Universiteit
Leiden
The Netherlands

# Unravelling vascular tumors : combining molecular and computational biology

IJzendoorn, D.G.P. van

**Citation**
IJzendoorn, D. G. P. van. (2020, January 16). *Unravelling vascular tumors : combining molecular and computational biology*. Retrieved from https://hdl.handle.net/1887/82754

| | |
|---|---|
| Version: | Publisher's Version |
| License: | Licence agreement concerning inclusion of doctoral thesis in the Institutional Repository of the University of Leiden |
| Downloaded from: | https://hdl.handle.net/1887/82754 |

**Note:** To cite this publication please use the final published version (if applicable).

Cover Page

Universiteit Leiden

The handle http://hdl.handle.net/1887/82754 holds various files of this Leiden University dissertation.

**Author**: IJzendoorn, D.G.P. van
**Title**: Unravelling vascular tumors : combining molecular and computational biology
**Issue Date**: 2020-01-16

# Part III

# Computational biology

# Chapter 7

# PyPanda: a Python package for gene regulatory network reconstruction

This chapter is based on the publication: **van IJzendoorn DGP**, Glass K, Quackenbush J, Kuijjer ML. PyPanda: a Python package for gene regulatory network reconstruction. *Bioinformatics*. 2016;32: 3363-3365.

## 7.1 Abstract

PANDA (Passing Attributes between Networks for Data Assimilation) is a gene regulatory network inference method that uses message-passing to integrate multiple sources of 'omics data. PANDA was originally coded in C ++. In this application note we describe PyPanda, the Python version of PANDA. PyPanda runs considerably faster than the C ++ version and includes additional features for network analysis.

## 7.2 Introduction

Accurately inferring gene regulatory networks is one of the most important challenges in the analysis of gene expression data. Although many methods have been proposed (1–4), computation time has been a significant limiting factor in their widespread use. PANDA (Passing Attributes between Networks for Data Assimilation) is a gene regulatory network inference method that uses message passing between multiple 'omics data types to infer the network of interactions most consistent with the underlying data (5). PANDA has been applied to understand transcriptional programs in a variety of systems (6–8). Here we introduce PyPanda, a Python implementation of the PANDA algorithm, following the approach taken in Glass et al (7). and optimized for matrix operations using NumPy (9). This approach enables the use of fast matrix multiplications using the BLAS and LAPACK functions, thereby significantly decreasing run-time for network prediction compared with the original implementation of PANDA, which was coded in C ++ and used for-loops (7). We observe further speed increase over the C ++-code because PyPanda automatically uses multiple processor-cores through the NumPy library. We have also expanded PyPanda to include common downstream analyses of PANDA networks, including the calculation of network in- and out-degrees and the estimation of single-sample networks using the recently developed LIONESS algorithm (10).

## 7.3 Approach

### 7.3.1 Comparing PANDA C ++-code to Python-code

We compared the C ++-code and Python-code versions of PANDA using several metrics. First, we assessed the two implementations by comparing the number of lines of code. Using the cloc utility we counted the number of lines of C ++-code and Python-code. The C ++-code counted 1132 lines of code. The Python-code counted 258 lines of code, significantly shorter (4.4 times) than the C ++-code. The Python-code also includes features such as the LIONESS equation and in- and out-degree calculation. Without these

features the Python-code is only 155 lines of code. Because the Python implementation is much more concise than the C ++-code it is easier to interpret and modify.

We compared the C ++-code and Python-code versions of PANDA using several metrics. First, we assessed the two implementations by comparing the number of lines of code. Using the cloc utility we counted the number of lines of C ++-code and Python-code. The C ++-code counted 1132 lines of code. The Python-code counted 258 lines of code, significantly shorter (4.4 times) than the C ++-code. The Python-code also includes features such as the LIONESS equation and in- and out-degree calculation. Without these features the Python-code is only 155 lines of code. Because the Python implementation is much more concise than the C ++-code it is easier to interpret and modify.

The speed of the network prediction was tested using simulated networks of Ne = Na dimensions, where Ne is the number of effector nodes and Na is the number of affected nodes. For each of several different network sizes (Ne = Na = 125 to Ne = Na = 2000 nodes, in steps of 125) we generated ten random 'motif data' networks according to the method described in Glass et al (7). We then ran the Python and C ++ versions of PANDA using these simulated motif data together with identity matrices for the protein-protein interaction and co-expression information. For runs on the same initial 'motif data' networks, we verified that the C ++-code and Python-code returned exactly the same output network, as expected due to the deterministic nature of PANDA.

The C ++-code only uses one CPU core. In comparing the C ++-code with the Python-code using a single core, we found a 2.07-fold speed-up relative to the C ++-code for the smallest network (Ne = Na = 125) tested. The speed increase of the Python-code over the C ++-code became larger as the network size increased. For example, the Python-code performed 12.31 times faster for the largest network (Ne = Na = 2000) (figure 7.1a). Recorded run times across the ten random networks had a standard deviation of 0.04s and 2.59s for the smallest (Ne = Na = 125) and largest (Ne = Na = 2000) networks, respectively using the C ++ code. Using the Python code these were reduced to 0.03s and 0.099s.

Given the abundance of multicore computing resources currently available, we also tested the speed increase when running the Python-code on multiple cores compared with running the Python-code on a single core. We found that for the smallest network the speed was 1.45 times faster when using 6 cores compared with using only a single core; for the largest network the speed increase was 3.7-fold (figure 7.1b).

This increase in speed enables reconstruction of networks with larger numbers of regulators and target genes. For example, using the Python-code significantly decreases the time required to infer a human gene regulatory network (Ne = 1000, Na = 20 000), from ∼18 h with the C ++-code to only about 2 h with the Python-code. This speed-up is

especially important as transcription factor motif databases are frequently updated to include more motifs. Further, the decreased running time helps to enable the estimation of network significance by making the use of bootstrapping/jackknifing methods much more feasible.

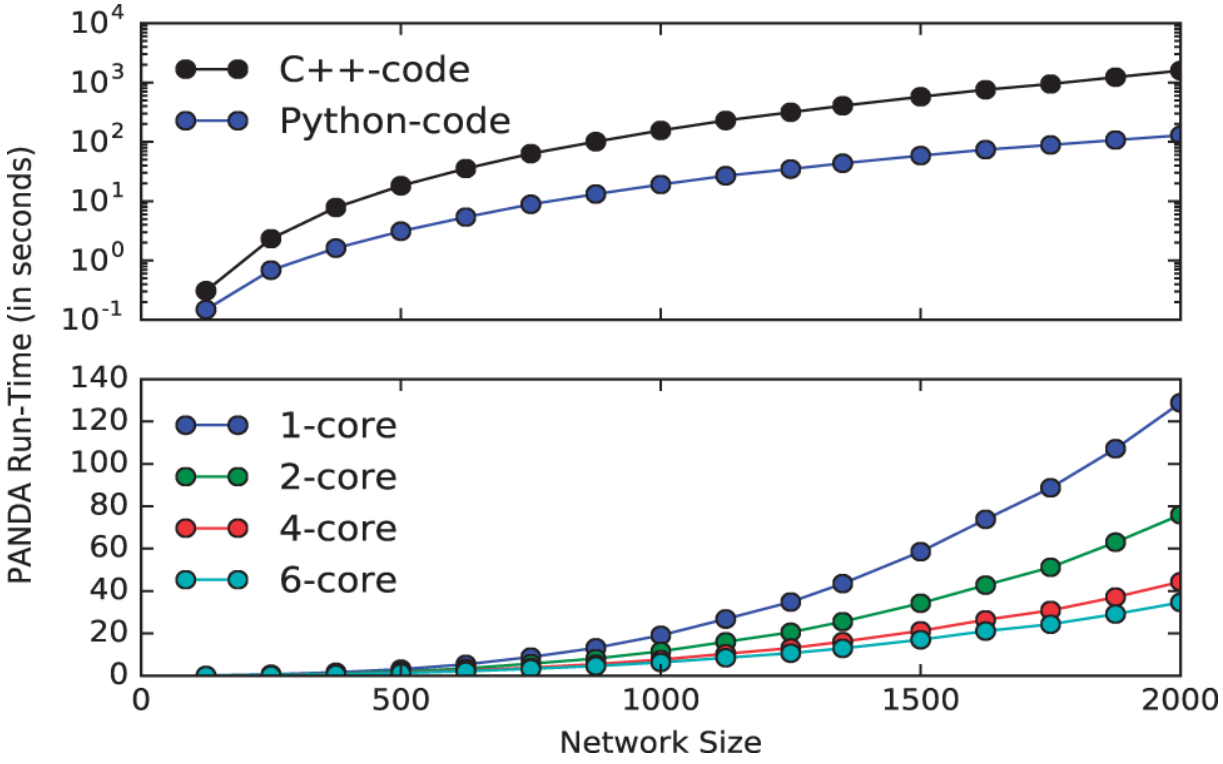


Figure 7.1: Speed comparison for network reconstruction on networks of different sizes using (a) the C ++-code and the Python-code, (b) the Python-code running on a single CPU compared with multicore (6 CPU cores).

### 7.3.2 Additional features

In addition to reconstructing one regulatory network based on a data set consisting of multiple samples, PyPanda can also reconstruct single-sample networks using the LIONESS algorithm (10). In PyPanda, the LIONESS method uses PANDA to infer an 'aggregate' network representing a set of N input samples, infers a network for N – 1 samples, and then applies a linear equation to estimate the network for the sample that had been removed. The process is then repeated for each sample in the original set, producing N single-sample networks. PyPanda can also use LIONESS to reconstruct single-sample networks based on Pearson correlation.

PyPanda also includes functions to calculate in-degrees (the sum of edge weights targeting a specific gene) and out-degrees (the sum of edge weights pointing out from a

regulator to its target genes). These summary metrics can be used for downstream network analysis (6).

## 7.4 Conclusion

PANDA is a proven method for gene regulatory network inference but, like most sophisticated network inference methods, its runtime has limited its utility. The Python implementation of PANDA uses matrix operations and incorporates the NumPy libraries, resulting in a significant simplification of the code and a dramatic increase in computing speed, even on a single processor. When applied to a test data set and run on multiple processing cores, this increase in speed was even greater, decreasing processing times by a factor of 46 relative to the original C ++-code. This creates opportunities to greatly expand the use of PANDA and to implement additional measures of network significance based on bootstrapping/jackknifing. PyPanda also includes the LIONESS method, which allows inference of single-sample networks, as well as a number of other useful network metric measures. The open source PyPanda package is freely available at `http://github.com/davidvi/pypanda`.

# Bibliography

[1] Altay G, Asim M, Markowetz F, Neal DE. Differential C3NET reveals disease networks of direct physical interactions. BMC Bioinformatics. 2011;12:296. doi:10.1186/1471-2105-12-296.

[2] Ernst J, Beg QK, Kay KA, Balázsi G, Oltvai ZN, Bar-Joseph Z. A semi-supervised method for predicting transcription factor-gene interactions in Escherichia coli. PLoS Computational Biology. 2008;4(3):e1000044. doi:10.1371/journal.pcbi.1000044.

[3] Faith JJ, Hayete B, Thaden JT, Mogno I, Wierzbowski J, Cottarel G, et al. Large-scale mapping and validation of Escherichia coli transcriptional regulation from a compendium of expression profiles. PLoS Biology. 2007;5(1):0054–0066. doi:10.1371/journal.pbio.0050008.

[4] Lemmens K, Dhollander T, De Bie T, Monsieurs P, Engelen K, Smets B, et al. Inferring transcriptional modules form ChIP-chip, motif and microarray data. Genome Biology. 2006;7(5):R37. doi:10.1186/gb-2006-7-5-r37.

[5] Glass K, Huttenhower C, Quackenbush J, Yuan GC. Passing Messages between Biological Networks to Refine Predicted Interactions. PLoS ONE. 2013;8(5):e64832. doi:10.1371/journal.pone.0064832.

[6] Glass K, Quackenbush J, Silverman EK, Celli B, Rennard SI, Yuan GC, et al. Sexually-dimorphic targeting of functionally-related genes in COPD. BMC systems biology. 2014;8:118. doi:10.1186/s12918-014-0118-y.

[7] Glass K, Quackenbush J, Spentzos D, Haibe-Kains B, Yuan GC. A network model for angiogenesis in ovarian cancer. BMC Bioinformatics. 2015;16:115. doi:10.1186/s12859-015-0551-y.

[8] Lao T, Glass K, Qiu W, Polverino F, Gupta K, Morrow J, et al. Haploinsufficiency of Hedgehog interacting protein causes increased emphysema induced by cigarette smoke through network rewiring. Genome Medicine. 2015;7(1):12. doi:10.1186/s13073-015-0137-3.

[9] van der Walt S, Colbert SC, Varoquaux G. The NumPy Array: A Structure for Efficient Numerical Computation. Computing in Science & Engineering. 2011;13(2):22–30. doi:10.1109/mcse.2011.37.

[10] Kuijjer ML, Tung M, Yuan G, Quackenbush J, Glass K. Estimating sample-specific regulatory networks. arXiv Molecular Networks (q-bioMN). 2015;doi:arXiv:1505.06440v2.