Tutorial on using BNfinder software

Norbert Dojer, Paweł Bednarz, Agnieszka Podsiadło and Bartek Wilczyński

December 18, 2012

Contents

1 Tutorial

This short tutorial presents the most common possible uses of the BNfinder software. The first part of this tutorial is devoted to presenting possible options of the software and the input files on simplistic, synthetic examples. In the second part, we provide more realistic examples taken from published studies of data for inferring dynamic and static networks.

In this tutorial, we will assume that you are using the standalone BNfinder application as downloaded from http://bioputer.mimuw.edu.pl/software/bnf, however if you want, you can also try out these examples with our web-server at http://bioputer.mimuw.edu.pl/BIAS/BNFinder.

If you have any questions regarding this document or the described software, please contact us: bartek@mimuw.edu.pl or dojer@mimuw.edu.pl

1.1 Synthetic examples

This section shows on several simple networks, how to prepare datasets and set the parameters for network reconstruction with BNfinder. The examples include a simple static network, dynamic network and a network requiring setting prior probabilities.
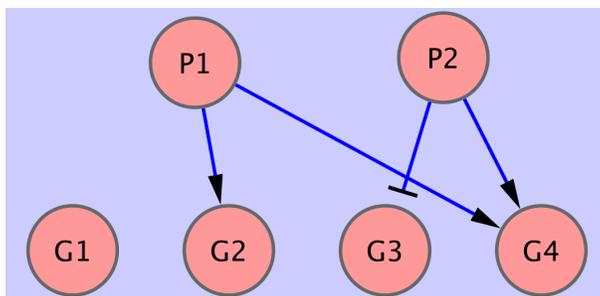
Figure 1: Very simple network consisting of 2 regulators and 4 observable regulatees.

Simple static network

The first example shows how to use BNfinder to learn a simple static Bayesian network. Let us imagine that we are analysisng cells under two conditions $P1$ and $P2$ and that we are interested whether any of the four genes: $G1, G2, G3, G4$ are responding to these conditions. We assume that the true network is depicted in Fig 1, i.e.

- $G1$ is not dependent on $P1$ or $P2$,

- $G2$ is more likely to be expressed under condition $P1$,

- $G3$ is less likely to be expressed under condition $P2$,

- $G4$ is more likely to be expressed under any of the conditions $P1$ or $P2$.

We have collected 100 datapoints from this network, each consisting of both the state of conditions and the discrete state of expression of the genes. You can download the input file here data/input1.txt.

If you open the file in a text editor, please note that the first line contains the information on the assumed structure:

```
#regulators P1 P2
```

This represents the fact, that we assume that genes $(G1..G4)$ can depend on conditions $(P1, P2)$ and not the other way around.

You can try to run BNfinder on this file:

```
bnf -e input1.txt -n output1.sif -v
```

and you will see, that the network topology is reconstructed properly. Also the orientation of the regulatory interactions is inferred properly as you can see in the output file data/output1.sif.

You can also try to see whether the optimal network is representative for a larger set of possible suboptimal networks:

```
bnf -e input1.txt -n output1w.sif -v -i 4 -t output1.txt
```

This time, in the output file (data/output1w.sif), the edge labels represent the relative weights of different edges. In the file data/output1.txt, we can find the originally computed weights (i.e. relative probabilities – see manual for details) for all considered suboptimal sets of parents for all genes.

Instead of fixing the number of returned parents sets (option -i 4) you can specify thresholds for their weights and/or weight ratios to optimal weights. For example, if you wish to get for each vertex v all parents sets with weights $> \max(0.1, 0.01 \cdot w_{opt}(v))$, where $w_{opt}(v)$ denotes the weight of the optimal parents set of v, you can type:

```
bnf -e input1.txt -n output1a.sif -v -i -1 -m 0.1 -o 0.01 -t output1a.txt
```

We can also try to analyze the data for this network without discretization: data/input2.txt. In this case we need another directive to indicate that some of the dataseries are continuous:

```
#continuous G1 G2 G3 G4
```

Again if we run BNfinder on this data,

```
bnf -e input2.txt -n output2.sif -v
```

we can verify, that the output file contains correct information data/output2.sif.

Simple dynamic network

BNfinder can be used also to infer dynamic Bayesian networks from time series data. In this case it is not necessary to specify the regulators sets, because DBNs, unlike static networks do not need to be acyclic.

In the first dataset: data/input3.txt, we have 1 serie of 20 consecutive measurements of gene expression from gene network depicted in Fig. 2.
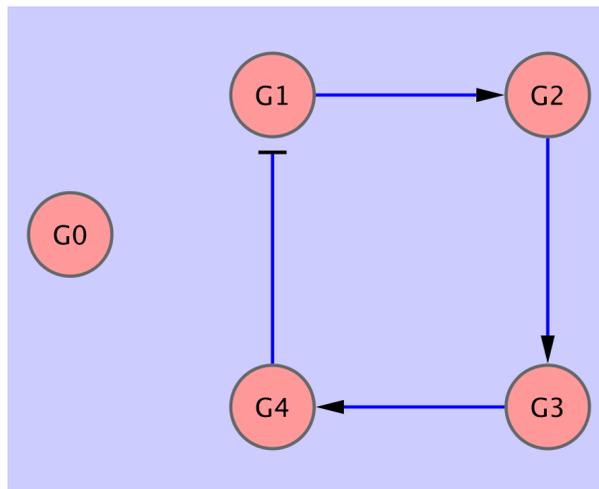

Figure 2: Very simple dynamic network consisting of 5 observables

If we run BNfinder on this data:

```
bnf -e input3.txt -n output3.sif -v
```

We can see that the program was unable to correctly reconstruct all the edges. Again, if we look at the statistics of edge occurences in suboptimal networks,

```
bnf -e input3.txt -n output3.sif -v -i 10 -t output3.txt
```

we can see that the correct edges are the most commonly occuring ones, but they score lower than empty parent sets.

In this case we can show how perturbational data can be integrated into this framework. We have collected gene expression from 5 time-series containing one single gene knockout for each of the genes: data/input4.txt. The perturbations are noted by including the following lines in the preamble of the data file:

```
#perturbed EXP1 G1
#perturbed EXP2 G2
#perturbed EXP3 G3
#perturbed EXP4 G4
#perturbed EXP5 G0
```

If we run BNfinder on the perturbed data, we can see that all the edges are reconstructed with high confidence.

```
bnf -e input4.txt -n output4.sif -v -i 10 -t output4.txt
```
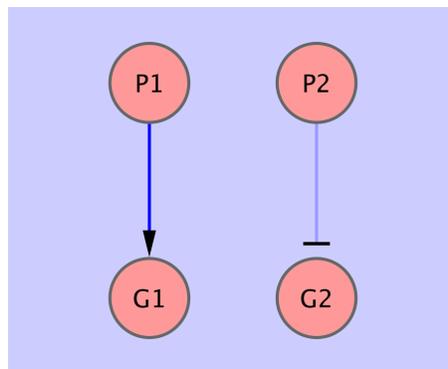
Setting priors

Figure 3: Exemplary network containing dependencies of different strength

In some cases it might be useful to include some prior information on the network structure into the process of inference. We will illustrate this on an example of a simple network similar to the one described in section 1.1. This time it is an even simpler network, with 2 conditions and 2 genes as depicted in Fig. 3. Even though topology of the network is very simple, the problem lies in the fact that the dependence of $G2$ on $P2$ is weaker than the dependence of $G1$ on $P1$. This is why if we run our software on the unmodified dataset data/input5.txt,

```
bnf -e input5.txt -n output5.sif -v  -i 10 -t output5.txt
```

we can see that the program is unable to recover the $P2 \rightarrow G2$ edge. However, if we expect that $G2$ is responding weakly to its regulators, we can increase the prior probability of $G2$ being regulated by any of the factors $P1, P2$ via decreasing its weight:

```
#prioredge G2 0.33 P2 P1
```

We can see the edge appearing in the result as expected (see data/input6a.txt):

```
bnf -e input6a.txt -n output6a.sif -v  -i 10 -t output6a.txt
```

Similarly, if we expect, that in general gene response to condition $P2$ is weaker, we may modify the prior probability of the condition $P2$ to be a regulator:

```
#priorvert 0.33 P2
```

The result of running BNFinder with this input (data/input6b.txt) is very similar to the previous one:

```
bnf -e input6b.txt -n output6b.sif -v  -i 10 -t output6b.txt
```

1.2   Examples of published datasets

In this section, we present two more realistic examples of published datasets used for inference of Bayesian networks. The first one consists of measurements of states of protein signalling network under different perturbations [1]. It's been used to infer causal relationships in the form of static Bayesian network.

The second dataset comes from documentation of the Banjo package [2] which can be downloaded from (http://www.cs.duke.edu/~amink/software/banjo). It consists of 2000 observations describing a relatively large dynamic network consisting of 20 nodes. It may be considered a benchmark of the efficiency of our algorithm.

The third dataset is converted from an example attached to the globalMIT software for Bayesian network reconstruction. It is similar to the second example as it is also generated from a dynamic Bayesian network and consists of 2000 observations of 20 variables. However, in this case the variables are much less interconnected and there are many self-regulatory loops.

PLEASE NOTE that these datasets are too large to be run through BNFinder webserver. If you would like to run them, please download the software.

Static Protein signalling network

In this section we present how BNfinder can be applied to a protein signalling network analyzed by Sachs et al [1]. We took the data from the article, and transformed it into the format suitable for BNfinder. We also needed to specify several properties of the data in the preamble of the file data/sachs.inp

Firstly, we needed to specify that the data are continuous measurements:

```
#continuous praf pmek plcg PIP2 PIP3 p44/42 pakts473 PKA PKC P38 pjnk
```

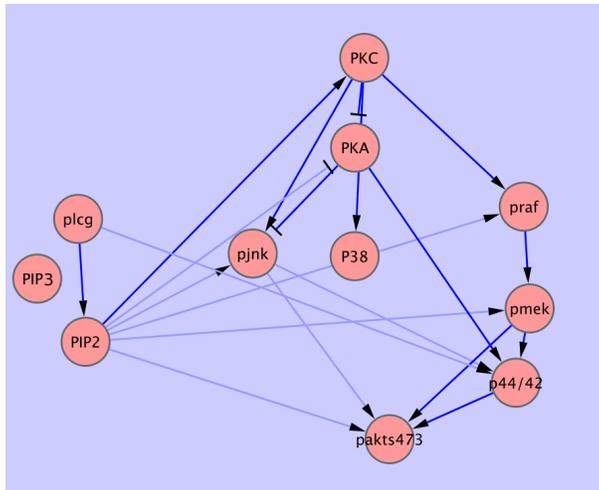Then, we needed to specify the expected layer structure of the signalling pathway we are studying:

Figure 4: Reconstruction of the protein signalling network. Dark blue arrows represent dependencies found in literature. Light blue arrows represent dependencies found by BNfinder but not expected by Sachs et al. [1]

```
#regulators plcg
#regulators PIP3
#regulators PIP2
#regulators PKC
#regulators PKA
#regulators praf
#regulators pjnk pmek P38
#regulators p44/42
#regulators pakts473
```

Then we needed to specify which of the proteins are affected by different perturbations.

```
#perturbed cd3cd28psitect_0 PIP2
#perturbed cd3cd28psitect_1 PIP2
#perturbed cd3cd28psitect_2 PIP2
...
#perturbed cd3cd28g0076_0 PKC
#perturbed cd3cd28g0076_1 PKC
#perturbed cd3cd28g0076_2 PKC
...
```

When we finally run the BNfinder:

```
bnf -e sachs.inp -n sachs.sif -v
```

We obtain the network presented in Fig. 4. As we can see, the topology is quite consistent with the literature data. Out of 17 expected edges, BNfinder recovers 11 correctly.

Dynamic Bayesian network

This is a dataset of substantial size which is used [3] to assess the performance of our inference algorithm. The input dataset (data/input7.txt) consists of 2000 measurements of 20 variables and it takes approximately 3 hours to compute it on a modern PC (2.4Ghz Intel Core 2 duo).

We can run BNfinder with the following command (note that we are using the -l option to limit the number of parents to 5):

```
bnf -e input7.txt -n output7.sif -v -l 5 -txt output7.txt
```

In Fig. 5 we can see part of the network reconstructed by BNfinder. All the edges reported by Banjo are also present in the optimal network (dark blue). The optimal network contains a number of additional edges, not reported by Banjo.
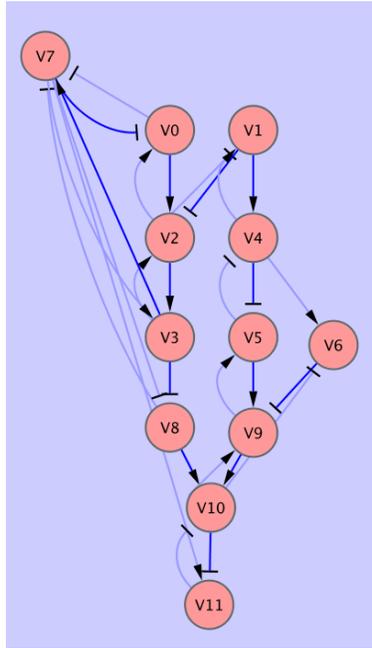

Figure 5: The optimal network reconstructed by BNfinder from the dynamic benchmark dataset. The edges reported also by Banjo are shown in dark blue.

If you want to see how much faster the MDL algorithm is, you can also run BNfinder with the following command:

```
bnf -s MDL -e input7.txt -n output7mdl.sif -v -l 5 -txt output7mdl.txt
```

Dynamic network containing self-regulatory loops

In this example, we can utilize both the -g 1 option for allowing self-regulations as well as the -s MIT option for using the MIT score.

```
bnf -s MIT -e input8.txt -n output8.sif -v -l 3 -txt output8.txt -c output8.cpd -g 1
```

One additional parameter that is unique to the MIT score is the significance level α of the χ^2 distribution (-a). The default level for α is .9999, but we can increase/decrease it if we want to see fewer/more edges respectively.

For example, setting the level alpha to a higher value should give us more edges in the result:

```
bnf -s MIT -e input8.txt -n output8.sif -v -l 3 -a .9  -g 1
```

Using multiple processors for faster computations

Since most current computers are equipped with multiple processors, we can take advantage of that fact to speed up BNFinder computation. Especially for large datasets, such as the ones described in previous sections, we can take full advantage of the parallell computation. For example, if we want BNfinder to run on 4 CPUs in parallell, we can use the `-k 4` option as in the following example:

```
bnf -s MIT -e input8.txt -n output8.sif -v -l 3 -a .9  -g 1 -k 4
```

1.3   Example of classification with BNfinder

`bnf-cv` and `bnc` tools can be used to solve classification tasks with classifier based on Bayesian networks. The former is used to perform a cross-validation test and the later to classify a dataset when you already have a classifier. In our example we will try to solve the following problem: we have points within the unit square; our positive set consists of those that are located in top-right and bottom-left corners, i.e. x + y > 1.8 or x + y < 0.2. The training set consists of 100 positive and 100 negative examples. They are visualised in the following Fig. 6. The data can be downloaded from here (data/training_set.txt). We marked x and y as continuous regulators. We will classify only one feature, but it is possible to perform cross-validation and classification procedure for more variables. All variables not marked as regulators are treated as variables to be explained by classifier.
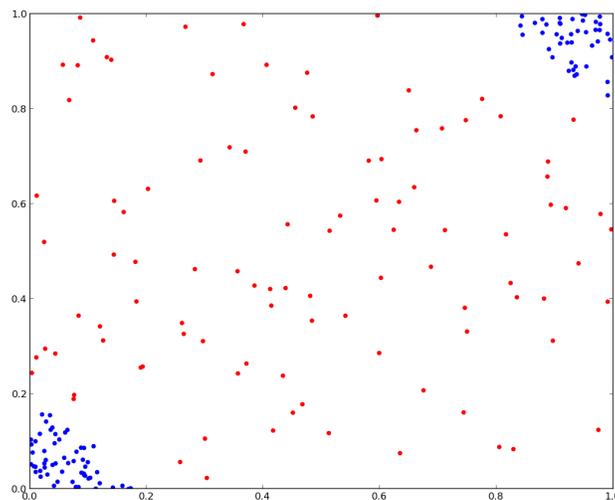

Figure 6: The training set used in the classification example. Positive examples are coloured blue.

To perform a 10-fold cross-validation we can use the following command:

```
bnf-cv -e training_set.txt -c net.cpd -k 10 -r ROC.pdf
```

As a result we obtain 10 files (net.cpd0, net.cpd1, ..., net.cpd9) containing networks in cpd format corresponding to respective folds of the cross-validation. Every execution of foregoing command will bring different results, because a split into 10 sets is done randomly. In the result file ROC.pdf there is a ROC plot showing the results of cross-validation (see 7). Further results are printed to the standard output and contains (among others) information about regulators taken to each of 10 classifiers and AUC measure of each classifier's performance.
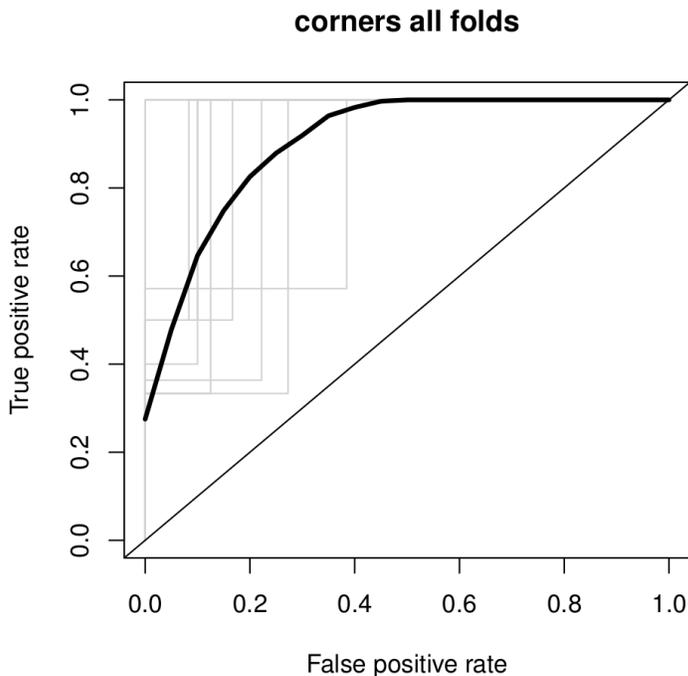
corners all folds

Figure 7: The Receiver operating characteristics curve for 10-fold cross-validation. The thick curve shows the average performance of classifiers.

To perform classification task on a test dataset we can use any of the nets obtained from cross-validation task but usually it is better to train a classifier on the whole training dataset. It can be done by the following command:

```
bnf -e training_set.txt -c net.cpd
```

We will test out classifier on this (data/test_set.txt) dataset which consists of 1000 points from the unit square. Now, by using the `bnc` tool we can obtain the classification. In the Fig. 8 we can see the result of classifying our test dataset by classifier in the file net.cpd (we used 0.63 probability threshold to generate the plot):

```
bnc -o result.cls -p 1 -c net.cpd -d test_set.txt
```

We can also find the most probable class for `corners` for every experiment by executing:

```
bnc -o result.cls -m 1 -c net.cpd -d test_set.txt
```
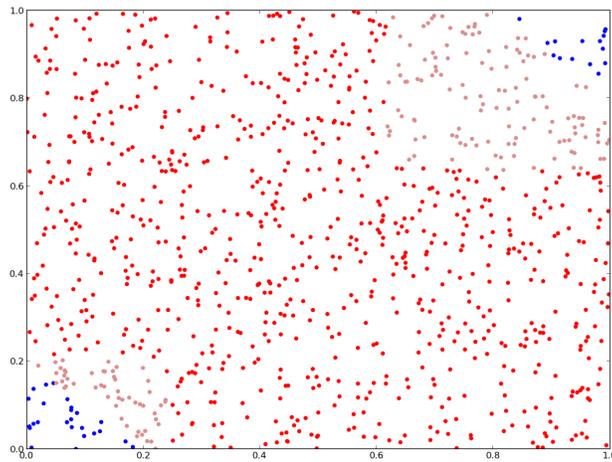
Figure 8: The result of classification. Blue and red points represents true positives and negatives. There was no false negatives. False positives are coloured light red.

References

[1] K. Sachs, O. Perez, D. Pe'er, D.A. Lauffenburger, and G.P. Nolan. Causal protein-signaling networks derived from multiparameter single-cell data. *Science*, 308:523–529, Apr 2005.

[2] V. Anne Smith, Jing Yu, Tom V Smulders, Alexander J Hartemink, and Erich D Jarvis. Computational inference of neural information flow networks. *PLoS Comput Biol*, 2(11):e161, Nov 2006.

[3] B. Wilczyński and N. Dojer. Bnfinder: exact and efficient method for learning bayesian networks. *Bioinformatics*, 25(2):286, 2009.