Klotz et al. BMC Bioinformatics 2013, 14(Suppl 10):S4
http://www.biomedcentral.com/1471-2105/14/S10/S4

BMC Bioinformatics

RESEARCH Open Access

Computing preimages of Boolean networks

Johannes Georg Klotz*, Martin Bossert, Steffen Schober

From 10th International Workshop on Computational Systems Biology
Tampere, Finland. 10-12 June 2013

Abstract

In this paper we present an algorithm based on the sum-product algorithm that finds elements in the preimage of a feed-forward Boolean networks given an output of the network. Our probabilistic method runs in linear time with respect to the number of nodes in the network. We evaluate our algorithm for randomly constructed Boolean networks and a regulatory network of *Escherichia coli* and found that it gives a valid solution in most cases.

Introduction

In systems and computational biology Boolean networks (BN) are widely used to model regulative dependencies of organisms [1,2]. We consider networks, which map a set of environmental conditions to the presence of proteins and finally to actual chemical reactions, which are often modeled as fluxes of a *flux-balance analysis* [3]. Hence, these networks are used to make *in silico* predictions of behavior of organisms in a certain environment [4].

In this paper we address the inverse problem, i.e., we want to predict environmental conditions that allow certain reactions to take place, and others not. Hence, in general, we need to find a set of possible inputs that lead to a given output. This so called *predecessor problem* or *preimage problem* has been addressed by Wuensche in [5] and has been shown to NP-hard in general [6], which makes it infeasible to solve it for large networks. In [7] an algorithm with reduced complexity for BNs with canalizing Boolean functions has been introduced. However, the problem is infeasible under certain conditions. Both algorithms are designed to find the whole set of preimages, i.e., all inputs to the BN with lead to a certain, desired, output.

In some applications, knowledge of the whole preimage set is not important, merely it can be sufficient to know a subset of the preimage set. Here, we propose a probabilistic algorithm, which solves this problem in linear time with respect to the number of nodes in the network, based on a variation of the well known Sum-Product Algorithm (SPA) [8], which is used for a variety of tasks, including decoding error correction codes in communication engineering [9].

Methods

Boolean networks and main idea

We consider networks like shown in Figure 1, mapping the values of the N in-nodes $\mathbb{I} = \{1, 2, 3\}$ to the M out-nodes $\mathbb{O} = \{12, 13, 14, 15, 16\}$, i.e., we can represent this BN as a function mapping the N input values uniquely to the M output values:

$$\mathbf{f} : \{0, 1\}^N \rightarrow \{0, 1\}^M.$$

The network itself consists of n nodes, and a set of directed edges connecting these nodes. Each node i has a certain state, which can be either zero or one, represented by a variable x_i. Its value is determined by evaluating a Boolean function (BF) f_i. Further, lets define the set $\tilde{n}(f_j)$ as the incoming nodes of node j. For example in Figure 1, $\tilde{n}(f_5) = \{1, 3\}$. The BF f_j is a function mapping $k_j = |\tilde{n}(f_j)|$ values of $\{0, 1\}^k$ to $\{0, 1\}$, where k is also called the in-degree of node j. The number of edges emerging from a node is called out-degree.

Given a vector of input values $\mathbf{x} \in \{0, 1\}^N$, $\mathbf{x} = (x_1, x_2, \ldots, x_N)$ the corresponding output of $\mathbf{f}$ is $\mathbf{y} = \mathbf{f}(\mathbf{x})$, $\mathbf{y} \in \{0, 1\}^M$. In general there does not exist a unique inverse function $\mathbf{f}^{-1}$. Instead the cardinality of the set $\Omega_y := \{x : \mathbf{f}(\mathbf{x}) = \mathbf{y}\}$ will be larger one. We call Ω_y the set of preimages of y. In this paper we are interested to find at least parts of Ω_y. Suppose there is a probability distribution $P_\mathbf{y}$ on $\{0, 1\}^N$ such that

* Correspondence: johannes.klotz@uni-ulm.de
Institute of Communications Engineering, Ulm University, Albert-Einstein-Allee 43, 89081 Ulm, Germany

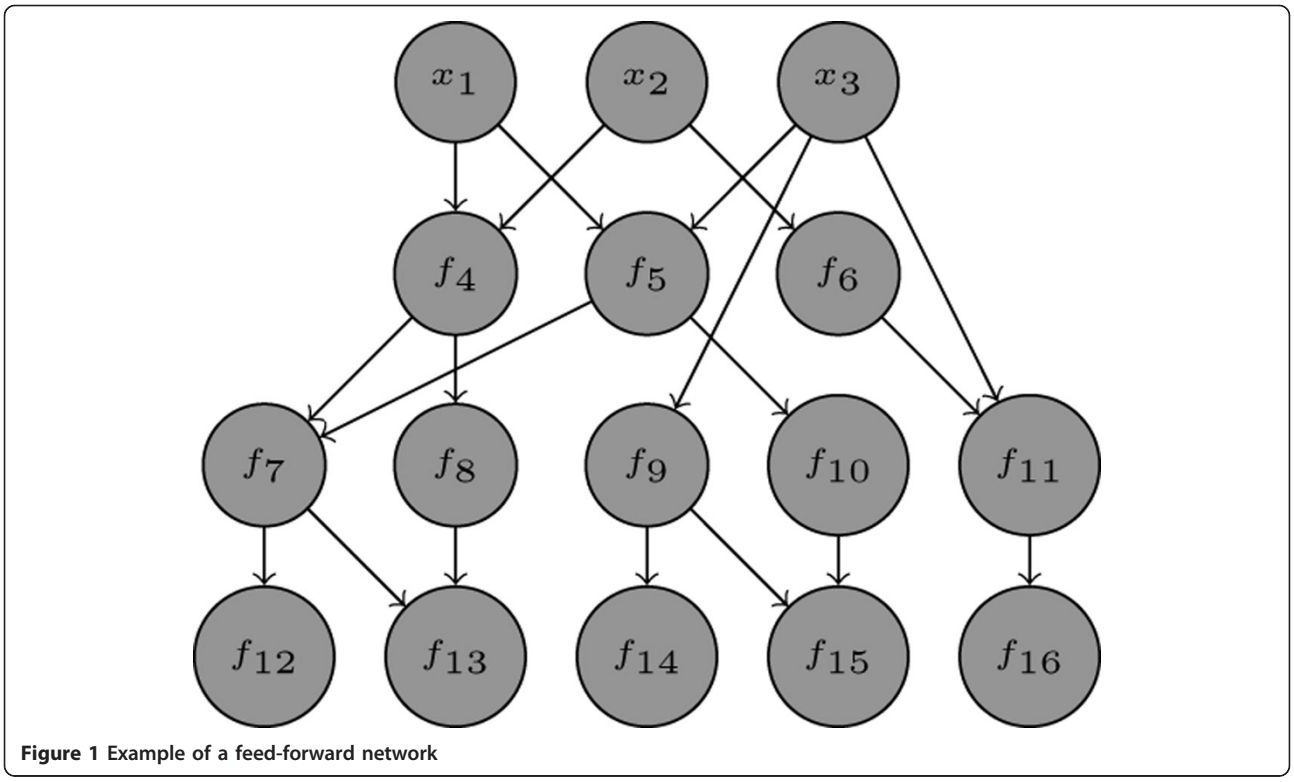

Figure 1 Example of a feed-forward network

$$P_{\mathrm{y}}\{\mathrm{x}\} = \begin{cases} \frac{1}{|\Omega_y|} & \text{if } \mathrm{x} \in \Omega_y \\ 0 & \text{else} \end{cases}.$$

If we knew the probability distribution $P_{\mathbf{y}}$, we would have solved the problem. But as explained, this is too difficult in general. Our main idea now is to approximate $P_{\mathbf{y}}$ by the product of the marginal distributions P_i on the individual x_i, i.e.,

$$P_{\mathrm{y}} \approx \prod_{i=1}^{N} P_i,$$

as the well-known SPA can be used to compute the marginals efficiently. If the approximation is *good enough* sampling out the product of the marginals will yield an element in Ω_y with reasonable probability.

Proposed algorithm

In this section we will first discuss the basic principles of factor graphs and the SPA. Then we will describe the BN as factor graph and will formulate the actual algorithm to find the marginals. Finally, the sampling is described.

Factor graphs and sum-product algorithm

Assume some function $g(x_1, \ldots, x_n)$ defined on some domain $\mathbb{A}^n$, which can be factorized in m *local* functions h_j, $j \in [m] := \{1, 2, \ldots, m\}$, i.e.,

$$g(x_1, \ldots, x_n) = \prod_j h_j(X_j),$$

where X_j is the subset of $[n]$ containing the argument of h_j . We can then define a factor graph [8] as a bipartite graph consisting of n nodes representing variables $\{x_1, \ldots, x_n\}$ (variable nodes) and of m nodes representing functions $\{h_1, \ldots h_m\}$ (function node). Edges exist between a function node and a variable node if and only if x_i is an input to function h_j .

The marginal function $g_i(x_i)$ is defined as [8]

$$g_i(x_i) = \sum_{\sim\{x_j\}} g(x_1, \ldots, x_n),$$

where $\sum_{\sim\{x_j\}} g(x_1, \ldots, x_n)$ is defined as

$$\sum_{\sim\{x_i\}} g(x_1, \ldots, x_n) = \sum_{x_1 \in \mathbb{A}} \cdots \sum_{x_{i-1} \in \mathbb{A}} \sum_{x_{i+1} \in \mathbb{A}} \cdots \sum_{x_n \in \mathbb{A}} g(x_1, \ldots, x_n),$$

In general the computation of the g_i is difficult, but due to the factorization of g the task can be efficiently solved using the the so called Sum-Product Algorithm (SPA) [8]. The algorithm iteratively passes *messages* between the nodes of the graph. At each iteration the messages μ are sent from the function nodes to the variable nodes, containing the corresponding marginal

function of the local function. These messages are computed as follows [8]:

Function to variable node

$$\mu_{h\to x}(x) = \sum_{\sim\{x\}} \left(h(n(h)) \prod_{\gamma\in n(h)\setminus\{x\}} \lambda_{\gamma\to h}(\gamma) \right),$$

where $n(i)$ give the set of neighboring nodes of node i.

At the variable nodes, these messages are then combined to a marginal function λ and sent back to the function nodes [8]:

Variable to function node

$$\lambda_{x\to h}(x) = \prod_{q\in n(x)\setminus\{h\}} \mu_{q\to x}(x).$$

The Boolean network as factor graph

We apply the concept of factor graphs to BNs. Each node in the network represents one variable $x_i \in \{0, 1\}$, $i \in [n]$ of the factor graph, hence we have n variable nodes. Each BF f_j of the BN ($j \in [n]\setminus\mathbb{I}$) is a function node and is connected to the node j and the incoming nodes $\tilde{n}(f_j)$. Lets to define $\tilde{X}_j$ as the variables of the incoming nodes of node j, i.e. the argument of the BN f_j. Further, we define $\widetilde{X}_j^{(i)}$ as $\widetilde{X}_j$ without the node i.

Finally, if we consider the variables as each node as random variables, we have a common distribution of all variables nodes described by the density function,

$$g_{x_1,\ldots,x_n}(x_1, \ldots, x_n) \equiv g(x_1, \ldots, x_n),$$

For sake of readability we will omit the subscripts of the density function, if they are obvious from context. We are interested in finding the marginal distributions of the in-nodes, which can be described by the density functions

$$g_{x_i}(x_i) = \sum_{\sim x_i} g_{x_1,\ldots,x_n}(x_1, \ldots, x_n) \qquad \forall i \in \mathbb{I}.$$

This problem is an instance of the problem described in Section *Factor Graphs and Sum-Product Algorithm*, hence we apply the same methods here.

Update rule: function to variable node

If we focus on one function node $j \in [n]\setminus\mathbb{I}$ there exists a common distribution of all variables relevant for this node. Namely, these relevant variables are the ones located in $\widetilde{X}_j$ of the BF f_j, and the value of node j. We can write the density of this distribution as:

$$p(x_j, \widetilde{X}_j).$$

Lets define $\tilde{n}(f_j)$ as the set of indices of the input nodes of the BF f_j.

We need to send the local marginal distribution of each variable $i \in \{j\} \cup \tilde{n}(f_j)$ back to the variable node, or more formally:

$$\mu_{j\to i}(x_i) = \sum_{\sim\{x_i\}} p(x_j, \widetilde{X}_j) = \sum_{\sim\{x_i\}} p(x_j, x_i, \widetilde{X}_j^{(i)}) \tag{1}$$

If $i = j$, i.e. if the message is designated for the node containing the output of the BF, the density of the marginal distribution becomes:

$$\begin{aligned}\mu_{j\to j}(x_j) &= \sum_{\sim\{x_j\}} p(x_j|\widetilde{X}_j)\cdot(\widetilde{X}_j) \\ &= \sum_{\sim\{x_j\}} f_j(\widetilde{X}_j)\cdot(\widetilde{X}_j)\end{aligned}$$

which is the probability distribution of the functions output. We can assume that the elements of $\tilde{X}_j$ are pairwise independent, hence we can write:

$$p(\widetilde{X}_j) = \prod_{l\in\tilde{n}(f_j)} \lambda_l(x_l),$$

where λ_l is the probability distribution of variable node l and is defined in Eq. 3.

In the other cases, i.e., $i \neq j$, Eq. (1) becomes:

$$\mu_j \to i(x_i) = \sum_{\sim\{x_i\}} p(x_i|x_j, \widetilde{X}_j^{(i)})\cdot(x_j, \widetilde{X}_j^{(i)}).$$

We still can assume that the elements of $\widetilde{X}_j^{(i)}$ are pairwise independent, hence we can write:

$$\begin{aligned}p(x_j, n(f_j)\setminus x_i) &= p(x_j|\widetilde{X}_j^{(i)})\cdot(\widetilde{X}_j^{(i)}) \\ &= p(x_j|\widetilde{X}_j^{(i)}) \prod_{l\in\tilde{n}(f_j)\setminus\{i\}} \lambda_l(x_l).\end{aligned}$$

If the Boolean functions output $x_j = f_j(\widetilde{X}_j)$ is already completely determined by $\widetilde{X}_j^{(i)}$, i.e., if the variable x_i has no influence on the output for this particular choice of the other variables, we assume x_i to be uniformly distributed:

$$p(x_i|x_j, \widetilde{X}_j^{(i)}) = \frac{1}{2} p_{x_j}(f(\widetilde{X}_j^{(i)}, x_i) = x_j)$$

and since x_j is completely determined by $\widetilde{X}_j^{(i)}$

$$p(x_j, \widetilde{X}_j^{(i)}) = \prod_{l\in\tilde{n}(f_j)\setminus\{i\}} \lambda_l(x_l).$$

Otherwise, x_i is totally determined by x_j and the other variables, i.e., x_i is 0 or 1 depending on BF. Hence, we can write

$$p(x_i|x_j, n(f_j)\setminus x_i) = p_{x_j}(f(\widetilde{X}_j^{(i)}, x_i) = x_j),$$

where $p_{x_j}(f(\widetilde{X}_j^{(i)},\ x_i) = x_j)$ is either 0 or 1. Further we can assume x_j independent of $\widetilde{X}_j^{(i)}$, hence

$$p(x_j, \widetilde{X}_j^{(i)}) = \lambda_j(x_j) \prod_{l \in \tilde{n}(f_j)\setminus\{i\}} \lambda_l(x_l).$$

Finally, we can summarize for $i \neq j$:

$$\mu_{j\to i}(x_i) = \sum_{\sim\{x_i\}} \xi_{i,j} p_{x_j}(f(\widetilde{X}_j^{(i)}, x_i) = x_j) \prod_{l \in \tilde{n}(f_j)\setminus\{i\}} \lambda_l(x_l), \quad (2)$$

with

$$\xi_{i,j} = \begin{cases} \frac{1}{2} & , \text{if } f_j(\widetilde{X}_j^{(i)},\ x_i = 0) = f_j(\widetilde{X}_j^{(i)},\ x_i = 1) \\ \lambda_j(x_j) & , \text{else} \end{cases}.$$

Update rule: variable to function node

The update rule is the same for all variable nodes $j \in [n]$ and is independent of the function node to which they are directed.

$$\lambda_j(x_j) = \prod_{l \in \mathbb{S}_j} \mu_{l\to j}(x_j),$$

where $\mathbb{S}_j$ is the set of all function nodes, which have node j as input.

Finding the input distributions

In our algorithm, we use the well known log-likelihood ratio (LLR) to represent the probability distribution of binary variables [10]. It is defined as:

$$L_X = \ln \frac{p(x = 0)}{p(x = 1)}. \quad (4)$$

A scheme of the algorithm is given in Algorithm 1.

The probability distribution of each node $j \in [n]$ at iteration t is given as $L_j^{(i)}$ and are initialized with $L_j^{(0)} = 0$, which is equivalent to the uniform distribution. Then we set the LLRs for the out-nodes to either $-\infty$ or $+\infty$ depending on the desired output $\mathbf{y}$ of the BN. At each iteration the algorithm can be split in two steps. The first step iterates over all function nodes $(j \in [n]\setminus\mathbb{I})$ and all input variables $i \in \tilde{n}(f_j)$ calculating the LLR $L_{j\to i}^{(t)}$ using Eq. (2) and Eq. (4).

In the second step we update all variables-nodes, where the LLRs L_j represents the distributions λ_j and, hence the product of Eq. 3 becomes a summation. Please note, that the LLR of the previous iteration is also added to the sum, in order to prevent rapid changes of the distributions.

After performing a certain number of iterations t_{max}, the desired marginal distributions of the input variables are found.

Algorithm 1

Initialize $L_j^{(0)} = 0$ for all nodes

Set the desired LLRs of the out-nodes, i.e., $L_j^{(0)}$ is either $-\infty$ or $+\infty$, for all out-nodes $j \in \mathbb{O}$.

t = 0

repeat

t=t+1

for each non-in-node $(j \in [n]\setminus\mathbb{I})$**do**

for each input variable $i \in \tilde{n}(f_j)$ **do**

calculate $L_{j\to i}$ using Eq. (2) and Eq. (4)

end for

end for

for each non-out-node ν **do**

$L_j^{(t)} = L_j^{(t-1)} + \sum_{l \in \mathbb{S}_j} L_{l\to j}^{(t)}$

end for

until maximum number of iterations reached

Sampling

The sampling part of our approach is straightforward. Using the marginal distributions $L_j^{(t_{max})}$, $j \in \mathbb{I}$ we randomly draw vectors $\mathbf{x}$ and check if they fulfill $\mathbf{y} = f(\mathbf{x})$. If so, they are added to the set $\widetilde{\Omega}_\mathbf{y}$. This procedure is repeated for a certain number of samples.

Simulation results and discussion

We tested our algorithm with randomly generated networks and the regulatory network of *Escherichia coli* (*E-coli*) [2]. The random networks consist of 2400 nodes with $N = 200$ and $M = 1200$. We have chosen the BFs from:

· all functions with $k \leq 15$ (Type A)

unate, i.e. locally monotone, functions with $k \leq 15$ (Type B)

After generating a network we draw a certain number T of uniformly distributed input vectors $\mathbf{x}$ and obtain $\mathbf{y} = \mathbf{f}(\mathbf{x})$. For each $\mathbf{y}$ we applied then Algorithm 1 to obtain the marginal distributions $L_j^{(t_{max})}, j \in \mathbb{I}$. To investigate the convergence behavior with respect to t_{max} we first apply hard-decision to evaluate a good choice for t_{max}, i.e., we generate an estimate $\tilde{x}$ by setting

$$\tilde{x}_j = \begin{cases} 0 \text{ if } L_j^{(t_{max})} > 0 \\ 1 \text{ if } L_j^{(t_{max})} < 0 \end{cases}$$

Then we evaluate the network $\tilde{y} = \mathbf{f}(\tilde{x})$, and measure the similarity between $\mathbf{y}$ and $\tilde{y}$ by counting the equal entries and divide them by the length of $\mathbf{y}$. We did so for 100 networks of Type A and B, and set $T = 100$. The averaged results can be seen in Figure 2.

One can see, that for $t_{max} \geq 14$ there is almost no improvement in the similarity. This number is equal to two times the number of nodes between input and output, i.e., it seems to be sufficient that the messages travel once through the network and back. Thus, the following simulations have been perform setting $t_{max} = 14$.

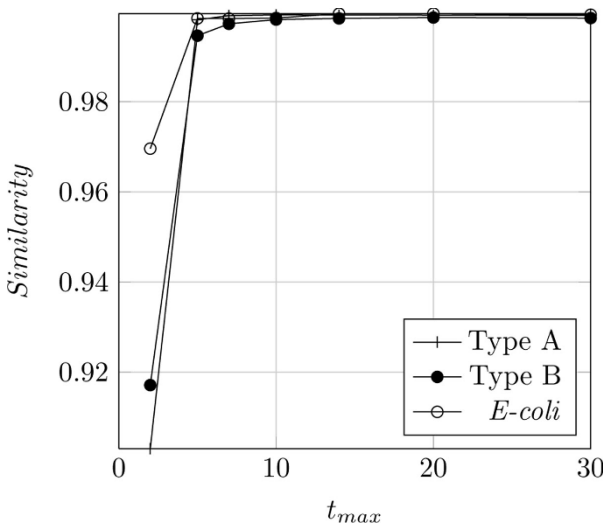

Figure 2 Similarity of y and $\tilde{y}$ versus t_{max}

Next, we apply sampling as described in Section *Sampling*. We did so for 100 different networks of Type A and B, and the *E-coli* network. For each random network we did T = 100 runs, for *E-coli* T = 1000. The results can be viewed in Table 1. We depict the percentage of solved networks, i.e. the portion of networks we found at least one valid $\mathbf{x} \in \Omega_{\mathbf{y}}$. Further, we give the average number of valid $\mathbf{x}$ and the average number of unique $\mathbf{x}$.

One can see from the results, that in general for most networks and $\mathbf{y}$s at least one preimage can be found. It is worth mentioning, that for the *E-coli* network every sampled solution was unique. This is due to the fact, that there exist a few inputs, who completely determine the output. The other input variables have then no influence and hence a marginal distribution of 0.5. Further, the results for the network of type B are much better than for type A. It seems that the marginal distributions for unate functions give better estimation of the actual distribution than the marginal distributions for non-unate functions.

Conclusions

In this work, we proposed a probabilistic algorithm to address the preimage problem of Boolean networks. This is of interest when designing experiments, in which certain regulators are supposed to be in a specific state. Performing a series of simulations with Random networks we showed, that the algorithm works not only for unate functions, of which most biologically motivated networks consist, but for any kind of Boolean functions. By replacing the fixed output values of the network by probabilities one can simply apply the algorithm to networks, whose designated output is described by probability distributions. Further, the algorithm may be easily adjusted to work on stochastic, e.g. Bayesian, networks, where the nodes contain only transition probabilities instead of Boolean function. Therefore, it is needed to adapt the update rules accordingly. It remains an open question, which influence topographic properties, such as number of layers and number of nodes in these layers, have to the performance of the proposed algorithms, since we only investigated networks which are similar to the regulatory network of *E-coli*.

Table 1 Simulation results for different networks

network	num of samples	solved	valid	Unique
Type A	1000	89%	608.81	4.43
Type B	1000	95.9%	270.74	68.60
E-coli	1000	98.6%	193.3	193.3

List of abbreviations

E-coli: Escherichia coli
BF: Boolean Function
BN: Boolean Network
Eq: Equation
LLR: Log-Likelihood Ratio
SPA: Sum-Product Algorithm

Competing interests

The authors declare that they have no competing interests.

Authors' contributions

Idea and Concept: JK, SS. Design of the overall project: MB. Scientific mentor of JK and SS: MB. Implementation and Evaluation: JK. Wrote Paper: JK and SS. All authors discussed the results and implications and commented on the manuscript at all stages.

Acknowledgements

The authors would like to thank Shrief Rizkalla for implementing parts of the simulation.

Declarations

This work was supported by the German research council " Deutsche Forschungsgemeinschaft" (DFG) under Grant Bo 867/25-2 and the Ulm University.
This article has been published as part of BMC Bioinformatics Volume 14 Supplement 10, 2013: Selected articles from the 10th International Workshop on Computational Systems Biology (WCSB) 2013: Bioinformatics. The full contents of the supplement are available online at http://www.biomedcentral.com/bmcbioinformatics/supplements/14/S10

Published: 12 August 2013

References

1. Kauffman SA, Peterson C, Samuelsson B, Troein C: **Random Boolean network models and the yeast transcriptional network.** *Proceedings of the National Academy of Sciences of the United States of America* 2003, **100(25)**:14796-14799.
2. Covert MW, Knight EM, Reed JL, Herrgard MJ, Palsson BO: **Integrating high-throughput and computational data elucidates bacterial networks.** *Nature* 2004, **429(6987)**:92-96.
3. Feist AM, Henry CS, Reed JL, Krummenacker M, Joyce AR, Karp PD, Broadbelt LJ, Hatzimanikatis V, Palsson BO: **A genome-scale metabolic reconstruction for Escherichia coli K-12 MG1655 that accounts for 1260 ORFs and thermodynamic information.** *Molecular Systems Biology* 2007, **3**:121.
4. Feuer R, Gottlieb K, Viertel G, Klotz J, Schober S, Bossert M, Sawodny O, Sprenger G, Ederer M: **Model-based analysis of an adaptive evolution**

experiment with Escherichia coli in a pyruvate limited continuous culture with glycerol. *EURASIP Journal on Bioinformatics and Systems Biology* 2012, **2012**:14.

5. Wuensche A: **The Ghost in the Machine: Basins of Attraction of Random Boolean Networks.** *Artificial Life III Proceedings, Santa Fe Institute Studies in the Sciences of Complexity* Addison-Wesley; 1994.
6. Akutsu T, Hayashida M, Zhang SQ, Ching WK, Ng MK: **Analyses and Algorithms for Predecessor and Control Problems for Boolean Networks of Bounded Indegree.** *Information and Media Technologies* 2009, **4(2)**:338-349.
7. Klotz JG, Schober S, Bossert M: **On the Predecessor Problem in Boolean Network Models of Regulatory Networks.** *International Journal of Computers and Their Applications* 2012, **19(2)**:93-100.
8. Kschischang F, Frey BJ, Loeliger HA: **Factor Graphs and the Sum-Product Algorithm.** *IEEE Transactions on Information Theory* 2001, **47**:498-519.
9. Gallager RG: *Low-Density Parity-Check Codes* Cambridge: M.I.T. Press; 1963.
10. Hagenauer J, Offer E, Papke L: **Iterative decoding of binary block and convolutional codes.** *Information Theory, IEEE Transactions on* 1996, **42(2)**:429-445.

doi:10.1186/1471-2105-14-S10-S4
Cite this article as: Klotz *et al.*: Computing preimages of Boolean networks. *BMC Bioinformatics* 2013 **14**(Suppl 10):S4.