Bioinformatics, 33(5), 2017, 770–772
doi: 10.1093/bioinformatics/btw682
Advance Access Publication Date: 5 December 2016
Applications Note

OXFORD

Systems biology

PyBoolNet: a python package for the generation, analysis and visualization of boolean networks

Hannes Klarner*, Adam Streck and Heike Siebert

Institut für Mathematik, Freie Universität Berlin, 14195 Berlin, Germany

*To whom correspondence should be addressed.

Associate Editor: Cenk Sahinalp

Received on June 21, 2016; revised on October 12, 2016; editorial decision on October 24, 2016; accepted on October 25, 2016

Abstract

Motivation: The goal of this project is to provide a simple interface to working with Boolean networks. Emphasis is put on easy access to a large number of common tasks including the generation and manipulation of networks, attractor and basin computation, model checking and trap space computation, execution of established graph algorithms as well as graph drawing and layouts.

Results: PyBoolNet is a Python package for working with Boolean networks that supports simple access to model checking via NuSMV, standard graph algorithms via NetworkX and visualization via dot. In addition, state of the art attractor computation exploiting Potassco ASP is implemented. The package is function-based and uses only native Python and NetworkX data types.

Availability and Implementation: https://github.com/hklarner/PyBoolNet

Contact: hannes.klarner@fu-berlin.de

1 Background and motivation

Boolean networks offer an intuitive approach to simulate the dynamics of interaction networks. In cell biology these are usually gene regulatory or signal transduction networks. Each biological component is modelled by a Boolean variable that can only switch between two states, *true* and *false*. Depending on the component, these states represent whether a protein is present at high or low concentration levels, a gene is being transcribed at or above its base rate, a signalling molecule is phosphorylated or not, and so on. The conditions under which variables switch on and off are specified by Boolean expressions that involve other variables of the network. Together with a transition relation, which determines how many and which components are allowed to change during a state transition, Boolean networks can be used to predict transient and long term activity profiles of the involved components. Although the restriction to only two states is arguably an over-simplification for many processes, Boolean networks have frequently been shown to be fruitful to predict drug targets, phenotypes, novel interactions, etc. For a detailed introduction and further reading see Naldi *et al.*, (2015).

Working with Boolean networks requires access to standard functionalities for construction, analysis and visualization. Model definition from scratch should be convenient and intuitive, allowing for easy manipulation. The analysis then raises challenging computational problems, since the state space is exponential in the number of Boolean variables. Apart from sophisticated graph algorithms, so-called *symbolic model checking* algorithms efficiently provide answers to reachability and stability queries, see Baier and Katoen (2008). Visualization is then a significant help in understanding the results. In addition, it is desirable to integrate this within a high level language that allows rapid prototyping and testing of new algorithms and work-flows before investing energy in highly optimized implementations.

Various tools that address aspects of these requirements exist. GINsim is a Java application for the construction and analysis of multi-valued logical models, see Naldi *et al.*, (2009). While offering a large number of sophisticated analysis options, access to the large palette of more general graph algorithms or model checking is not directly integrated. Also, GINsim is not intended to be a scripting interface for custom work-flows. BoolNet is a library for R that

implements several published algorithms for the reconstruction of Boolean networks from time series data, see Maucher *et al.*, (2011), the binarization of expression profiles, see Hopfensitz *et al.*, (2012), the computation of stable states and limit cycles and the search for complex attractors as well a random network generator, see Müssel *et al.*, (2010). It does not support model checking and the intricacies of the various R data types can sometimes be hindering for prototyping. BooleanNet is a Python package for the generation and simulation of Boolean networks that includes the option of converting a network into a system of piece-wise linear differential equations, see Albert *et al.*, (2008). It is designed as a pure simulator rather than for access to graph algorithms, model checking or the implementation of new functionalities.

The motivation for developing PyBoolNet was to offer a simple, well-documented interface to manipulating Boolean networks, model checking, standard graph algorithms, visualization and state of the art attractor detection. We believe that these goals distinguish it from existing tools and packages and contribute to the study of Boolean networks.

2 Results and functionality

The package is compatible with Python 2.x and 3.x. It is purely function based and uses only native Python and NetworkX data types (see below). All functions are documented and unit tested. There is a thorough manual and tutorials for common use cases.

Boolean networks may be defined in BoolNet format or by Python functions, which are sometimes shorter than constructing equivalent Boolean expression, for example when defining so-called threshold functions that are best defined as arithmetic sums. Networks in other formats, e.g. the GINsim file format, must be converted to BoolNet first. To do so we suggest to go via the SBML-qual standard, see Chaouiya *et al.*, (2013), and use the functions LoadSbml and SaveNetwork of the R package BoolNet. Internally, PyBoolNet represents networks by their so-called prime implicants, which are all minimal conditions under which update functions change their values, see Klarner *et al.*, (2015).

The interaction graph (IG) and state transition graph (STG) of Boolean networks are represented as instances of the digraph class of NetworkX, see Hagberg *et al.*, (2008). Node, edge and graph attributes that are also dot keywords specify the visual style when graphs are drawn, see Ellson *et al.*, (2001). There are predefined functions that add frequently used combinations of properties to graphs, for example to distinguish activating and inhibiting edges of the IG. Nodes may be grouped into sub-graphs so that they appear closer together in the resulting layout. In addition, dynamical properties can be visualized on the IG, e.g. by indicating fixed and unsteady nodes in subspaces of interest for the dynamics, see Figure 1. More examples of visualizations of different aspects of the IG and STG can be found in the user manual. Currently, PyBoolNet supports three update strategies: the synchronous update of S. Kauffmann, see Kauffman (1993), the asynchronous update of R. Thomas, see Thomas (1991), and the so-called mixed update in which any number of variables may change their value during a transition. As a rule of thumb, images of IGs or STGs with around a thousand nodes can be computed within seconds using a force directed layout. Synchronous STGs are generally easier to draw because of the restricted out-degree. They can be drawn within a minute for networks with 15 components.

NetworkX offers various common graph algorithms. For example, there is a function to compute all shortest paths between two

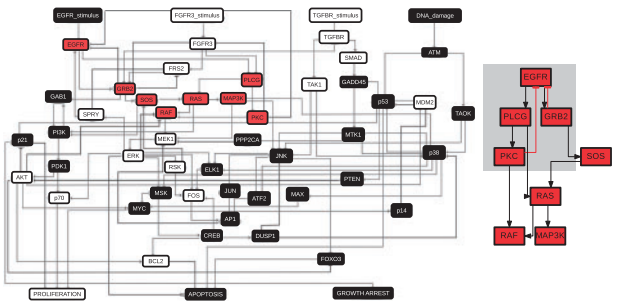

Fig. 1. An example of an interaction graph which is coloured according to one of its minimal trap spaces. Steady variables are black (*active*) or white (*inactive*) and oscillating variables are red. The sub-network that corresponds to the oscillating variables is given on the right. For details see Klarner *et al.*, (2015)

states in the STG, to compute all strongly connected components (SCCs), to enumerate simple cycles, perform link and clustering analysis, and so on. Depending on the network topology, a shortest path between two nodes can usually be found with seconds for graphs with around 50 000 nodes.

Model checking queries are decided by evoking NuSMV, a state of the art model checking tool, see Cimatti *et al.*, (2002), which supports specifications formulated in linear time logic (LTL) and computation tree logic (CTL). There is a module for the generation of common model checking query patterns. Supported transition relations are asynchronous, synchronous and mixed. PyBoolNet also supports counterexamples that are generated by NuSMV. They can be visualized as coloured paths in the STG or as animated IGs. In addition, the so-called accepting states of CTL queries can be returned as either Boolean expression representations or explicitly. To this end we wrote a custom extension of NuSMV, called NuSMV-a. Depending on the network topology and update strategy, model checking queries can usually be answered within minutes for networks with around 50 components, i.e. 2^{50} states.

There are several functions for detecting attractors of Boolean networks. The first one is based on explicitly generating the SCCs of the STG using NetworkX and enumerates all attractors. The second one detects states that belong to attractors by a random walk technique combined with model checking. The third one is based on so-called trap spaces and the iterative refinement algorithm proposed in Klarner and Siebert (2015). Steady states and trap spaces can usually be computed within seconds for networks with hundreds of components. It is to our knowledge the state of the art for computing the attractors of asynchronous STGs but an exact comparison with other methods remains to be done. Basins of attraction can also be computed via the accepting states of CTL queries. The approach is similar to the one described in Arellano *et al.*, (2011).

PyBoolNet is a convenient, general purpose Boolean network package that offers a simple interface to state of the art algorithms. Learning to use it is made easy by a thorough documentation and tutorials.

Funding

The work was partially funded by the German Federal Ministry of Education and Research (BMBF), grant no. 0316195. The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

Conflict of Interest: none declared.

References

Albert,I. *et al*. (2008) Boolean network simulations for life scientists. *Source Code Biol. Med*., **3**, 1.

Arellano,G. *et al*. (2011) Antelope: a hybrid-logic model checker for branching-time Boolean GRN analysis. *BMC Bioinformatics*, **12**, 490.

Baier,C. and Katoen,J.P. (2008) *Principles of Model Checking*. The MIT Press.

Chaouiya,C. *et al*. (2013) SBML qualitative models: a model representation format and infrastructure to foster interactions between qualitative modelling formalisms and tools. *BMC Syst. Biol*., **7**, 135.

Cimatti,A. *et al*. (2002) Nusmv 2: an opensource tool for symbolic model checking. In: *Proceedings of the 14th International Conference on Computer Aided Verification, CAV '02*, pp. 359–364, Springer-Verlag, London, UK.

Ellson,J. *et al*. (2001) Graphviz – open source graph drawing tools. In: *Graph Drawing*, pp. 483–484. Springer.

Hagberg,A. *et al*. (2008) Exploring network structure, dynamics, and function using networkx. In: *Proceedings of the 7th Python in Science Conferences (SciPy 2008)*, vol. **2008**, pp. 11–16.

Hopfensitz,M. *et al*. (2012) Multiscale binarization of gene expression data for reconstructing boolean networks. *IEEE/ACM Trans. Comput. Biol. Bioinf. (TCBB)*, **9**, 487–498.

Kauffman,S.A. (1993) *The Origins of Order: Self Organization and Selection in Evolution*. Oxford University Press, USA.

Klarner,H. and Siebert,H. (2015) Approximating attractors of Boolean networks by iterative CTL model checking. *Front. Bioeng. Biotechnol*., **3**, 130.

Klarner,H. *et al*. (2015) Computing maximal and minimal trap spaces of Boolean networks. *Nat. Comput*., **14**, 535–544.

Maucher,M. *et al*. (2011) Inferring Boolean network structure via correlation. *Bioinformatics*, **27**, 1529–1536.

Müssel,C. *et al*. (2010) Boolnet – an R package for generation, reconstruction, and analysis of boolean networks. *Bioinformatics*, 1378–1380.

Naldi,A. *et al*. (2009) Logical modelling of regulatory networks with ginsim 2.3. *Biosystems*, **97**, 134–139.

Naldi,A. *et al*. (2015) Cooperative development of logical modelling standards and tools with CoLoMoTo. *Bioinformatics*, **31**, 1154–1159.

Thomas,R. (1991) Regulatory networks seen as asynchronous automata: a logical description. *J. Theor. Biol*., **153**, 1–23.