Proceedings of the Eleventh International AAAI Conference on Web and Social Media (ICWSM 2017)

Junet: A Julia Package for Network Research

Igor Zakhlebin
SONIC Lab, Northwestern University, Evanston, IL
ANR Lab, Higher School of Economics, Moscow, Russia
izakhlebin@u.northwestern.edu

Abstract

Network science is moving at a rapid pace. However, mainstream analytic packages often fall behind: it is too difficult to implement new complex algorithms in them or it is hard to make them fast. With Junet, we address this problem by implementing a high-performance network analysis package in a high-level language. It allows users to write concise Julia code with performance on par with analogous C/C++ code and have unparalleled control over memory consumption for working with large networks.

Most network analysis packages do not allow to customize algorithms provided with them. They implement algorithms in low-level languages and interface them to high-level ones. This combination provides both speed and convenience, but also restricts users to the choices made by package developers. To implement their own algorithms, users have to dive into complicated low-level internals or stay in the high-level languages and suffer from poor performance.

To address this problem, we introduce Junet—a network analysis package written in Julia, a high-level and easy to learn language, just-in-time (JIT) compiled into machine code with execution times close to those of C (Bezanson et al. 2014). With Junet, users write code in the same language as the package and it has the same high performance as package-provided algorithms and primitives.

Usage

From the user's perspective, Junet works much like other network analysis libraries and has a similar syntax. It provides the means to both directly manipulate nodes, edges, and their attributes and to run algorithms on the whole network. A usage example is shown on Figure 1. It includes loading the network, computing its node centralities, and incorporating them into visualization. Users can run it either interactively (line by line) or as a script.

Out of the box, Junet provides such capabilities as: input and output in a number of common network storage formats; random graph generation; computing node, edge and graph statistics; manipulating node and edge attributes; community detection; visualization; and more. The choice of available algorithms is already broad and continues to grow.


```
> using Junet
> g = readgraph("edges.txt", directed=true)
> pr = pagerank(g)
> kc = kcores(g)
> g[:, :size] = 1 + pr * 1000
> g[:, :opacity] = kc / maximum(kc)
> plot(g)
```

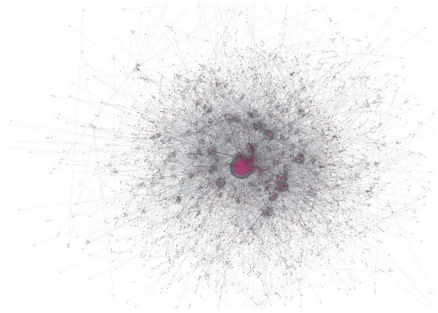

Figure 1: Example block of Julia code using Junet.

Internal Design

From the developer's perspective, Junet provides a number of distinctive features compared to the other libraries. In its development, we focus on the following three objectives.

Ease of experimentation Users should be able to easily modify and build upon provided algorithms.

Towards this goal, Junet's code is structured into loosely coupled modules that depend on a common set of operations (like node and edge addition and removal, or graph traversal), most of which are directly accessible by the users. As they learn to use the package, they simultaneously learn its internals, which helps in writing new algorithms.

Another feature provided by Julia language is support for hot-swapping most of the package's method definitions from within the users' code. Thus, users can modify behavior of Junet without actually modifying its source code or restarting the Julia interpreter.

Performance Code based on Junet should approach execution times of analogous low-level code written in C/C++.

Table 1: Comparison of network packages in terms of memory consumption and algorithm execution times. Marked with an asterisk (*) are versions using all optimizations available. Two best results in each row are highlighted.

	igraph	graph-tool	SNAP.py	SNAP.py*	Junet	Junet*	NetworkX
Memory (MB)	2,285	3,457	5,120	**1,697**	2,247	**591**	43,343
Components (s)	3.5	**3.4**	22.5	7.9	3.6	**2.8**	35.5
K-cores (s)	**6.2**	**3.2**	39.4	30.4	9.5	8.5	349.2
PageRank (s)	**22.2**	50.6	250.2	50.1	24.3	**17.3**	625.9
Clustering (s)	**22.2**	254.2	266.9	249.2	44.9	**35.2**	2804.4

The package relies on the Julia compiler to produce efficient machine code. Additionally, it exploits the fact that its code is JIT-compiled in conjunction with a multiple dispatch mechanism built into Julia to optimize the code on a level of entire functions. For example, all checks for graph directionality happen at compile time, and pieces of code that depend on them are automatically swapped for more efficient, specialized versions. Furthermore, many operations like graph reversal do not change or copy the underlying data and instead create lightweight views of it, which makes them very cheap to use.

Memory efficiency Researchers often develop the algorithms on their personal computers that are constrained in RAM, so it's important to utilize memory as efficiently as possible. Alternatively, when used on big RAM machines, the package should support working with as many and as large networks as possible.

For that, Junet provides efficient container types for node and edge attributes and automatically chooses which ones to use depending on the user's code. It also allows using non-standard integer types for node and edge identifiers. Switching on a 64-bit machine to unsigned 32-bit integers reduces memory consumption twofold, while still allowing to work with any network under 4Bn nodes and edges. Forsaking the ability to assign edge attributes brings consumption even further down, for a total of about 4x reduction.

Evaluation

We compare Junet with 4 other state-of-the-art packages for network analysis designed for high-level languages. For R, it is igraph (Csardi and Nepusz 2006), and for Python it is graph-tool (Peixoto 2014), SNAP.py (Leskovec and Sosič 2016), and NetworkX (Schult and Swart 2008). All of them except NetworkX use low-level libraries to back up their operation and the latter is entirely Python code.

Packages need to load a directed network of connections between LiveJournal blogs (retrieved from SNAP repository) comprising over 4.8M nodes and 68.9M edges. The amount of RAM used to represent this network is measured along with the time it took to execute four common algorithms: connected components, k-core decomposition, PageRank, and computing the global clustering coefficient. All benchmarks were run on a large-memory Linux server with Intel Xeon E5-2698B processor clocked at 2.0GHz. Results are shown in Table 1.

On average, Junet is one of the fastest packages, performing on par with igraph and graph-tool. It is worth reiterating that unlike them, Junet is pure high-level Julia code. The only other high-level library, NetworkX, performed on average 1 to 2 orders of magnitude worse than the rest.

Junet is also one of the best in terms of memory consumption. Like SNAP.py, it supports different representations to accommodate networks of different size and complexity, but tends to consume 2 to 3 times less memory.

Availability

Junet is open software and is distributed under MIT license. Its source code can be found at https://github.com/inguar/Junet.jl. As it is written entirely in Julia, it should work on each of Julia's supported platforms. Currently, these include Linux, macOS, and Windows.

During the demonstration session, we will overview the library and offer participants to use it for analysis of large networks either on their own computers or on specially prepared and configured cloud instances.

Acknowledgments

This work is partially supported by Russian Academic Excellence Project "5-100" and Microsoft Azure for Research Award. Author thanks Aleksandr Semenov, Petr Ermakov, Aaron Schecter, Noshir Contractor, and other SONIC Lab members for their feedback and support.

References

Bezanson, J.; Edelman, A.; Karpinski, S.; and Shah, V. B. 2014. Julia: A fresh approach to numerical computing. *arXiv:1411.1607*.

Csardi, G., and Nepusz, T. 2006. The igraph software package for complex network research. *InterJournal, Complex Systems* 1695(5).

Leskovec, J., and Sosič, R. 2016. Snap: A general-purpose network analysis and graph-mining library. *ACM Transactions on Intelligent Systems and Technology* 8(1).

Peixoto, T. P. 2014. The graph-tool python library. *figshare*.

Schult, D. A., and Swart, P. 2008. Exploring network structure, dynamics, and function using NetworkX. 2008.