nctx: Networks in ConTeXt

Mirco Schoenfeld
mirco.schoenfeld@uni-bayreuth.de

University of Bayreuth
Nuernberger Str. 38
95447 Bayreuth, Germany

July 2021

Abstract
In this paper, the programming library *nctx* is proposed. It is a collection of algorithms tailored to the analysis of attributed networks that have context information associated to nodes or edges. Key feature of this library is the ability to guide network analysis tasks by means of user-defined functions. These functions receive the current state of an analysis task such that context information can be accessed easily. The user-defined function is able to guide further execution of the analysis task providing a novel way of considering context information during the analysis of complex structure.

Keywords: network analysis, attributed networks, context-awareness, library, python, R, C++

1 Technical Report

1.1 Motivation

Analyzing complex phenomena by means of network analysis is a common way to gain knowledge about an underlying structure and the sensible mechanisms that influence interrelations between entities. Network analysis translates complex data into groups of nodes, metrics about the structure, or even rankings of nodes and edges. Such translations are easy to understand and well interpretable in terms of the metaphors for nodes and edges.

However, taking only structural aspects of complex data into account means leaving out a large part of the data that might illustrate or explain emergence and formation of the structure at hand [11]. Such illustrating information is called *context information*. Considering context information during the analysis of networks is the central advantage of the *nctx*-library compared to existing libraries providing network analysis functionality.

With *nctx*, researchers can define functions that are executed as part of a network analysis task and that allow to guide the task itself. Therefore, the functions are passed the current state of a network analysis task, i.e. the nodes that are currently processed or visited, such that context information can be accessed easily. The defined function can then use context information to allow or prohibit further processing of the current state of the network analysis task.

The *nctx*-library is built on top of Boost Graph Library [12] and is accessible via C++, R [10], and Python [13].

1.1.1 Objectives

The key objective of the *nctx*-library is to enable researchers to consider context information during the analysis of network data. Researchers are in control to model the way and the extent to which context information influence path discovery, path traversal, and centrality calculations. Further, researchers can use the provided functionality in C++, R, and Python.

1.1.2 Contributions

- We build a framework for modeling and analysing attributed networks
- Researchers can define custom functions that can guide shortest-path discovery, path traversal, and centrality calculations. This gives control to researchers how and to what extent attribute information have an influence on the analysis.
- We build on top of the Boost Graph Library, a peer-reviewed C++ library providing robust and efficient implementations of algorithms for network analysis
- The library can be used in C++, R, and Python

1.1.3 Outline

This work is structured as follows. The first part of this report provides information about the

The second part covered in Chapter 2 contains the library manual. The chapter provides information about installing the library in Section 2.1. It is

1.2 Related Work

There are a few libraries available that provide functionality for network analysis. One of the largest libraries that is also accessible cross-platform (in C/C++, R, and Python) is igraph [2]. It aims to provide efficient implementations of a large collection of network analysis tools with an easy-to-use interface to researchers.

A library with an emphasis on statistical analysis of networks is called graph-tool [9]. It is written in C++ and built upon the Boost Graph Library like the *nctx*-library. It is intended to be used via Python.

SNAP is a framework built for analysis and manipulation of large networks. SNAP is also written in C++ and provides interfaces in Python. SNAP is optimized for maximum performance and compact graph representation and, with SNAP, it is possible to process massive networks with hundreds of millions of nodes, and billions of edges [8].

A library only available in Python is networkx [7]. It provides a large collection of network analysis algorithms. Code written for networkx adheres to the look-and-feel of the Python language, i.e. nodes can be arbitrary objects like Python dictionaries, for example.

All of the above-mentioned libraries lack a core feature of *nctx*-library which is the ability to define functions that guide network analysis algorithms.

1.3 Concept

The general idea of the *nctx*-library is to enable researchers to consider context information during tasks of network analysis guiding, for example, shortest-path discovery or centrality calculations.

The library is therefore aimed at the analysis of attributed networks in which context information is associated with either nodes or edges. The type of context information is not restricted by the library. All conceivable data types are supported - from atomic Boolean or numeric values, to word strings, to complex data structures. For the most part, it is left to the researcher to manage the context information.

The library provides assistance where needed to make the data structures accessible for algorithmic analysis. This is especially true for the python-port of the library, where appropriate auxiliary structures are defined.

To take advantage of the special strength of the *nctx*-library, researchers can define functions that are executed during shortest-path discovery or centrality calculations. The functions are passed the status of the analysis by means of parameters. This ensures that the researcher can react to the current status of the analysis.

The intended use of the library is to retrieve context information about nodes during shortest-path discovery, for example, and to use the context information to decide whether certain subpaths should be followed or discarded. This allows for node-dependent and dynamic decision of how to process context information.

1.4 Implementation

The core of the *nctx*-library is written in C++. It is basically an adaptation of algorithms of the BGL, the Boost Graph Library [12].

Central elements of the C++ core of the library are a custom visitor to the BGL implementation of Dijkstra's algorithm for shortest path discovery, as well as a custom visitor to the implementation of Brandes' algorithm of Betweenness centrality. These visitors are responsible for calling the user-defined functions at each step of the algorithms. If a call to such a user-defined function prohibits the use of a certain edge, these visitors handle the required distance and centrality updates accordingly.

On top of this core there are two adaptors making the functionality available in R and Python. For Python, the adaptors make heavy use of Boost.Python [1]. For R, the adaptors are built around Rcpp [6, 3, 4] and the Boost library for Rcpp [5].

The Python port of nctx is split into three sub-modules, a sub-module for directed graphs, a sub-module for undirected graphs, and a sub-module containing utility functions. The decision for splitting the functionality for directed and undirected graphs into different submodules was due to the strongly typed nature of C++ and the BGL algorithms. They require a fixed type of graph at compile-time and, in BGL, the type specifies if the graph is directed or not. Using specific sub-modules allow to fix the type and directedness of the graph at compile time.

Beyond that, the most part of the Python port consists of details about type conversions between C++ and Python data structures and iterators for vertices and edges. An important part of that is passing BGL property maps to Python allowing researchers to save attribute information together with the graph and to export these information to files.

The R port of nctx is much more straightforward since Rcpp does a great job in mapping data structures between R and C++. This eases passing lists of data between R and C++ and less code is used to provide type conversions and details about iterators. The core of the adaptor is a C++ class consisting of templated calls to the library functions, and a mapping to specific S4 classes with concrete values for template parameters resulting in a S4 class for a directed graph and one for an undirected graph.

1.5 Future Work

At this point, the library is based on a single-core implementation of the Boost Graph Library. That means that shortest-path discovery and centrality calculations are executed on single-cores only while they could benefit tremendously from using all available cores instead. Future versions of this library will be adapted accordingly.

Further, we aim to integrate additional algorithms of network analysis into the library. For example, the PageRank algorithm is very well suitable to consider context information as well by replacing the `damping`-factor with a weight obtained from a user-defined function.

1.6 Conclusion

We built a library that assists in working with attributed networks having attributes or context information associated to nodes or edges. The novelty of *nctx* lies in the user-defined functions that can guide shortest-path discovery or centrality calculations allowing to enforce contextual constraints at the core of network analysis algorithms.

2 Library Manual

This Section provides a brief overview of how to install and use the library. For an up-to-date information on how to work with the library, visit https://nctx.mircoschoenfeld.de. The latest version of the library is available online: https://github.com/nctx.

2.1 Installation

This Section provides a brief overview of the steps required to use the *nctx*-library.

2.1.1 C++

To use this library as part of a C++ project, make sure to include the `nctx.hpp` in your source files, and link the `boost_graph`-library against your project. Make sure to include the directory of the `nctx.hpp` as well as the Boost include path.

2.1.2 Python

Installing the *nctx*-package for Python requires some compilation of source files. That requires the Boost Graph Library ($\geq 1.65.0$) to be installed on the system.

The compilation is then taken care of by the `skbuild`-package, however. If that package is installed, one needs to execute the following command in the directory of the package in order to build a `whl`-file:

```
python3 setup.py bdist_wheel
```

Inside the `dist`-folder there appears a `*.whl` file which needs to be installed:

```
pip install dist/nctx-0.1-*.whl
```

More detailed instructions on how to install this library can be found on https://github.com/nctx/py3nctx.

2.1.3 R

To install the *nctx*-extension for R use the `devtools`-library and install the library directly from github by executing the following lines in a R session:

```
library(devtools)
install_github("nctx/rnctx")
```

More detailed instructions on how to install this library can be found on https://github.com/nctx/rnctx.

2.2 Usage

The usage of this library is highlighted very briefly in the following Sections. For more details refer to the websites mentioned in the Sections.

The general idea is to have user-defined functions with the signature `(vertex index, vertex index, vertex index) -> bool`. These functions accept three vertex indices: the start vertex, the current vertex, and the descending vertex. The start vertex is the vertex for which the centrality is to be obtained, the source vertex of the shortest-path-discovery in the network, or the like. The current vertex is the one where the path traversal is currently at, deciding whether to visit the descending next vertex. The next vertex has a connection to the current vertex and the analysis algorithm is about to visit it. If the user-defined function evaluates to `false`, the edge to the next vertex is discarded.

2.2.1 C++

In C++, the user-defined function can be given using a lambda expression. The example demonstrates the use of a property map that is accessed inside the lambda expression. The `nxt` vertex is only visited if the lambda expression evaluates to `true`.

```
#include<nctx.hpp>
typedef typename b::graph_traits<Graph>::vertex_descriptor
   Vertex;
Graph g;
auto ctx_map = boost::make_vector_property_map<
   std::vector<double_t> >( boost::get(
   boost::vertex_index, g));
// ... fill the ctx_map here
auto weight_map = boost::make_constant_property< typename
   Graph::edge_descriptor >(1.0);
std::vector<double> centralityS(num_vertices(g), 0);
brandes_betweenness_centrality_ctx(g,
  centrality_map(make_iterator_property_map(centralityS.begin(),
     boost::get(boost::vertex_index, g)))
  .vertex_index_map(boost::get(boost::vertex_index, g))
  .weight_map(weight_map),
    [&ctx_map](Vertex start, Vertex current, Vertex nxt) ->
       bool {
      auto d_l =
         kl_divergence(ctx_map[current].begin(),ctx_map[current].end(),
                                      ctx_map[nxt].begin(),ctx_map[nxt].end());
      return d_l <= .35;
    });
```

More information can be found on https://nctx.mircoschoenfeld.de/cpp.

2.2.2 Python

In Python, we need to import either the submodule `undirected` or `directed` for the analysis of undirected or directed networks. The example demonstrates the use of a simple list consisting nominal data. The discovery of shortest paths between all pairs of nodes is guided by this attribute data such that edges are only allowed if both ends have the same attribute value in common.

```
import nctx.undirected as nctx
g = nctx.Graph()
# ... fill the graph here
context = [1,1,0,1,1,1,1,0,1,0]
def decision(strt, crnt, nxt):
    return context[crnt] == context[nxt]
distances = nctx.AlgPaths.dijkstra_apsp_ctx(g, decision)
```

More information can be found on https://nctx.mircoschoenfeld.de/python3.

2.2.3 R

The example demonstrates the use of a simple list consisting nominal data. The discovery of shortest paths between all pairs of nodes is guided by this attribute data such that edges are only allowed if both ends have the same attribute value in common.

```
require(nctx)
g <- create_graph(directed=FALSE)
# ... fill the graph here
context <- c(1,1,0,1,1,1,1,0,1,0)
decision_fct <- function(strt, crnt, nxt){
  context[crnt] == context[nxt]
}
distances <- all_pairs_shortest_paths_ctx(g, decision_fct)
```

More information can be found on https://nctx.mircoschoenfeld.de/R.

References

[1] David Abrahams and Ralf W Grosse-Kunstleve. "Building hybrid systems with Boost.Python". In: *C/C++ Users Journal* 21.7 (May 2003). URL: https://www.osti.gov/biblio/815409.

[2] Gabor Csardi and Tamas Nepusz. "The igraph software package for complex network research". In: *InterJournal* Complex Systems (2006), p. 1695. URL: https://igraph.org.

[3] Dirk Eddelbuettel. *Seamless R and C++ Integration with Rcpp*. ISBN 978-1-4614-6867-7. New York: Springer, 2013. DOI: 10.1007/978-1-4614-6868-4.

[4] Dirk Eddelbuettel and James Joseph Balamuta. "Extending R with C++: A Brief Introduction to Rcpp". In: *The American Statistician* 72.1 (2018), pp. 28–36. DOI: 10.1080/00031305.2017.1375990.

[5] Dirk Eddelbuettel, Jay Emerson, and Michael Kane. *bh: Boost Headers for R*. 2014. URL: http://dirk.eddelbuettel.com/code/bh.html.

[6] Dirk Eddelbuettel and Romain François. "Rcpp: Seamless R and C++ Integration". In: *Journal of Statistical Software* 40.8 (2011), pp. 1–18. DOI: 10.18637/jss.v040.i08. URL: https://www.jstatsoft.org/v40/i08/.

[7] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. "Exploring Network Structure, Dynamics, and Function using NetworkX". In: *Proceedings of the 7th Python in Science Conference*. Ed. by Gaël Varoquaux, Travis Vaught, and Jarrod Millman. Pasadena, CA USA, 2008, pp. 11–15.

[8] Jure Leskovec and Rok Sosič. "SNAP: A General-Purpose Network Analysis and Graph-Mining Library". In: *ACM Transactions on Intelligent Systems and Technology (TIST)* 8.1 (2016), p. 1.

[9] Tiago P. Peixoto. “The graph-tool python library”. In: *figshare* (2014). DOI: 10.6084/m9.figshare.1164194. (Visited on 09/10/2014).

[10] R Core Team. *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing. Vienna, Austria, 2020. URL: https://www.R-project.org.

[11] Mirco Schoenfeld and Juergen Pfeffer. “Networks and Context: Algorithmic Challenges for Context-Aware Social Network Research”. In: *Challenges in Social Network Research: Methods and Applications.* Ed. by Giancarlo Ragozini and Maria Prosperina Vitale. Cham: Springer International Publishing, 2020, pp. 115–130. ISBN: 978-3-030-31463-7. DOI: 10.1007/978-3-030-31463-7_8.

[12] Jeremy Siek, Andrew Lumsdaine, and Lie-Quan Lee. *The boost graph library: user guide and reference manual.* Addison-Wesley, 2002.

[13] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual.* Scotts Valley, CA: CreateSpace, 2009. ISBN: 1441412697.