SciencesPo

Implementing the ForceAtlas 2 graph layout algorithm in Rust

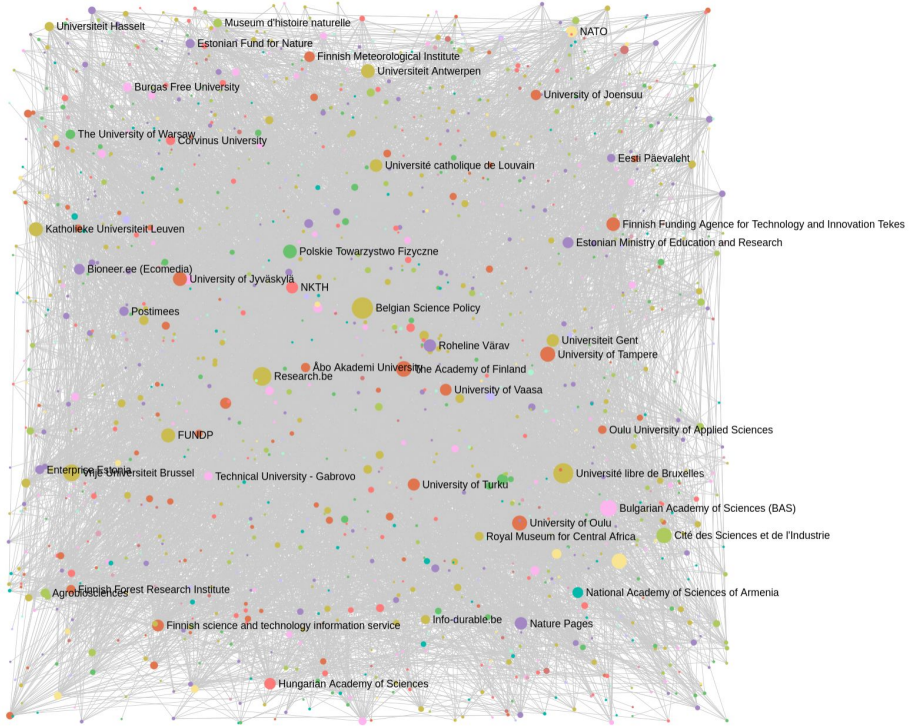
Guillaume Plique (@Yomguithereal)

Scientific Computing in Rust 2026, 09/07/2026

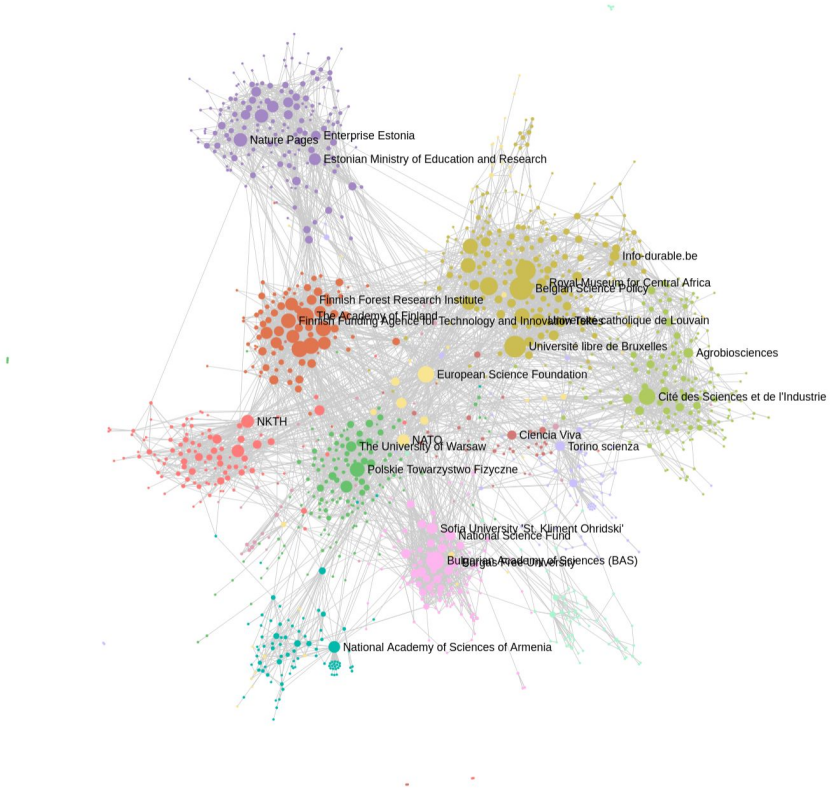
Graphs

- Nodes
- Edges (potentially weighted)
- How to project them into an arbitrary 2D space so we can:
    1. Do visual network analysis
    2. Perform a kind of dimensionality reduction

We want to go from this



To that


Nature Pages
Enterprise Estonia
Estonian Ministry of Education and Research
Info-durable.be
Royal Museum for Central Africa
Belgian Science Policy
Finnish Forest Research Institute
The Academy of Finland
Finnish Funding Agency for Technology and Innovation Tekes
Université catholique de Louvain
Université libre de Bruxelles
Agrobiosciences
European Science Foundation
Cité des Sciences et de l'Industrie
NKTH
NATO
Ciencia Viva
The University of Warsaw
Torino scienza
Polskie Towarzystwo Fizyczne
Sofia University 'St. Kliment Ohridski'
National Science Fund
Bulgarian Academy of Sciences (BAS)
National Academy of Sciences of Armenia

Graph layout algorithms

- Most popular graph layout algorithms (at least for Visual Network Analysis) are “Force-Directed”
- They are a physics simulation:
  - nodes repulse each others as bodies in space
  - linked nodes attract each other as if bound by overstretched springs
  - a gravity force pulls everything to the center

Graph layout algorithms (cont'd)

- Most known algorithm is Fruchterman-Reingold

Fruchterman, Thomas M. J.; Reingold, Edward M. (1991), "Graph Drawing by Force-Directed Placement", *Software: Practice and Experience*, **21** (11), Wiley: 1129–1164, doi:10.1002/spe.4380211102, S2CID 31468174.

- State-of-the-art algorithm is ForceAtlas2

Jacomy M, Venturini T, Heymann S, Bastian M (2014) ForceAtlas2, a Continuous Graph Layout Algorithm for Handy Network Visualization Designed for the Gephi Software. PLoS ONE 9(6): e98679. https://doi.org/10.1371/journal.pone.0098679

ForceAtlas2 canonical implementations

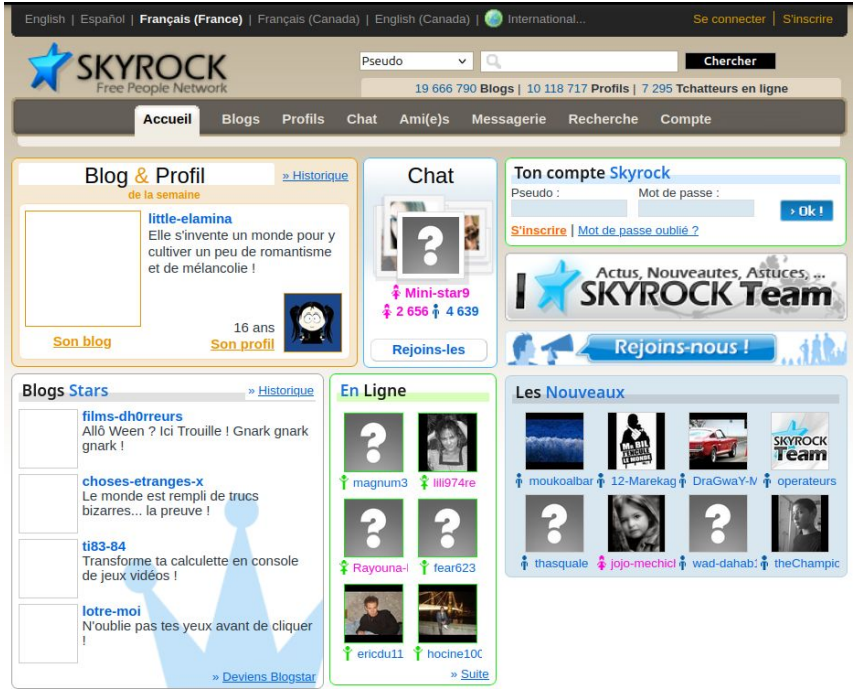
- In Java: Gephi (FA2 authors are actually behind Gephi) https://gephi.org/
- In JavaScript: graphology (mine) https://graphology.github.io/standard-library/layout-forceatlas2

They are too slow for some purposes

- JVM tweaking is pain
- JavaScript has structural issues (even if you can achieve very good single-threaded performance by relying on Typed Arrays and/or wasm):
    - floats everywhere
    - no efficient multithreading
    - Maps limited to ~24 bits addressing (lol)
    - etc.

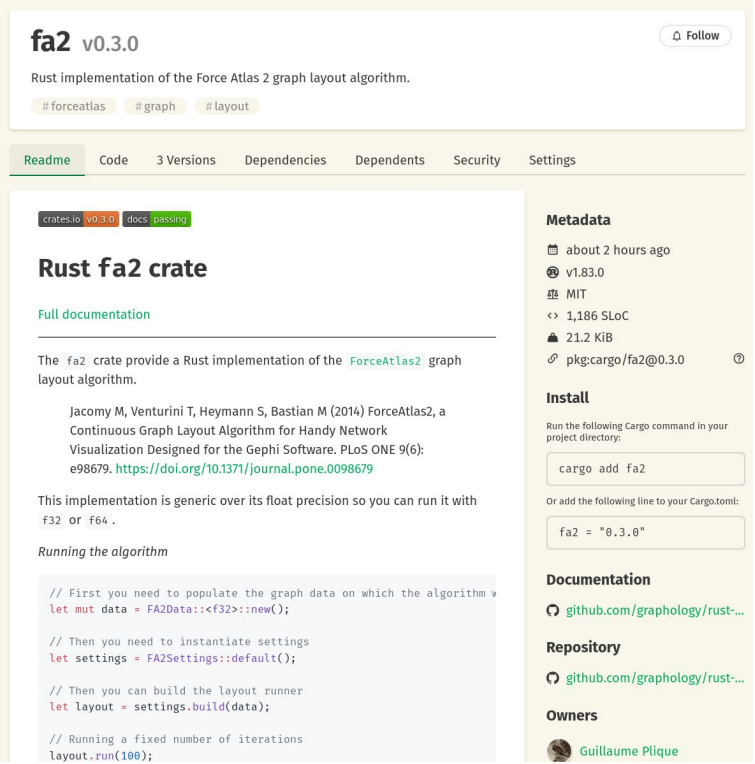
I have a big network

- the “French myspace”: skyblogs
- 12M nodes
- 400M edges



Let’s Rewrite it in Rust!

The fa2 crate



fa2 v0.3.0

Follow

Rust implementation of the Force Atlas 2 graph layout algorithm.

forceatlas # graph # layout

Readme | Code | 3 Versions | Dependencies | Dependents | Security | Settings

crates.io v0.3.0 docs passing

Rust fa2 crate

Full documentation

The `fa2` crate provide a Rust implementation of the `ForceAtlas2` graph layout algorithm.

> Jacomy M, Venturini T, Heymann S, Bastian M (2014) ForceAtlas2, a Continuous Graph Layout Algorithm for Handy Network Visualization Designed for the Gephi Software. PLoS ONE 9(6): e98679. https://doi.org/10.1371/journal.pone.0098679

This implementation is generic over its float precision so you can run it with `f32` or `f64`.

Running the algorithm

```
// First you need to populate the graph data on which the algorithm w
let mut data = FA2Data::<f32>::new();

// Then you need to instantiate settings
let settings = FA2Settings::default();

// Then you can build the layout runner
let layout = settings.build(data);

// Running a fixed number of iterations
layout.run(100);
```

Metadata

about 2 hours ago
v1.83.0
MIT
1,186 SLoC
21.2 KiB
pkg:cargo/fa2@0.3.0

Install

Run the following Cargo command in your project directory:

```
cargo add fa2
```

Or add the following line to your Cargo.toml:

```
fa2 = "0.3.0"
```

Documentation

github.com/graphology/rust-...

Repository

github.com/graphology/rust-...

Owners

Guillaume Plique

Problem: generic over float precision

- Solved using the `num-traits` crate and the following shenanigans.

```rust
use std::iter::Sum;
use std::ops::{AddAssign, DivAssign, SubAssign};

use num_traits::{Float as NumFloat, FloatConst};

1 implementation
pub trait Float:
    NumFloat + FloatConst + SubAssign + AddAssign + DivAssign + Sum + Send + Sync
{
}
impl<T: NumFloat + FloatConst + SubAssign + AddAssign + DivAssign + Sum + Send + Sync> Float for T {}
```

Problem: parallel computing

- The algorithm is usually framed in steps: repulsion, attraction, gravity & force application (with swing, traction adjustments etc.)
- When running in parallel, the algorithm must be reframed to apply each step nodewise
- This means that repulsion now needs some CSR-adjacent structure to hold the sparse adjacency matrix
- All plumbing is done with *rayon*

```rust
/// A romantic name for a variant of CSR matrix.
#[derive(Debug)]
3 implementations
pub(crate) struct NeighborhoodIndex<F: Float> {
    node_offsets: Vec<usize>,
    edge_stubs: Vec<(usize, F)>,
}
```

Problem: Amdahl's law

- Pairwise repulsion is obviously quadratic
- People use the Barnes-Hut method, using an adaptive quadtree to scale repulsion to O(n log(n))
- Repulsion through Barnes-Hut **can** be parallelized
- But Barnes-Hut tree construction **cannot** (at least not easily)
- On my graph, using 64 CPUs: 1 iteration = 10s, tree construction = 7s

Problem: SoA vs. AoS, cache lines, SIMD etc.

- Struct of Arrays is usually better for cache alignment and was proven better to hold the graph's data for FA2 in my implementation
- The case of the Barnes-Hut tree is different and is a compromise between Struct of Arrays and Array of Structs.
- Explicit SIMD usage did not yield improvements. Some auto-vectorization by the compiler is already happening in some parts of the algorithm.
- Barnes-Hut repulsion is actually branching-bound

Niche optimization & memory layout

```rust
/// A struct holding the data necessary to represent a graph that will be
/// spatialized by the FA2 algorithm.
#[derive(Debug)]
3 implementations
pub struct FA2Data<F: Float> {
    pub(crate) xs: Vec<F>,
    pub(crate) ys: Vec<F>,
    pub(crate) ms: Vec<F>,
    pub(crate) delta_xs: Vec<F>,
    pub(crate) delta_ys: Vec<F>,
    pub(crate) old_delta_xs: Vec<F>,
    pub(crate) old_delta_ys: Vec<F>,
    pub(crate) convergences: Vec<F>,
    pub(crate) edges: Vec<(usize, usize, F)>,
}
```

```rust
#[derive(Debug, PartialEq)]
2 implementations
enum RegionKind<F: Float> {
    Empty,
    Leaf {
        node: usize,
    },
    Internal {
        first_child: usize,
        mass: F,
        mass_center_x: F,
        mass_center_y: F,
    },
}

#[derive(Debug, PartialEq)]
4 implementations
pub struct BarnesHutTree<F: Float> {
    kinds: Vec<RegionKind<F>>,
    center_xs: Vec<F>,
    center_ys: Vec<F>,
    sizes: Vec<F>,
    next_siblings: Vec<Option<NonZeroUsize>>,
}
```

Problem: animating the layout to debug it

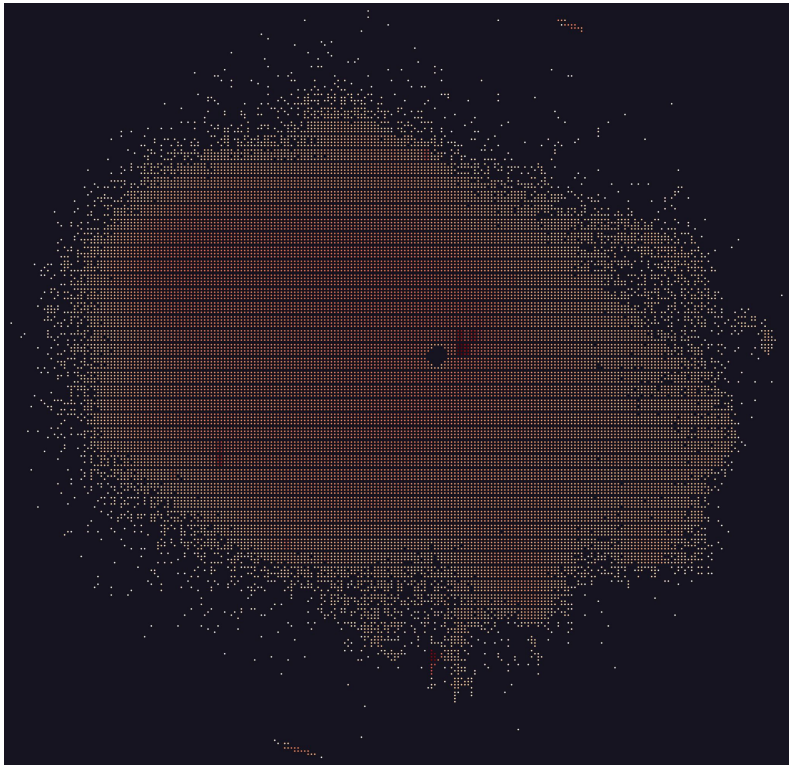
- Force-directed graph layout algorithms are **non-deterministic** and their results are dependent on nodes' starting positions
- They don't have trivial stopping conditions
- People usually rely on animating the layout to debug it and/or know when to stop it
- Demo

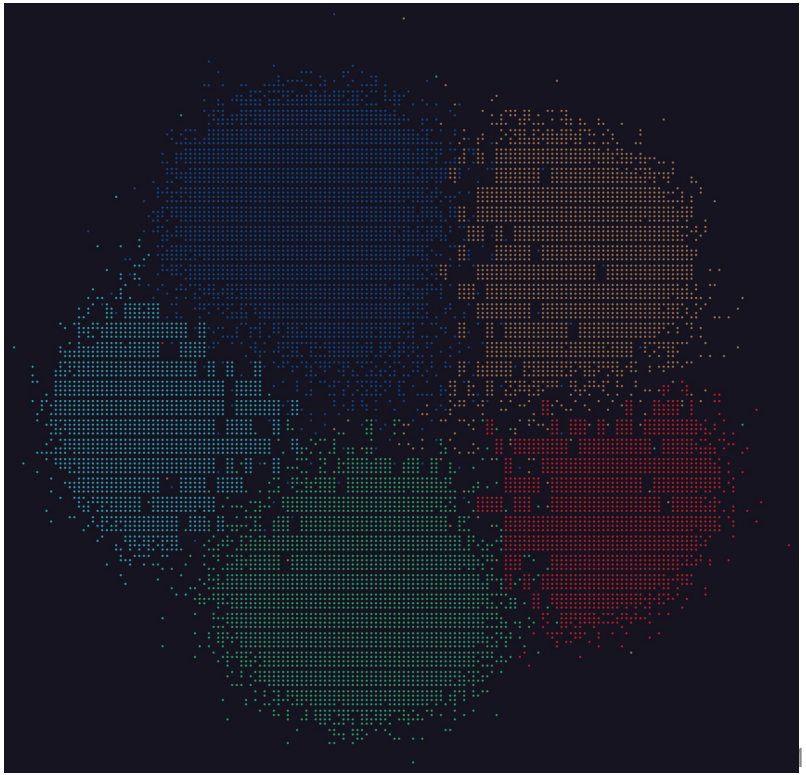
Animating the layout in the shell

- 12M nodes is a hassle to raster
- Doing so in a terminal using braille characters is actually fast enough
- We'll add a density gradient on top for better precision
- Using xan (in Rust) plotting capabilities and a bash loop (more)

```
ls dump/*.csv | sort | while read positions;
do
    xan plot x y -Q --hide-all -D or_rd $positions
done
```

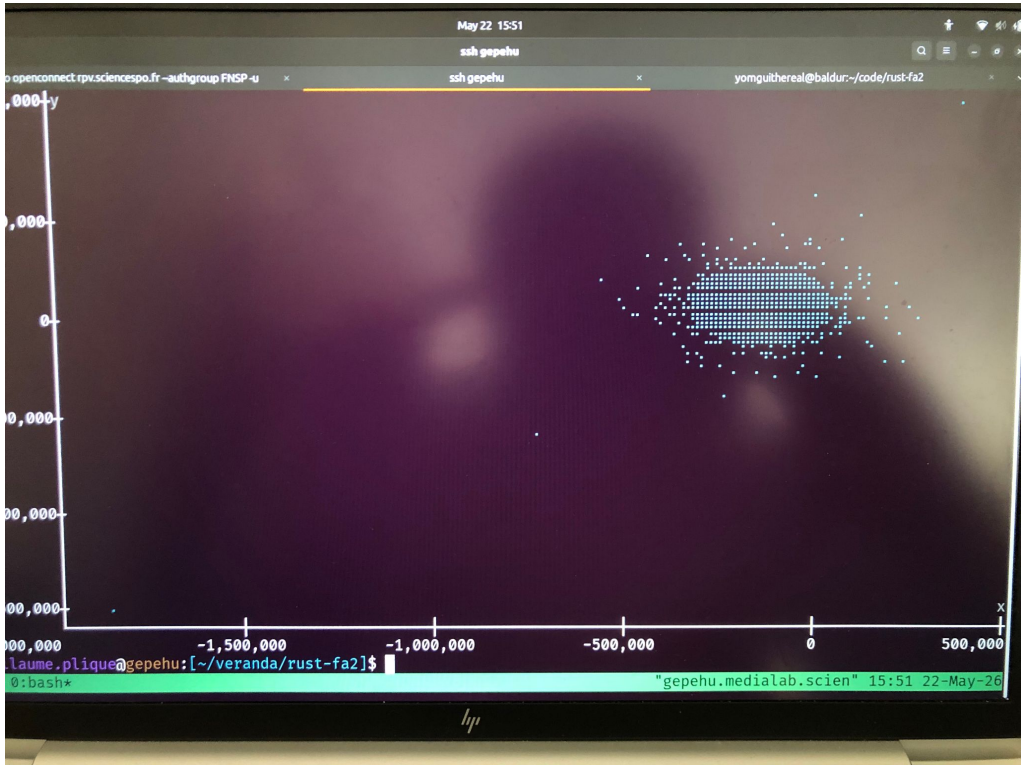
Rendering a graph in the terminal



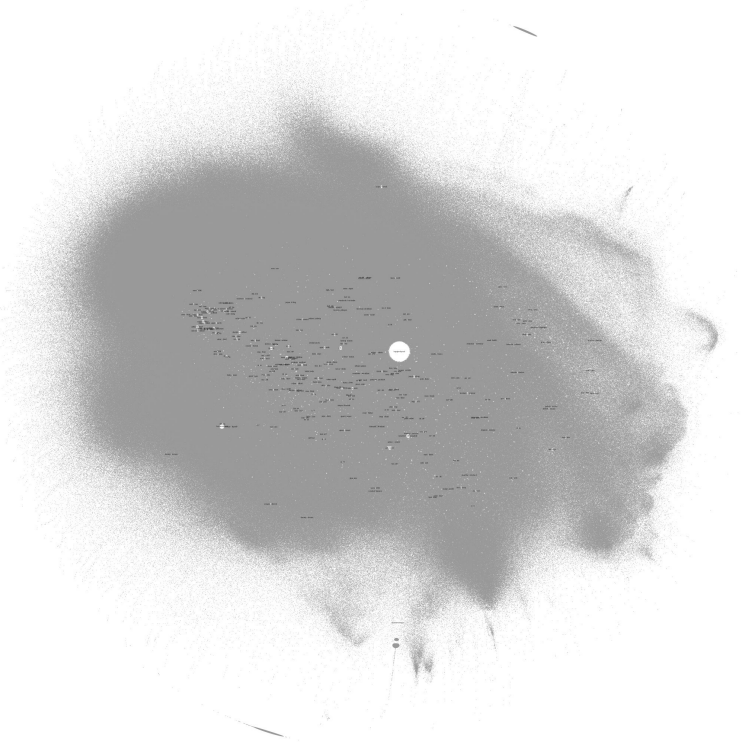


Animating the layout in the terminal

Debugging offshoots



My social network is now spatialized (albeit still ugly)!



The Future

- Parallel Barnes-Hut forest
- K-means repulsion
- N-dimensional layout (> 4 dims, which is novel)
- Bench wasm vs. JS low-level implementation

Rust is perfect to my eyes

- Very fine control over performance and memory usage
- Parallelization is so easy
- I don’t want to use C or C++
- Julia is 1-based

Thank you for listening!

Crate: https://crates.io/crates/fa2
Docs: https://docs.rs/fa2/latest/fa2/

Barnes-Hut artifacts art –>