Generating Realistic Large Bayesian Networks by Tiling

Ioannis Tsamardinos **Alexander Statnikov** **Laura E. Brown** **Constantin F. Aliferis**
Vanderbilt University, Department of Biomedical Informatics, 2209 Garland Avenue, Nashville, TN 37232-8340
ioannis.tsamardinos, alexander.statnikov, laura.e.brown, constantin.aliferis@vanderbilt.edu

Abstract

In this paper we present an algorithm and software for generating arbitrarily large Bayesian Networks by tiling smaller real-world known networks. The algorithm preserves the structural and probabilistic properties of the tiles so that the distribution of the resulting tiled network resembles the real-world distribution of the original tiles. By generating networks of various sizes one can study the behavior of Bayesian Network learning algorithms as a function of the size of the networks only while the underlying probability distributions remain similar. We demonstrate through empirical evaluation examples how the networks produced by the algorithm enable researchers to conduct comparative evaluations of learning algorithms on large real-world Bayesian networks.

Introduction

A Bayesian network (BN) is a mathematical construct that compactly represents a joint probability distribution P among a set variables $\mathcal{V}$. BNs are frequently employed for modeling domain knowledge in Decision Support Systems, particularly in medicine (Beinlich *et al.* 1989).

Learning a BN from observational data is an important problem that has been studied extensively during the last decade. One reason for this is because it can be used to automatically construct Decision Support Systems. In addition, while still controversial, learning BNs is being used for inferring probable causal relations since, under certain conditions (Spirtes, Glymour, & Scheines 2000) the edges in the graph of a Bayesian network have causal semantics (i.e., they represent direct causal influences). For example, in bioinformatics learning Bayesian networks have been used for the interpretation and discovery of gene regulatory pathways (Friedman *et al.* 2000).

Several algorithms for learning the complete BN from data have been proposed in the literature the last two decades (Spirtes, Glymour, & Scheines 2000; Cheng *et al.* 2002; Tsamardinos, Brown, & Aliferis 2006). In addition, algorithms have been proposed for learning interesting regions of BNs (local BN learning) such as the Markov Blanket of a target variable (Tsamardinos, Aliferis, & Statnikov 2003a) or the network structure around a target variable of interest (Tsamardinos *et al.* 2003b). The Markov Blanket is of particular interest for variable selection for classification as under certain conditions, it is the minimal-size, maximally-predictive variable set (Tsamardinos & Aliferis 2003).

The main technique for evaluating and comparing such algorithms is by simulation of data from a network of known structure. Then, it is easy to compare the reconstructed network as learnt by an algorithm with the true data-generating network to assess the quality of learning. For the results of the evaluations to carry to real-world data distributions the networks used for data simulations have to be representative of the real-world examples. Typically, the networks employed for the data simulation are extracted from real-world BN-based decision support systems.

While the above is the most common technique for evaluating learning algorithms, the size of the existing known BNs is relatively small in the order of at most a few hundred variables (see for example, the BN repository Elidan, 2001). Thus, typically learning algorithms were so far validated on relatively small networks (e.g., with less than 100 variables), such as the classical ALARM network (Beinlich *et al.* 1989) or other ”toy-networks”. Algorithms have also been developed to generate large random BNs. The BNGenerator system is one example for generating large random BNs from a uniform distribution (Ide, Cozman, & Ramos 2004). However, the BNGenerator system and other algorithms of this type do not provide any guarantees that these networks resemble the networks of the distributions likely to be encountered in practice (Aliferis & Cooper 1994). The emergence of datasets of very high-dimensionality poses significant challenges to the development of new learning algorithms. However, the means to experiment with realistic networks of the sizes that appear in several domains is currently impossible.

In this paper we present an algorithm and software for generating arbitrarily large Bayesian Networks by tiling smaller real-world known networks. The algorithm preserves the structural and probabilistic properties of the tiles so that the distribution of the resulting tiled network resembles the real-world distribution of the original tiles. The algorithm has already been proven valuable in conducting large scale evaluation and comparative studies in our previous work. We prove the theoretical properties of the BN generation process. In addition, we present illustrative computational experiments that demonstrate the utility of the method for studying BN learning algorithms in how the learning quality and time complexity vary as a function of the number of variables in the network, while the underlying probability distributions remain similar.

Generation of Realistic BNs by Tiling

We denote a variable with an upper-case letter (e.g., A, V_i) and a state or value of that variable by the same lower-case letter (e.g., a, v_i). We denote a set of variables by upper-case bold-face (e.g., $\mathbf{Z}$, $\mathbf{Pa_i}$) and we use the corresponding lower-case bold-face symbol to denote an assignment of state or value to each variable in the given set (e.g., $\mathbf{z}$, $\mathbf{pa_i}$). Important sets or mathematical concepts are denoted in calligraphic notation (e.g., $\mathcal{V}$, $\mathcal{E}$).

Definition 1. Let P be a discrete joint probability distribution of the random variables[1] in some set $\mathcal{V}$ and $\mathcal{G} = \langle \mathcal{V}, \mathcal{E} \rangle$ be a Directed Acyclic Graph (DAG). We call $\mathcal{N} = \langle \mathcal{G}, P \rangle = \langle \mathcal{V}, \mathcal{E}, P \rangle$ a (discrete) *Bayesian network* if $\langle \mathcal{G}, P \rangle$ satisfies the Markov Condition: every variable is independent of any subset of its non-descendant variables conditioned on its parents (Pearl 1988; Glymour & Cooper 1999; Spirtes, Glymour, & Scheines 2000).

We denote the set of the parents of variable V_i in the graph $\mathcal{G}$ as $\mathbf{Pa}_i^{\mathcal{G}}$. By utilizing the Markov Condition, it is easy to prove that for a Bayesian network $\langle \mathcal{G}, P \rangle$ the distribution P of the variables $\mathcal{V}$ can be factored as follows:

$$P(\mathcal{V}) = P(V_1, \ldots, V_n) = \prod_{V_i \in \mathcal{V}} P(V_i | \mathbf{Pa}_i^{\mathcal{G}}) \qquad (1)$$

To represent a Bayesian network the structure (i.e., the BN graph) and the joint probability distribution have to be encoded; for the latter, and according to the above equation, one is required to only specify the conditional probabilities $P(V_i = v_i | \mathbf{Pa}_i^{\mathcal{G}} = \mathbf{pa}_j)$ for each variable V_i, each possible value v_i of V_i, and each possible joint instantiation $\mathbf{pa}_j$ of its parents $\mathbf{Pa}_i^{\mathcal{G}}$.

The general problem we address in this paper is the generation of a BN $\mathcal{N} = \langle \mathcal{G}, P \rangle$ of arbitrary size such that its distribution P is "realistic" and representative of the type of distributions that are likely to be encountered in a specific domain (e.g., medicine). Moreover, we would like to generate sequences of BNs $\mathcal{N}^1, \ldots, \mathcal{N}^n$ of different sizes but with similar distributional properties. Such a sequence can then be used to study the behavior of a learning algorithm as a function of the size of the network.

The approach we take to solve the above problem is the following. We start with an available network $\mathcal{N}' = \langle \mathcal{V}', \mathcal{E}', P' \rangle$ (that is $\mathcal{G}' = \langle \mathcal{V}', \mathcal{E}' \rangle$) whose distribution is assumed to be a representative example of the distributions in the given domain and likely to be encountered in practice in data analysis problems. Such networks exist for several domains such as medicine, agriculture, weather forecasting, financial modeling, animal breeding, and biology to name a few (see BN site repository Elidan, 2001). We then create copies of these networks $\mathcal{N}_i = \langle \mathcal{V}_i, \mathcal{E}_i, P_i \rangle$ each with variable set $\mathcal{V}_i$ and edges $\mathcal{E}_i$ copies of $\mathcal{V}'$ and $\mathcal{E}'$. Finally, we interconnect the copies by a few additional edges and tile them together to create a new network $\mathcal{N} = \langle \mathcal{V}, \mathcal{E}, P \rangle$ with variables $\mathcal{V} = \cup_i \mathcal{V}_i$ and edges $\mathcal{E} = I \cup_i \mathcal{E}_i$, where I is the set of interconnecting edges. We will call $\mathcal{N}'$ the *generating network*, the subnetwork $\mathcal{N}_i$ the ith *tile*, and $\mathcal{N}$ the *output network* or simply the network.

The requirement of the above tiling process is that the (a) structural and probabilistic properties of $\mathcal{N}$ are similar to the generating network $\mathcal{N}'$ while (b) the tiles are probabilistically dependent of each other. The latter requirement is necessary or else each tile would not be connected to any other and could be learnt independently of any other tile. In that case, learning the output network would be similar to learning n times the generating network.

By the term structural properties we mean such quantities as the average or median number of parents or children (fan-in and fan-out degree), the average number of direct paths connecting two nodes, etc. It is unknown which of these quantities are important to a learning algorithm and should be maintained in the output network. However, by tiling several copies of a generating network $\mathcal{N}'$ to create the output network $\mathcal{N}$ and keeping the structure in each tile the same as in $\mathcal{N}'$ we expect that most of these quantities will remain the same. Of course, structural properties that depend on the number of variables in the network (e.g., the diameter of the graph) will not stay constant.

To maintain the probabilistic properties in the tiled network we impose the constraint that $P(\mathcal{V}_i) = P'(\mathcal{V}')$, i.e., we require that the marginal distribution of the variables $\mathcal{V}_i$ in tile i is the same as the distribution of the corresponding variables in the generating network. This way, the marginal distribution of each tile is guaranteed to have a realistic distribution. It is not trivial to satisfy this property since the addition of interconnecting edges may change the marginal distribution of the variables in a tile.

The TileBN Algorithm

We now describe the TileBN algorithm for generating large networks with the desired properties as described in the previous section. The algorithm accepts as input the originating network $\mathcal{N}'$, the number n of tiles to create, and the parameter *maxPa* (standing for maximum parents) that determines the connectivity between the tiles. It then creates copies of $\mathcal{N}'$ corresponding to tiles $\mathcal{N}_i$ as described in the previous section and interconnects them.

By $V_{i,j}$ we denote the variable in the ith tile $\mathcal{V}_i$ whose corresponding variable in the originating network is V_j'. Also, we extend our notation to denote by $\mathbf{Pa}_{i,j}^{\mathcal{G}}$ the parents of variable $V_{i,j}$. We will drop the superscript $\mathcal{G}$ when it is obvious by the presence of one or two subscripts whether we refer to the graph of the generating network $\mathcal{G}'$ or of the output network $\mathcal{G}$. Finally, whenever the distribution of variable $V_j \in \mathcal{V}'$ is equated with the distribution of $V_{i,j} \in V$ we mean that the probabilities of their corresponding values are the same: $P(V_j)$=$P(V_{i,j})$ implies $P(V_j = v_j)$=$P(V_{i,j} = v_j)$, for all values v_j of their common domains.

[1] Variables are interchangeably called nodes or vertices in the context of a Bayesian network.

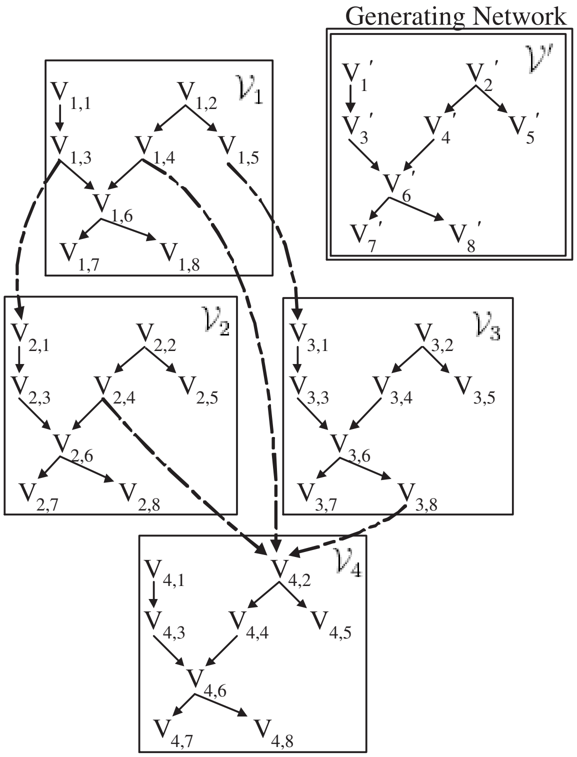


Figure 1: An example output of the TileBN algorithm. The generating network is Asia (Lauritzen & Spiegelhalter 1988) shown in the top right. The output network consists of four tiles of Asia with the addition of several interconnecting edges shown with the dashed edges.

Interconnecting the tiles properly is important for satisfying the property requirements of the algorithm. The tiles are topologically sorted according to their index, i.e., tile $\mathcal{N}_i$ is lower in the order than tile $\mathcal{N}_{i+1}$. TileBN then adds a number of randomly selected interconnecting edges. TileBN may add the edge $V_{p,k} \rightarrow V_{q,j}$ if the following three requirements are satisfied:

1. $p < q$, i.e., the edge goes from a variable to a variable belonging in a tile downstream.
2. $\mathbf{Pa}_j = \emptyset$, i.e., the node V'_j of the generating network that corresponds to the endpoint of the edge $V_{q,j}$ is a minimal node in the generating network (it has no parents).
3. If there exist another edge $V_{p',k'} \rightarrow V_{q,j'}$ to the same tile q then $j = j'$, i.e., for each tile q only one minimal node can receive tile interconnecting edges.

The first requirement guarantees that the interconnecting edges will not create any cycles and that the final resulting graph will be acyclic. The second and third requirements will be used to determine the probabilistic properties of the output network respectively. An example is shown in Figure 1. The generating network $\mathcal{V}$ is the Asia network (Lauritzen & Spiegelhalter 1988) shown in the top right. The output network has four tiles which are copies of the generating one, each tile shown within a box. The dashed edges are the tile interconnecting edges while the rest are copies of edges in the generating network.

TileBN is shown in Algorithm 1. To interconnect the tiles (Lines 4-8) a special node $V_{i,s}$ is selected from each tile $\mathcal{V}_i$, such that V'_s is minimal, to receive interconnecting edges. Then, the number of parents p_i of $V_{i,s}$ is found by sampling from a uniform discrete distribution $U(0, maxPa)$. Subsequently, p_i edges are added with end-points $V_{i,s}$ and start-points nodes randomly and uniformly selected from all tiles $\mathcal{V}_q$ with $q < i$. After the interconnecting edges are in place the structure of the output network is determined.

We now describe the determination of the conditional probabilities of the output network (and thus, of its distribution, lines 9-15). Let us now consider a node $V_{i,j}$ different from the special node $V_{i,s}$. $V_{i,j}$ then has parents $\mathbf{Pa}_{i,j}$ that are copies of $\mathbf{Pa}_j$ since no interconnecting edges has been added to it. We can then index the joint instantiations of both $\mathbf{Pa}_{i,j}$ and $\mathbf{Pa}_j$ the same way and specify that

$$P(V_{i,j}|\mathbf{Pa}_{i,j}) = P'(V'_j|\mathbf{Pa}_j) \qquad (2)$$

That is, the conditional probability tables for each variable $V_{i,j}$ such that V'_j is not minimal in the output network are copies of the conditional probability tables of the variable V'_j in the generating network.

For any node $V_{i,s}$ such that V'_s is minimal and had its parent set changed by the addition of interconnecting edges we set the conditional probabilities $P(V_{i,s}|\mathbf{Pa}_{i,s})$ so that the following condition holds:

$$\sum_{\mathbf{Pa}_{i,s}} P(V_{i,s}|\mathbf{Pa}_{i,s})P(\mathbf{Pa}_{i,s}) = P(V'_s) \qquad (3)$$

where the summation is over all joint instantiations of $\mathbf{Pa}_{i,s}$. We next show that the above process guarantees that $P(\mathcal{V}_i)$=$P'(\mathcal{V}')$ for each tile and present a procedure to select the conditional probabilities $P(V_{i,s}|\mathbf{Pa}_{i,s})$ so they satisfy Equation 3.

Maintaining the Probabilistic Properties While Tiling

We will use the notation $\sum_v$ to denote the sum over all values of v if it is a single variable, or over all joint instantiations of v if it is a set of variables. The notation $\prod_{V_j \in V}$ denotes a product of factors, one for each member of $\mathcal{V}$. Let $V_{i,s}$ denote the single node that may receive interconnecting edges in tile $\mathcal{V}_i$. Then,

$$P(\mathcal{V}_i) = \sum_{\mathbf{Pa}_{i,s}} P(\mathcal{V}_i|\mathbf{Pa}_{i,s})P(\mathbf{Pa}_{i,s})$$

and by factoring the probability $P(\mathcal{V}_i|\mathbf{Pa}_{i,j})$ in a topological order

$$= \sum_{\mathbf{Pa}_{i,s}} \prod_{V_{i,j} \in \mathcal{V}_i} P(V_{i,j}|\mathbf{Pa}_{i,j})P(\mathbf{Pa}_{i,s})$$

$$= \prod_{\substack{V_{i,j} \in \mathcal{V}_i \\ j \neq s}} P(V_{i,j}|\mathbf{Pa}_{i,j}) \sum_{\mathbf{Pa}_{i,s}} P(V_{i,s}|\mathbf{Pa}_{i,s})P(\mathbf{Pa}_{i,s})$$

Algorithm 1 TileBN Algorithm

```
procedure TileBN($\mathcal{N}' = \langle \mathcal{V}', \mathcal{E}', P' \rangle$, $n$, $maxPa$)
    $\mathcal{V}_i \leftarrow$ copy of $\mathcal{V}'$, $\mathcal{E}_i \leftarrow$ copy of $\mathcal{E}'$, $i = 1, \ldots, n$.
    % Initializing the output network $\langle \mathcal{V}, \mathcal{E}, P \rangle$
    $\mathcal{V} = \cup_i \mathcal{V}_i$,  $\mathcal{E} = \cup_i \mathcal{E}_i$
    % Add Interconnecting Edges
    for every tile $\mathcal{V}_i$, $i \geq 2$ do
        Select $V_{i,s}$ from $\mathcal{V}_i$ s.t. $V'_s$ is minimal
        $p_i \leftarrow UniformDiscrete(0, maxPa)$
        Add $p_i$ number of edges to $\mathcal{E}$ with end node
            $V_{i,s}$ and the start node selected randomly and
            uniformly from within any $\mathcal{V}_q$ with $q < i$
    end for
    % Determine Conditional Probability Tables
    for all $V_{i,j} \in \mathcal{V}$ do
        if $j \neq s$ then
            $P(V_{i,j}|\mathbf{Pa}_{i,j}) = P'(V'_j|\mathbf{Pa}_j)$
        else $(s = j)$
            $P(V_{i,s}|\mathbf{Pa}_{i,s})$ = SolveEq ($V_{i,s}$, $\mathbf{Pa}_{i,s}$)
        end if
    end for
    return $\mathcal{N} = \langle \mathcal{V}, \mathcal{E}, \prod_{V_{i,j} \in \mathcal{V}} P(V_{i,j}|\mathbf{Pa}_{i,j}) \rangle$
end procedure

function SolveEq($T$, $\mathbf{Pa}_T$)
    % Find a random solution for the condition of Eq. 3
    $a_p \leftarrow P(\mathbf{Pa}_T = p), \forall p$
    $b_t \leftarrow P(T = t), \forall t$
    $x'_{tp} \leftarrow UniformContinuous(0, 1), \forall t, p$
    Solve the system of equations for $r_t$ and $c_p$ $\forall t, p$
        $\sum_p a_p x'_{tp} r_t c_p = b_t, \forall t$
        $\sum_t x'_{tp} r_t c_p = 1, \forall p$
        Subject to the constraints
        $r_t \geq 0, c_p \geq 0, \forall t, p$
    return $\{P(T = t|\mathbf{Pa}_T = p) = x'_{tp} r_t c_p, \forall t, p\}$
end function
```

The last equation is because for every $V_{i,j}$ with $j \neq s$ the parents $\mathbf{Pa}_{i,j}$ are within the ith tile and so $\mathbf{Pa}_{i,j} \cap \mathbf{Pa}_{i,s} = \emptyset$. Also, $V_{i,j} \notin \mathbf{Pa}_{i,s}$ since the parents of $V_{i,s}$ are not within tile i. So, all factors $P(V_{i,j}|\mathbf{Pa}_{i,j}), j \neq s$ can be taken out of the sum. Finally, by combining the above result with Equations 2 and 3 we get that

$$P(\mathcal{V}_n) = \prod_{V_j \in \mathcal{V}', j \neq s} P'(V'_j|\mathbf{Pa}_j) P'(V'_s) = P'(\mathcal{V}')$$

Satisfying the Conditions for Tiling

We now present a method for satisfying the conditions sufficient for out method to produce a network with the desired properties. Whenever a node does not receive any interconnecting edges then its conditional probability tables are set according to Equation 2 by directly copying the corresponding table from the generating network.

However, for a node $V_{i,s}$ that does receive interconnecting edges it is not trivial to satisfy Equation 3. First notice that $P(\mathbf{Pa}_{i,s})$ can be calculated from the distribution of only the tiles $\mathcal{V}_q$ with $q < i$, i.e., from the tiles lower in the topological order, by utilizing the Markov Condition. For the first tile there are no incoming interconnecting edges to it and so $P(V_{1,s}) = P'(V'_s)$. For the second tile all members of $\mathbf{Pa}_{2,s}$ are in tile $\mathcal{V}_1$ and so $P(\mathbf{Pa}_{2,s})$ can be calculated from the generating network by running an inference algorithm (exact or approximate). Then, one can solve Equation 3 to determine $P(V_{2,s}|\mathbf{Pa}_{2,s})$ (as shown below). For the third tile $P(\mathbf{Pa}_{3,s})$ can be calculated by inference on the subnetwork of the first two tiles and so on. Given the above recursive method, the $P(\mathbf{Pa}_{i,s})$ can be considered fixed when one is trying to solve Equation 3 for a tile $\mathcal{V}_i$.

Let us now denote with $x_{tp} = P(V_{i,s} = v_t|\mathbf{Pa}_{i,s} = \mathbf{pa}^p_{i,s})$, where v_t is the tth value of $V_{i,s}$ and $\mathbf{pa}^p_{i,s}$ the pth joint instantiation of $\mathbf{Pa}_{i,s}$. Similarly, let $a_p = P(\mathbf{Pa}_{i,s} = \mathbf{pa}^p_{i,s})$ and $b_t = P(V'_s = v_t)$. Then, Equation 3 can be rewritten as the set of equations:

$$\sum_p a_p x_{tp} = b_t, \forall t$$

Also, for x_{tp} to be conditional probabilities they have to belong in $[0, 1]$ and the following holds for any p:

$$\sum_t x_{tp} = 1, \forall p.$$

This gives us a set of $|V_{i,s}|$+$|\mathbf{Pa}_{i,s}|$ equations and $|V_{i,s}|$×$|\mathbf{Pa}_{i,s}|$ unknowns x_{tp}, where $|S|$ denotes the cardinality of the domain of the variable or set of variables S.

One could solve this underconstrained system of linear equations for x_{tp}. In particular, we would like to select randomly a solution to the system of equation as to not favor a specific type of networks and conditional probability distribution tables. Most linear equation solvers will arbitrarily, *but not randomly*, select a solution for the system (e.g. solve for the first $|V_{i,s}| + |\mathbf{Pa}_{i,s}|$ values of the unknowns and set the rest to zeros).

To find a random solution to the system of equations we take the following approach. We randomly select values x'_{tp} uniformly from $[0, 1]$. The random values will most probably not be solutions to the equations. However, they can become a solution if they are appropriately rescaled. We express the rescaling factors as the unknown quantities r_t and c_p and rewrote the equations as:

$$\sum_p a_p x'_{tp} r_t c_p = b_t, \forall t$$

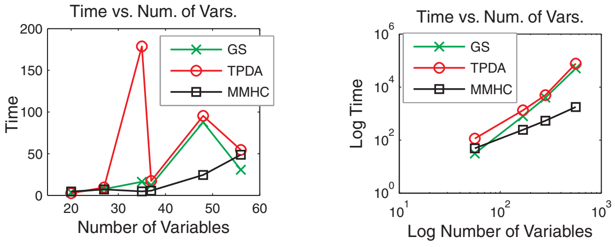
$$\sum_t x'_{tp} r_t c_p = 1, \forall p$$

i.e., we replaced each unknown quantity x_{tp} with the quantity $x'_{tp} r_t c_p$. We also need to introduce the constraints:

$$r_t c_p \geq 0, \forall t, p, \text{ or } (r_t \geq 0, c_p \geq 0, \forall p, t)$$

to ensure that $x_{tp} = x'_{tp} r_t c_p \geq 0$ (x_{tp} is automatically ≤ 1 since $\sum_t x_{tp} = 1, \forall p$)

Now the underconstrained linear system has become a quadratic system of $|V_{i,s}| + |\mathbf{Pa}_{i,s}|$ equations and $|V_{i,s}| + |\mathbf{Pa}_{i,s}|$ unknowns (the quantities r_t and c_p), and $|V_{i,s}| \times$



(a) (b)

Figure 2: Comparison of execution time of several algorithms on different networks. (a) Time of the Greedy Search (GS), Three Phase Dependency Analysis (TPDA), and Max-Min Hill-Climbing (MMHC) algorithms over several non-tiled networks: Child, Insurance, Mildew, Alarm, Barley, and HailFinder. (b) Time of the GS, TPDA, and MMHC algorithms over 4 versions of the Hailinder network: the original and tiled versions consisting of 3, 5, and 10 tiles.

$|\mathbf{Pa}_{i,s}|$ nonlinear constraints which can be solved by any standard method for solving constrained quadratic systems of equations. The above procedure is shown in Algorithm 1, procedure SolveEq.

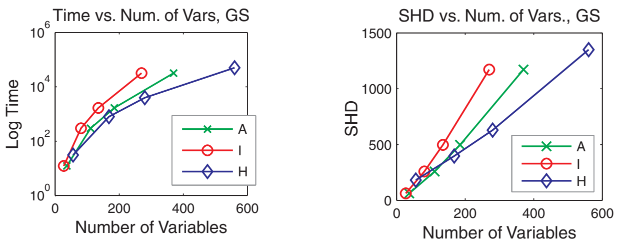
Application and Analysis with TileBN

The TileBN algorithm is implemented in Mathworks Matlab (The Mathworks Inc. 2003) and is publicly available and maintained as part of the Causal Explorer system (Aliferis *et al.* 2003, http://www.dsl-lab.org/causal_explorer). The generating BN is specified in HUGIN (Jensen *et al.* 2002) format, and a simple parser was written to read HUGIN BNs in a custom Matlab format. We used an iterative solver from Matlab Optimization toolbox employing Gauss-Newton algorithm with BFGS updating scheme.

We have found TileBN a useful algorithm for simulating large BNs in several occasions in our prior work. We used an earlier version of TileBN for comparing different Markov Blanket learning algorithms in large (5000 variables) networks (Tsamardinos, Aliferis, & Statnikov 2003a), comparing many BN learning algorithms (Brown, Tsamardinos, & Aliferis 2005; Tsamardinos, Brown, & Aliferis 2006), and examining the time efficiency and quality of a local BN learning algorithm to reconstruct local regions in a large (10000 variable) network (Tsamardinos *et al.* 2003b).

To illustrate the use and benefit of the TileBN we present a series of experiments over several tiled networks. The experiments are run using several algorithms: Max-Min Hill Climbing (MMHC) (Tsamardinos, Brown, & Aliferis 2006), Greedy Search (GS), and Three Phase Dependency Analysis (TPDA) (Cheng *et al.* 2002). All algorithms' performance results on a given network were averaged over 5 datasets sampled from the distribution of that network. The datasets all have 1000 samples.

We first examine how the execution time of an algorithm scales as the number of variables in the network to be learned increases. The execution time is plotted over several networks (see Figure 2(a)). Specifically, the following networks were used in the analysis, listed in order of increasing number of variables (number of variables given in parenthesis): Child (20), Insurance (27), Mildew (35), Alarm (37), Barley (48), and HailFinder (56). From this presentation of the data it is difficult to identify trends as the number of variables increase. This is due in part to the fact that the difficulty of learning a network is not only a function of its size, but also of its distributional properties which varies among arbitrary networks. However, when the execution time is plotted across tiled networks of different sizes trends in the performance become observable as the distributional properties remain similar. In Figure 2(b), the same algorithms are compared for different networks that contain 1, 3, 5, and 10 tiles of the original HailFinder network.

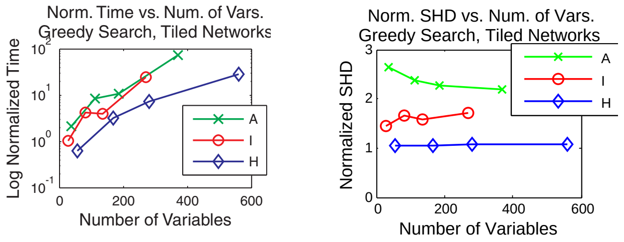


(a) (b)

Figure 3: The performance of the Greedy Search (GS) algorithm over three different series of tiled networks, Alarm (A), Insurance (I), and HailFinder (H), each series containing 1, 3, 5, and 10 tiled networks. (a) Time of the GS algorithm. (b) SHD of the GS algorithm.

Another example is shown in Figure 3. Specifically, Figure 3(a) plots the time of the Greedy Search (GS) algorithm for three series each containing 1,3,5, and 10 tiles of the following networks respectively: Alarm (A), Insurance (I), and HailFinder (H). This plot shows the time complexity of the GS algorithm to be increasing more than linearly with the number of variables (notice the y-axis scale is logarithmic in this graph). Similarly, a graph plotting the Structural Hamming Distance (SHD) versus the number of variables in Figure 3(b). SHD is a measure of the structural quality of the learned network that accounts for missing and extra edges as well as errors of edge orientation; for more information on this measure see (Tsamardinos, Brown, & Aliferis 2006). The number of errors (i.e., the SHD) increases linearly with the number of variables.

In addition, the relative performance of one algorithm versus another can be studied. The three series of tiled networks from above (Alarm, Insurance, HailFinder) are also used in this example. In Figure 4(a), the normalized time of the GS algorithm is plotted for each of the series of networks. From this figure you can conclude that the ratio of time complexities of GS with MMHC increases exponentially with the number of variables. A similar graph is plotted for the normalized SHD as well in Figure 4(b). Here the relative quality between the two algorithms is close to constant as the number of variables increase. Thus, while the relative learning quality between the algorithm remains constant, MMHC is



(a)

(b)

Figure 4: The comparative performance of the Greedy Search (GS) versus MMHC algorithms over three different series of tiled networks, Alarm (A), Insurance (I), and HailFinder (H), each series containing 1, 3, 5, and 10 tiled networks. (a) Normalized Time of the GS algorithm. (b) Normalized SHD of the GS algorithm.

more time efficient as the number of variables increases.

Two demonstrated uses of the TileBN networks are for studying how the learning quality and time complexity scale and algorithmic comparisons. Whether the repetition of the network structure affects and/or biases these evaluations is still an open question to be pursued in further research.

Conclusions

In this paper we presented an algorithm called TileBN, publicly available in the Causal Explorer library, for generating arbitrarily large Bayesian Networks by tiling smaller real-world known networks. The algorithm preserves the structural and probabilistic properties of the tiles so that the distribution of the resulting tiled network resembles the real-world distribution of the original tiles. This is an advancement over other random BN generation methods that provide no guarantees about their output networks. We also presented illustrative examples how the algorithm can be used to study the behavior of BN learning algorithms as a function of the size of the network learnt.

Aknowledgements

The first and last author were supported by NLM grant LM-7948-01; NLM grant T15 LM07450-01 supports the third.

References

Aliferis, C. F., and Cooper, G. 1994. An evaluation of an algorithm for inductive learning of bayesian belief networks using simulated data sets. In *Proceedings of Uncertainty in Artificial Intelligence*, 8–14.

Aliferis, C. F.; Tsamardinos, I.; Statnikov, A.; and Brown, L. E. 2003. Causal explorer: A causal probabilistic network learning toolkit for biomedical discovery. In *International Conference on Mathematics and Engineering Techniques in Medicine and Biological Sciences (METMBS '03)*, 371–376.

Beinlich, I.; Suermondt, G.; Chavez, R.; and Cooper, G. 1989. The alarm monitoring system: A case study with two probabilistic inference techniques for belief networks. In *2nd European Conference in Artificial Intelligence in Medicine*, 247–256.

Brown, L.; Tsamardinos, I.; and Aliferis, C. 2005. A comparison of novel and state-of-the-art polynomial bayesian network learning algorithms. In *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI)*, 739–745. AAAI Press.

Cheng, J.; Greiner, R.; Kelly, J.; Bell, D.; and Liu, W. 2002. Learning bayesian networks from data: An information-theory based approach. *Artificial Intelligence* 137:43–90.

Elidan, G. 2001. Bayesian network repository. http://www.cs.huji.ac.il/labs/compbio/Repository/.

Friedman, N.; Linial, M.; Nachman, I.; and Pe'er, D. 2000. Using bayesian networks to analyze expression data. *Journal of Computational Biology* 7(3/4):601–620.

Glymour, C., and Cooper, G. 1999. *Computation, Causation, and Discovery*. Menlo Park, CA: AAAI Press / MIT Press.

Ide, J.; Cozman, F.; and Ramos, F. 2004. Generating random bayesian networks with constraints on induced width. In *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI-04)*, 323–327. IOS Press.

Jensen, F.; Kjærulff, U.; Lang, M.; and Madsen, A. 2002. Hugin - the tool for bayesian networks and influence diagrams. In *First European Workshop on Probabilistic Graphical Models*, 212–221.

Lauritzen, S., and Spiegelhalter, D. 1988. Local computations with probabilities on graphical structures and their application on expert systems. *Journal of the Royal Statistical Society B* 50:157–224.

Pearl, J. 1988. *Probabilistic Reasoning in Intelligent Systems*. San Mateo, CA: Morgan Kaufmannn Publishers.

Spirtes, P.; Glymour, C.; and Scheines, R. 2000. *Causation, Prediction, and Search*. Cambridge, MA: MIT Press, 2nd edition.

The Mathworks Inc. 2003. Matlab. http://www.mathworks.com.

Tsamardinos, I., and Aliferis, C. F. 2003. Towards principled feature selection: Relevancy, filters and wrappers. In *Ninth International Workshop on Artificial Intelligence and Statistics (AI and Stats 2003)*.

Tsamardinos, I.; Aliferis, C.; and Statnikov, A. 2003a. Time and sample efficeint discovery of markov blankets and direct causal relations. In *Proceedings of Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 673–678.

Tsamardinos, I.; Aliferis, C.; Statnikov, A.; and Brown, L. 2003b. Scaling-up bayesian network learning to thousands of variables using local learning techniques. Technical Report TR-03-02, Vanderbilt University.

Tsamardinos, I.; Brown, L.; and Aliferis, C. 2006. The max-min hill-climbing bayesian network structure learning algorithm. *To Appear: Machine Learning*.