark results are summarized in Figure 3.

6.3 Benchmarks on Data from Existing Studies

We ran the Python and Julia version of Circuitscape on a dataset used to model connectivity in the Sonoran desert [Drake et al., 2017] (SONORAN). We used to all-to-one scenario [McRae and Shah, 2009], which models species migrating across the landscape from many different areas to one area. When the researcher used the old version of Circuitscape, this mode did not support parallelism, and the entire computation took over 2 days to run. Parallelism was trivial to support in the Julia version, which reduced execution time to less than 3 hours, resulting in a speedup of nearly 18x. We also benchmarked against a new range-wide model of connectivity for the endangered Mojave desert tortoise [Gray and Chang, 2019](MOJAVE). We found that the Python version crashed, but the Julia package was able to scale effectively. The benchmark results are summarized in Figure 4.

7. Discussion

The growth in popularity of circuit theory to model ecological processes has to lead to widespread adoption of the Circuitscape software package. [McRae et al., 2019]. The Circuitscape project has always evolved with the demands of the users and this upgrade to the Julia programming language seeks to drive the next generation of compute-intensive connectivity models, by shortening execution time from weeks to days and from days to hours. Users no longer have to coarsen their analysis due to computational limitations [Drake et al., 2017]. Faster results often enhance stakeholder engagement, improve the feedback loop with policy

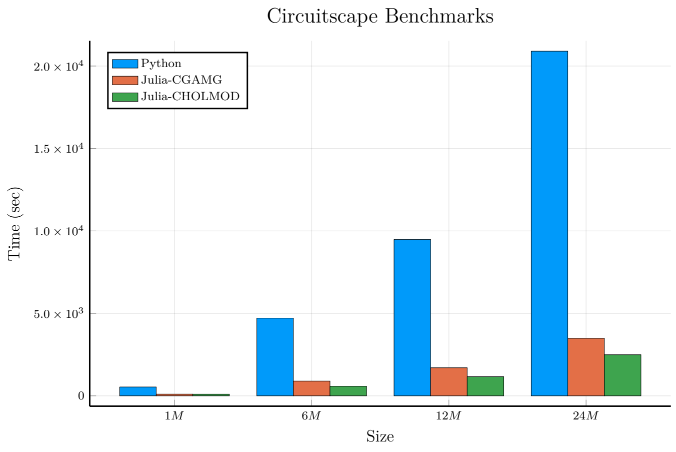

Fig. 3. **Results on the standard Circuitscape benchmark suite (smaller is better)**. The red column standards for the Julia package run under the AMG solver and green column stands for the Julia package run with the Cholesky-based CHOLMOD solver. Note that we benchmark only up to size 24M because at higher sizes the CHOLMOD solver runs out of memory and the Python version takes much, much longer. In summary, this chart demonstrates that Julia-CHOLMOD is the right choice for smaller problem sizes and Julia-CGAMG is the right choice for large problem sizes.

Benchmark	**Size**	**Number of Solves**	**Time (Python)**	**Time (Julia)**
SONORAN	3339832	6002	56.62 hrs	2.838 hrs
MOJAVE	437 million	1	Crashed	3.385 hrs

Fig. 4. **Benchmarks on data from two conservation studies**. Two metrics determine the execution time of a Circuitscape run: the size of the problem and the number of linear system solves. Our package scales desirably in both directions.

makers and result in faster turnaround time for conservation decisions. Julia is already being used by ecologists to model ecological networks [Poisot and Bélisle, 2018] and bioenergetic food webs [Delmas et al., 2019] and we see a strong role for it in other areas of computational ecology and conservation science. Its high level mathematical syntax is accessible to scientists while its speed allows them to simulate large, intricate models that capture complexity. We hope to see more ecologists and practitioners use and adopt Julia.

8. Conclusion

We present an upgrade to the Circuitscape package, which will allow researchers to analyze ecological processes over large landscapes at fine resolutions. Our upgrade in the high performance Julia programming language presents upto a 1800% improvement in computation time and the ability to solve landscapes with hundreds of millions (and potentially billions) of grid cells. Julia's sophisticated compiler allows for faster parallelism, generic programming and composability with other software packages. Circuitscape.jl is open source and is available at `https://github.com/Circuitscape/Circuitscape.jl` under the MIT license. Binaries are available on `https://circuitscape.org/downloads/`. It can also be installed using the Julia package manager by starting Julia, and entering `using Pkg; Pkg.add("Circuitscape")`.

9. Acknowledgement

The work to update Circuitscape to the Julia computing language was funded through a grant from NASA's Ecological Forecasting Program (16-ECO4CAST0018), and a grant from the Wilburforce Foundation. The authors also wish to acknowledge the Wilburforce Foundation's long-term investment in Circuitscape and related tools through several previous awards to Dr. Brad McRae. We thank all of the Circuitsape users who have shared datasets, reported issues, and provided feedback on various versions of these tools, and look forward to continued collaboration with the user community.

10. References

[Balay et al., 2004] Balay, S., Buschelman, K., Eijkhout, V., Gropp, W. D., Kaushik, D., Knepley, M. G., McInnes, L. C., Smith, B. F., and Zhang, H. (2004). Petsc users manual. Technical report, Technical Report ANL-95/11-Revision 2.1. 5, Argonne National Laboratory.

[Baldwin et al., 2018] Baldwin, R. F., Trombulak, S. C., Leonard, P. B., Noss, R. F., Hilty, J. A., Possingham, H. P., Scarlett, L., and Anderson, M. G. (2018). The future of landscape conservation. *BioScience*, 68(2):60–63.

[Bell et al., 2015] Bell, W., Olson, L., and Schroder, J. (2015). Pyamg: Algebraic multigrid solvers in python v3. 0, 2015. *URL http://www. pyamg. org. Release*, 3.

[Bezanson et al., 2017] Bezanson, J., Edelman, A., Karpinski, S., and Shah, V. B. (2017). Julia: A fresh approach to numerical computing. *SIAM review*, 59(1):65–98.

[Chen et al., 2008] Chen, Y., Davis, T. A., Hager, W. W., and Rajamanickam, S. (2008). Algorithm 887: Cholmod, supernodal sparse cholesky factorization and update/downdate. *ACM Transactions on Mathematical Software (TOMS)*, 35(3):22.

[Crooks and Sanjayan, 2006] Crooks, K. R. and Sanjayan, M. (2006). *Connectivity conservation*, volume 14. Cambridge University Press.

[Delmas et al., 2019] Delmas, E., Bideault, A., Clegg, T., and Poisot, T. (2019). Simulations of biomass dynamics and extinctions in food webs.

[Dickson et al., 2019] Dickson, B. G., Albano, C. M., Anantharaman, R., Beier, P., Fargione, J., Graves, T. A., Gray, M. E., Hall, K. R., Lawler, J. J., Leonard, P. B., et al. (2019). Circuit-theory applications to connectivity science and conservation. *Conservation Biology*, 33(2):239–249.

[Doyle and Snell, 1984] Doyle, P. G. and Snell, E. (1984). Random walks and electrical networks. carus math.

[Drake et al., 2017] Drake, J. C., Griffis-Kyle, K., and McIntyre, N. E. (2017). Using nested connectivity models to resolve management conflicts of isolated water networks in the sonoran desert. *Ecosphere*, 8(1).

[Gray and Chang, 2019] Gray, M.E., B. D. K. N. T. E. and Chang, T. (2019). A range-wide model of contemporary, omnidirectional connectivity for the threatened mojave desert tortoise. (in review). *Ecosphere*.

[Gray and Dickson, 2016] Gray, M. E. and Dickson, B. G. (2016). Applying fire connectivity and centrality measures to mitigate the cheatgrass-fire cycle in the arid west, usa. *Landscape ecology*, 31(8):1681–1696.

[Heller and Zavaleta, 2009] Heller, N. E. and Zavaleta, E. S. (2009). Biodiversity management in the face of climate change: a review of 22 years of recommendations. *Biological conservation*, 142(1):14–32.

[Higham, 2009] Higham, N. J. (2009). Cholesky factorization. *Wiley Interdisciplinary Reviews: Computational Statistics*, 1(2):251–254.

[Jones et al., 2014] Jones, E., Oliphant, T., and Peterson, P. (2014). {SciPy}: Open source scientific tools for {Python}.

[Kareiva and Wennergren, 1995] Kareiva, P. and Wennergren, U. (1995). Connecting landscape patterns to ecosystem and population processes. *Nature*, 373(6512):299.

[Keeley et al., 2017] Keeley, A. T., Beier, P., Keeley, B. W., and Fagan, M. E. (2017). Habitat suitability is a poor proxy for landscape connectivity during dispersal and mating movements. *Landscape and Urban Planning*, 161:90–102.

[Leonard et al., 2017] Leonard, P. B., Duffy, E. B., Baldwin, R. F., McRae, B. H., Shah, V. B., and Mohapatra, T. K. (2017). gflow: software for modelling circuit theory-based connectivity at any scale. *Methods in Ecology and Evolution*, 8(4):519–526.

[Littlefield et al., 2017] Littlefield, C. E., McRae, B. H., Michalak, J. L., Lawler, J. J., and Carroll, C. (2017). Connecting today's climates to future climate analogs to facilitate movement of species under climate change. *Conservation biology*, 31(6):1397–1408.

[McRae et al., 2016] McRae, B., Popper, K., Jones, A., Schindel, M., Buttrick, S., Hall, K., Unnasch, R., and Platt, J. (2016). Conserving nature's stage: Mapping omnidirectional connectivity for resilient terrestrial landscapes in the pacific northwest. *The Nature Conservancy, Portland, Oregon*.

[McRae et al., 2019] McRae, B., Shah, V., Mohapatra, T., and Anantharaman, R. (2019). https://circuitscape.org/applications/.

[McRae, 2006] McRae, B. H. (2006). Isolation by resistance. *Evolution*, 60(8):1551–1561.

[McRae et al., 2008] McRae, B. H., Dickson, B. G., Keitt, T. H., and Shah, V. B. (2008). Using circuit theory to model connectivity in ecology, evolution, and conservation. *Ecology*, 89(10):2712–2724.

[McRae and Shah, 2009] McRae, B. H. and Shah, V. B. (2009). Circuitscape user's guide. *The University of California, Santa Barbara*.

[Pelletier et al., 2014] Pelletier, D., Clark, M., Anderson, M. G., Rayfield, B., Wulder, M. A., and Cardille, J. A. (2014). Applying circuit theory for corridor expansion and management at regional scales: tiling, pinch points, and omnidirectional connectivity. *PLoS One*, 9(1):e84135.

[Poisot and Bélisle, 2018] Poisot, T. and Bélisle, Z. (2018). EcologicalNetworks.jl - analysing ecological networks.

[Spear et al., 2010] Spear, S. F., Balkenhol, N., FORTIN, M.-J., McRae, B. H., and Scribner, K. (2010). Use of resistance surfaces for landscape genetic studies: considerations for parameterization and analysis. *Molecular ecology*, 19(17):3576–3591.

[Stüben, 2001] Stüben, K. (2001). A review of algebraic multigrid. In *Numerical Analysis: Historical Developments in the 20th Century*, pages 331–359. Elsevier.

[Trefethen and Bau III, 1997] Trefethen, L. N. and Bau III, D. (1997). *Numerical linear algebra*, volume 50. Siam.

[Urban and Keitt, 2001] Urban, D. and Keitt, T. (2001). Landscape connectivity: a graph-theoretic perspective. *Ecology*, 82(5):1205–1218.

[Van Der Walt et al., 2011] Van Der Walt, S., Colbert, S. C., and Varoquaux, G. (2011). The numpy array: a structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22.

[Van Rossum and Drake, 2009] Van Rossum, G. and Drake, F. L. (2009). Python 2.7.16 reference manual.

[Vaněk et al., 1996] Vaněk, P., Mandel, J., and Brezina, M. (1996). Algebraic multigrid by smoothed aggregation for second and fourth order elliptic problems. *Computing*, 56(3):179–196.

[Wright, 1943] Wright, S. (1943). Isolation by distance. *Genetics*, 28(2):114.

[Yang et al., 2002] Yang, U. M. et al. (2002). Boomeramg: a parallel algebraic multigrid solver and preconditioner. *Applied Numerical Mathematics*, 41(1):155–177.

[Zeller et al., 2012] Zeller, K. A., McGarigal, K., and Whiteley, A. R. (2012). Estimating landscape resistance to movement: a review. *Landscape ecology*, 27(6):777–797.