Circuitscape.jl: Connectivity Modelling meets Computer Science

Ranjan Anantharaman, Viral Shah, Kimberly Hall

MIT, Julia Computing, The Nature Conservancy

May 22, 2019

What is Circuitscape?

- Circuitscape borrows algorithms from electronic circuit theory to estimate **connectivity** in heterogeneous landscapes.
- Applications in **movement ecology**, **climate connectivity**, **epidemiology**, and many other areas. (`circuitscape.org/applications/`)

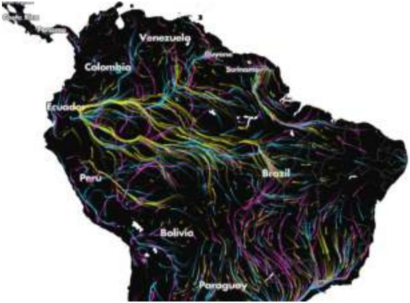


Figure 1: Niche models and Circuitscape connectivity [Lawler et al 2013]

Why does it work?

Based on two fundamental ideas:

- Landscapes are large (but simple) weighted graphs [Urban and Keitt 2001]

Why does it work?

Based on two fundamental ideas:

- Landscapes are large (but simple) weighted graphs [Urban and Keitt 2001]
- Effective resistance is a measure of ecological distance [*Isolation by Resistance*, McRae 2006]

How does it work?

Main stages of simulation:

1. Construct graph representation of landscape. Take $m \times n$ raster map, generate $mn \times mn$ *sparse* matrix.

How does it work?

Main stages of simulation:

1. Construct graph representation of landscape. Take $m \times n$ raster map, generate $mn \times mn$ *sparse* matrix.
2. Compute its graph laplacian G.

How does it work?

Main stages of simulation:

1. Construct graph representation of landscape. Take $m \times n$ raster map, generate $mn \times mn$ *sparse* matrix.
2. Compute its graph laplacian G.
3. Set input sources I (comes from focal node file).

How does it work?

Main stages of simulation:

1. Construct graph representation of landscape. Take $m \times n$ raster map, generate $mn \times mn$ *sparse* matrix.
2. Compute its graph laplacian G.
3. Set input sources I (comes from focal node file).
4. Solve Ohm's Law:
$$GV = I$$
   Need to solve a large sparse linear system.

Numerics - Direct Methods

- Direct methods: Construct a sparse cholesky factorization $G = LL^*$, where L is lower triangular. [Chen, Davis et al 2008]. Works well for small problems ($\sim$ max 12M)

Numerics - Direct Methods

- Direct methods: Construct a sparse cholesky factorization $G = LL^*$, where L is lower triangular. [Chen, Davis et al 2008]. Works well for small problems ($\sim$ max 12M)
- **Limitation**: At large sizes, suffers from a phenomenon called *fill-in*. Results in loss of sparsity and spike in memory consumption.

Numerics - Direct Methods

- Direct methods: Construct a sparse cholesky factorization $G = LL^*$, where L is lower triangular. [Chen, Davis et al 2008]. Works well for small problems ($\sim$ max 12M)
- **Limitation**: At large sizes, suffers from a phenomenon called *fill-in*. Results in loss of sparsity and spike in memory consumption.

When do I use this?

Small problem ($\sim$ 12M or less), a lot of pairs.

Numerics - Iterative Methods

So how do you solve large matrices ($\sim$ 100M+)?

You come up with a series of approximations, which *hopefully* converge to the solution.

Numerics - Iterative Methods

So how do you solve large matrices ($\sim$ 100M$+$)?
You come up with a series of approximations, which *hopefully* converge to the solution.

Analyze matrix properties

The matrices in Circuitscape are often referred to as **laplacian** systems: they are symmetric ($A = A^*$) and positive semi-definite (Eigenvalues ≥ 0).

Numerics - Iterative Methods

So how do you solve large matrices ($\sim$ 100M+)?
You come up with a series of approximations, which *hopefully* converge to the solution.

Analyze matrix properties

The matrices in Circuitscape are often referred to as **laplacian** systems: they are symmetric ($A = A^*$) and positive semi-definite (Eigenvalues ≥ 0).

Choose best method

The **conjugate gradient** method is generally considered the right method for this problem.

But, iterative methods only guarantee convergence in n steps!

Preconditioners

We can't afford a million iterations (or even thousands).

Preconditioners

We can't afford a million iterations (or even thousands).

To accelerate convergence, we use a preconditioner.

Preconditioners

We can't afford a million iterations (or even thousands).
To accelerate convergence, we use a preconditioner.

Algebraic Multigrid

Come up with a **hierarchy** of smaller graphs that *approximate* our landscape graph.

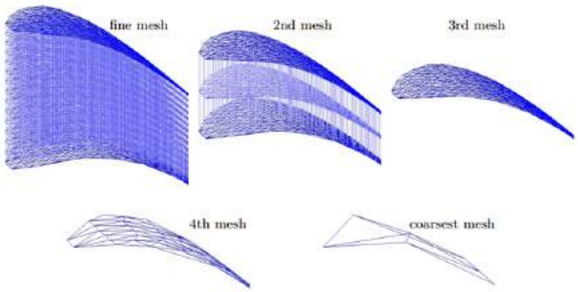


Figure 2: Source: SIAM News

Computational Challenges

- **Scalability**: Processing large datasets at fine resolution (NASA datasets, climate datasets).

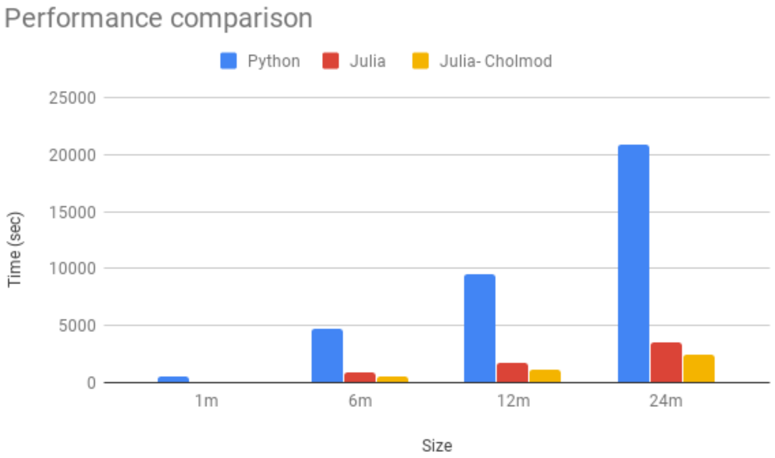
Computational Challenges

- **Scalability**: Processing large datasets at fine resolution (NASA datasets, climate datasets).
- **Extensibility**: Compose with other tools and Circuitscape extensions (Omniscape, Wall-to-Wall). Can switch out and experiment with different solvers.

Upgrade to the Julia programming language

- **Easy to use**: Interactive, feels like a scripting language like Python/R with high level syntax
- **Fast**: Designed from from the very beginning to be fast.

Benchmarks



Performance comparison
Python
Julia
Julia- Cholmod
25000
20000
15000
10000
5000
0
Time (sec)
1m
6m
12m
24m
Size

User Testimonials

"Even in CG+AMG solver mode, this problem takes only **35 hours** in Julia compared to **8 days** with original Circuitscape."
"The CHOLMOD solver mode is a **full order of magnitude faster** than the original Circuitscape, which took at least 8 days to run."
- Dr. Megan Jennings, San Diego State University.

User Testimonials

"Even in CG+AMG solver mode, this problem takes only **35 hours** in Julia compared to **8 days** with original Circuitscape."
"The CHOLMOD solver mode is a **full order of magnitude faster** than the original Circuitscape, which took at least 8 days to run."
- Dr. Megan Jennings, San Diego State University.

"In python, the problem took 36 minutes but in Julia this problem solved in under 3 minutes."
"We were able to solve massive problems (of size **437 million**) in Julia, but the older version crashed".
- Dr. Miranda Gray, Conservation Science Partners.

User Data Tests

- Joseph Drake (UMass) - All to One :
  56 hours on the old version.
  3 hours on the new version
- Open to more user case studies!

Our core contribution

- **Faster** - upto 8x faster than the previous version

Our core contribution

- **Faster** - upto 8x faster than the previous version
- **New solver** – Performs cholesky decomposition on the underlying graph

Our core contribution

- **Faster** - upto 8x faster than the previous version
- **New solver** – Performs cholesky decomposition on the underlying graph
- **Parallelism on Windows** (and Linux and MacOS) Earlier version didn't support parallelism on Windows

Our core contribution

- **Faster** - upto 8x faster than the previous version
- **New solver** – Performs cholesky decomposition on the underlying graph
- **Parallelism on Windows** (and Linux and MacOS) Earlier version didn't support parallelism on Windows
- **Single precision** support (experimental)

Our core contribution

- **Scalable Parallelism**: While solving multiple source/sink pairs in parallel, we can serialize the preconditioner and send to other processes. And, we can now call Circuitscape itself in parallel!

Our core contribution

- **Scalable Parallelism**: While solving multiple source/sink pairs in parallel, we can serialize the preconditioner and send to other processes. And, we can now call Circuitscape itself in parallel!
- **Generic Software**: We support arbitrary precision and indexing(Float32 vs Float64 computation). Rely on the compiler to generate optimal code.

Our core contribution

- **Scalable Parallelism**: While solving multiple source/sink pairs in parallel, we can serialize the preconditioner and send to other processes. And, we can now call Circuitscape itself in parallel!
- **Generic Software**: We support arbitrary precision and indexing(Float32 vs Float64 computation). Rely on the compiler to generate optimal code.
- **Composability**: We be able to try different solvers and preconditioners and have them compose well with the core simulation.

Future Work

- **Extensions**: Wall to Wall, Omniscape - important for climate connectivity.
- Resistance surface creation and improvement
- Composability with other models
- Improvements in numerics

Acknowledgements

Figure 3: In honor of Brad McRae (1966-2017)

Acknowledgements

Thank You!

- **Website**: `https://circuitscape.org`
- **Project Website**: `https://github.com/Circuitscape/Circuitscape.jl`
- **Google Group**: Long url, just google search for it.