Bayesian Network Constraint-Based Structure Learning Algorithms: Parallel and Optimised Implementations in the \texttt{bnlearn} R Package

Marco Scutari
University College London (UCL)

Abstract

It is well known from the literature that the problem of learning the structure of Bayesian networks is very hard to tackle: its computational complexity is super-exponential in the number of nodes in the worst case and polynomial in most real-world scenarios.

Efficient implementations of score-based structure learning benefit from past and current research in optimisation theory, which can be adapted to the task by using the network score as the objective function to maximise. This is not true for approaches based on conditional independence tests, called constraint-based learning algorithms. The only optimisation in widespread use, backtracking, leverages the symmetries implied by the definitions of neighbourhood and Markov blanket.

In this paper we show that backtracking degrades the stability of Bayesian network structure learning, for little gain in terms of speed. As an alternative, we introduce a general framework to parallelise constraint-based structure learning algorithms, and we investigate its performance using the \texttt{bnlearn} R package on four reference networks and two real-world data sets from genetics and systems biology. We show that on modern multi-core or multiprocessor hardware the proposed approach is always preferable over backtracking, which was developed when single-processor machines were the norm.

\textit{Keywords:} Bayesian networks, structure learning, parallel programming, R.

1. Background and Notations

Bayesian networks (BNs) are a class of \textit{graphical models} (Pearl 1988; Koller and Friedman 2009) composed by a set of random variables $X = \{X_i, i = 1, \ldots, m\}$ and a \textit{directed acyclic graph} (DAG), denoted $G = (V, A)$. The probability distribution of $X$ is called the \textit{global distribution} of the data, while those associated with each $X_i$ are called \textit{local distributions}. 
Each node in $V$ is associated with one variable, and they are referred to interchangeably. The directed arcs in $A$ that connect them represent direct stochastic dependencies; so if there is no arc connecting two nodes, the corresponding variables are either marginally independent or conditionally independent given (a subset of) the rest. As a result, each local distribution depends only on a single node $X_i$ and on its parents (i.e. the nodes $X_j$ such that $X_j \rightarrow X_i$, here denoted $\Pi_{X_i}$):

$$P(X) = \prod_{i=1}^{m} P(X_i | \Pi_{X_i}).$$

(1)

Common choices for the local and global distributions are multinomial variables (discrete BNs, Heckerman, Geiger, and Chickering 1995); univariate and multivariate normal variables (Gaussian BNs, Geiger and Heckerman 1994); and, less frequently, a combination of the two (conditional Gaussian (CG) BNs, Lauritzen and Wermuth 1989). In the first case, the parameters of interest are the conditional probabilities associated with each variable, represented as conditional probability tables (CPTs); in the second, the parameters of interest are the partial correlation coefficients between each variable and its parents. As for CG BNs, the parameters of interest are again partial correlation coefficients, computed for each node conditional on its continuous parents for each configuration of the discrete parents.

The key advantage of the decomposition in Equation (1) is to make local computations possible for most tasks, using just a few variables at a time regardless of the magnitude of $m$. A related quantity that works to the same effect is the Markov blanket of each node $X_i$, defined as the set $B(X_i)$ of nodes which graphically separates $X_i$ from all other nodes $V \setminus \{X_i, B(X_i)\}$ (Pearl 1988, p. 97). In BNs such a set is uniquely identified by the parents ($\Pi_{X_i}$), the children (i.e. the nodes $X_j$ such that $X_i \rightarrow X_j$) and the spouses of $X_i$ (i.e. the nodes that share a child with $X_i$). By definition, $X_i$ is independent of all other nodes given $B(X_i)$, thus making them redundant for inference on $X_i$.

The task of fitting a BN is called learning and is generally implemented in two steps.

The first is called structure learning, and consists in finding the DAG that encodes the conditional independencies present in the data. This has been achieved in the literature with constraint-based, score-based and hybrid algorithms; for an overview see Koller and Friedman (2009) and Nagarajan, Scutari, and Lèbre (2013). Constraint-based algorithms are based on the seminal work of Pearl on causal graphical models and his Inductive Causation algorithm (IC, Verma and Pearl 1991), which provides a framework for learning the DAG of a BN using conditional independence tests under the assumption that graphical separation and probabilistic independence imply each other (the faithfulness assumption). Tests in common use are the mutual information test (for discrete BNs) and the exact Student’s $t$ test for correlation (for GBNs). Score-based algorithms represent an application of heuristic optimisation techniques: each candidate DAG is assigned a network score reflecting its goodness of fit, which the algorithm then attempts to maximise. BIC and posterior probabilities arising from various priors are typical choices. Hybrid algorithms use both conditional independence tests and network scores, the former to reduce the space of candidate DAGs and the latter to identify the optimal DAG among them. Some examples are PC (Spirtes, Glymour, and Scheines 2000), Grow-Shrink (GS; Margaritis 2003), Incremental Association (IAMB; Tsamardinos, Aliferis, and Statnikov 2003), Interleaved IAMB (Inter-IAMB; Yaramakala and Margaritis 2005), hill-climbing and tabu search (Russell and Norvig 2009), Max-Min Parents & Children (MMPC; Tsamardinos, Brown, and Aliferis 2006) and the Semi-Interleaved HITON-PC.
The second step is called parameter learning and, as the name suggests, deals with the estimation of the parameters of the global distribution. Since the graph structure is known from the previous step, this can be done efficiently by estimating the parameters of the local distributions.

Most problems in BN theory have a computational complexity that, in the worst case, scales exponentially with the number of variables. For instance, both structure learning (Chickering 1996; Chickering, Geiger, and Heckerman 1994) and inference (Cooper 1990; Dagum and Luby 1993) are NP-hard and have polynomial complexity even for sparse networks. This is especially problematic in high-dimensional settings such as genetics and systems biology, in which BNs are used for the analysis of gene expressions (Friedman 2004) and protein-protein interactions (Jansen, Yu, Greenbaum, Kluger, Krogan, Chung, Emili, Snyder, Greenblatt, and Gerstein 2003; Sachs, Perez, Pe’er, Lauffenburger, and Nolan 2005); for integrating heterogeneous genetic data (Chang and McGeachie 2011); and to determine disease susceptibility to anemia (Sebastiani, Ramoni, Nolan, Baldwin, and Steinberg 2005) and hypertension (Malovini, Nuzzo, Ferrazzi, Puca, and Bellazzi 2009).

Even though algorithms in recent literature are designed taking scalability into account, it is often impractical to learn BNs from data containing more than a few hundreds of variables without restrictive assumptions on either the structure of the DAG or the nature of the local distributions. Two ways to address this problem are:

1. **optimisations**: reducing the number of conditional independence tests and network scores computed from the data, either by skipping redundant ones or by limiting local computations to a few variables;

2. **parallel implementations**: splitting learning across multiple processors or cores to make better use of modern multi-core and multiprocessor hardware.

As far as score-based learning algorithms are concerned, both possibilities have been explored and a wide range of solutions proposed, from efficient caching using decomposable scores (Daly and Shen 2007), to parallel meta-heuristics (Rauber and Rünger 2010) and integer programming (Cussens 2011). The same cannot be said of constraint-based algorithms; even recent ones such as SI-HITON-PC, while efficient, are still implemented with basic backtracking as the only optimisation. We will examine them in Section 2, arguing that they provide only modest speed gains and increase the variability of the learned DAGs by introducing spurious dependencies between the variables. As an alternative, in Section 3 we propose a general framework to create parallel implementations of constraint-based algorithms that scale well on large data sets and do not suffer from this problem.

### 2. Constraint-Based Structure Learning and Backtracking

All constraint-based structure learning algorithms share a common three-phase structure inherited from the IC algorithm through PC and GS; it is illustrated in Algorithm 1. The first, optional, phase consists in learning the Markov blanket of each node to reduce the number of candidate DAGs early on. Any algorithm for learning Markov blankets can be plugged in step 1 and extended into a full BN structure learning algorithm, as originally suggested in
Margaritis (2003) for GS. Once all Markov blankets have been learned, they are checked for consistency (step 2) using their symmetry; by definition $X_i \in B(X_j) \Leftrightarrow X_j \in B(X_i)$. Asymmetries are corrected by treating them as false positives and removing the offending nodes from each other’s Markov blankets.

The second phase learns the skeleton of the DAG, that is, it identifies which arcs are present in the DAG modulo their direction. This is equivalent to learning the neighbours $\mathcal{N}(X_i)$ of each node: its parents and children. As illustrated in step 3, the absence of a set of nodes $S_{X_i,X_j}$ that separates a particular pair $X_i,X_j$ implies that either $X_i \to X_j$ or $X_j \to X_i$. Separating sets are considered in order of increasing size to keep computations as local as possible. Furthermore, if $B(X_i)$ and $B(X_j)$ are available from steps 1 and 2 the search can be greatly reduced because $\mathcal{N}(X_i) \subseteq B(X_i)$. On the one hand, if $X_j \notin B(X_i)$ by definition $X_i$ is separated from $X_j$ by $S_{X_i,X_j} = B(X_i)$. On the other hand, if $X_j \in B(X_i)$ number of candidate sets can be greatly reduced because $S_{X_i,X_j} \subseteq B(X_i) \setminus X_j$ and $S_{X_i,X_j} \subseteq B(X_j) \setminus X_i$; both sets are typically much smaller than $\mathbf{V}$. With the exception of the PC algorithm, which is structured exactly as described in step 3, constraint-based algorithms learn the skeleton by learning each $\mathcal{N}(X_i)$ and then enforcing symmetry (step 4).

Finally, in the third phase arc directions are established as in Meek (1995). It is important to note that, for some arcs, both directions are equivalent in the sense that they identify equivalent class containing multiple DAGs. Such a class is uniquely identified by the skeleton learned in steps 3 and 4, and by the $v$-structures $\mathcal{V}_i$ of the form $X_i \to X_k \leftarrow X_j, X_i \notin \mathcal{N}(X_j)$ learned in step 5 (Chickering 1995). Additional arc directions are inferred indirectly in step 6 by ruling out those that would introduce additional $v$-structures (which would have been identified in step 5) or cycles (which are not allowed in DAGs).

Even in such a general form, we can see that Algorithm 1 performs many checks for graphical separation that are redundant given the symmetry of the $B(X_i)$ and the $\mathcal{N}(X_i)$. Intuitively, once we have concluded that $X_j \notin B(X_i)$ there is no need to check whether $X_i \in B(X_j)$ in step 1; and similar considerations can be made for neighbours in step 3. In practice, this translates to redundant independence tests computed on the data. To reduce the number of such tests, it is common practice to assume $X_1, \ldots, X_m$ are processed sequentially and use backtracking in steps 1 and 3 as follows:

- If $X_j \notin B(X_i), i < j$, then do not consider $X_i$ for inclusion in $B(X_j)$; and if $X_j \notin \mathcal{N}(X_i)$, then do not consider $X_i$ for inclusion in $\mathcal{N}(X_j)$.

- If $X_j \in B(X_i)$, then consider $X_i$ for inclusion in $B(X_j)$ by initialising $B(X_j) = \{X_i\};$ and if $X_j \in \mathcal{N}(X_i)$ then initialise $\mathcal{N}(X_j) = \{X_i\}$. Not that in both cases $X_i$ can be discarded in the process of learning $B(X_j)$ and $\mathcal{N}(X_j)$.

Such a strategy is described in detail, for example, in Tsamardinos et al. (2006, p. 46). It is important to note that, in contrast with enforcing symmetry by construction (e.g. $X_j \notin B(X_i) \Rightarrow X_i \notin B(X_j)$ and $X_j \in B(X_i) \Rightarrow X_i \in B(X_j)$), this heuristic does not automatically introduce a false positive or a false negative in the learning process for every type I or type II error in the independence tests. Even so, backtracking has the undesirable effect of making structure learning depend on the order the variables are stored in the data set, which has been shown to increase errors in the PC algorithm (Colombo and Maathuis 2013). In addition,
Algorithm 1 A Template for Constraint-Based Structure Learning Algorithms

Input: a data set containing the variables \(X_i, i = 1, \ldots, m\).

Output: a completed partially directed acyclic graph.

Phase 1: learning Markov blankets (optional).

1. For each variable \(X_i\), learn its Markov blanket \(B(X_i)\).

2. Check whether the Markov blankets \(B(X_i)\) are symmetric, e.g. \(X_i \in B(X_j) \iff X_j \in B(X_i)\). Assume that nodes for whom symmetry does not hold are false positives and drop them from each other’s Markov blankets.

Phase 2: learning neighbours.

3. For each variable \(X_i\), learn the set \(N(X_i)\) of its neighbours (i.e. the parents and the children of \(X_i\)). Equivalently, for each pair \(X_i, X_j, i \neq j\) search for a set \(S_{X_i,X_j} \subset V\) (including \(S_{X_i,X_j} = \emptyset\)) such that \(X_i\) and \(X_j\) are independent given \(S_{X_i,X_j}\) and \(X_i, X_j \notin S_{X_i,X_j}\). If there is no such a set, place an undirected arc between \(X_i\) and \(X_j\) (\(X_i - X_j\)). If \(B(X_i)\) and \(B(X_j)\) are available from points 1 and 2, the search for \(S_{X_i,X_j}\) can be limited to the smallest of \(B(X_i) \setminus X_j\) and \(B(X_j) \setminus X_i\).

4. Check whether the \(N(X_i)\) are symmetric, and correct asymmetries as in step 2.

Phase 3: learning arc directions.

5. For each pair of non-adjacent variables \(X_i\) and \(X_j\) with a common neighbour \(X_k\), check whether \(X_k \in S_{X_i,X_j}\). If not, set the direction of the arcs \(X_i - X_k\) and \(X_k - X_j\) to \(X_i \rightarrow X_k\) and \(X_k \leftarrow X_j\) to obtain a v-structure \(V_l = \{X_i \rightarrow X_k \leftarrow X_j\}\).

6. Set the direction of arcs that are still undirected by applying the following two rules recursively:
   
   (a) If \(X_i\) is adjacent to \(X_j\) and there is a strictly directed path from \(X_i\) to \(X_j\) (a path leading from \(X_i\) to \(X_j\) containing no undirected arcs) then set the direction of \(X_i - X_j\) to \(X_i \rightarrow X_j\).
   
   (b) If \(X_i\) and \(X_j\) are not adjacent but \(X_i \rightarrow X_k\) and \(X_k - X_j\), then change the latter to \(X_k \rightarrow X_j\).

Return the resulting completed partially directed acyclic graph.

backtracking provides only a modest speed increase compared to a parallel implementation of steps 1-4; we will compare the respective running times in Section 3.

In the remainder of this section we will focus on the effects of backtracking on learning the skeleton of the DAG, because steps 1-4 comprise the vast majority of the overall conditional independence tests and thus control most of the variability of the DAG. To investigate it, we used the \texttt{bnlearn} R package (Scutari 2010) and 5 reference BNs of various size and complexity from \url{http://www.bnlearn.com/bnrepository}:
ALARM (Beinlich, Suermontd, Chavez, and Cooper 1989), with 37 nodes, 46 arcs and $p = 509$ parameters;

- HEPAR II (Onisko 2003), with 70 nodes, 123 arcs and $p = 1453$ parameters;

- ANDES (Conati, Gertner, VanLehn, and Druzdzel 1997) with 223 nodes, 338 arcs and $p = 1157$ parameters;

- LINK (Jensen and Kong 1999) with 724 nodes, 1125 arcs and $p = 14211$ parameters;

- MUNIN (Andreassen, Jensen, Andersen, Falck, Kjærulff, Woldbye, Sørensen, Rosenfalck, and Jensen. 1989) with 1041 nodes, 1397 arcs with $p = 80592$ parameters.

Simulations were performed on a cluster of 7 dual AMD Opteron 6136, each with 16 cores and 78GB of RAM, with R 3.1.0 and bnlearn 3.5.

For each BN, we considered 6 different sample sizes ($n = 0.1p, 0.2p, 0.5p, p, 2p, 5p$), chosen as multiples of $p$ to facilitate comparisons between networks of such different complexity; and 4 different constraint-based structure learning algorithms (GS, Inter-IAMB, MMPC, SI-HITON-PC). Since all reference BNs are discrete, we used the asymptotic $\chi^2$ mutual information test with $\alpha = 0.01$. For each combination of BN, sample size and algorithm we repeated the following simulation 20 times. First, we loaded the BN from the RDA file downloaded from the repository (alarm.rda below) and generated a sample of the appropriate size with rbn.

```r
> library(bnlearn)
> load("alarm.rda")
> sim = rbn(bn, n = round(0.1 * npars(bn)))
```

From that sample, we learned the skeleton of the DAG with (optimized = TRUE) and without backtracking (optimized = FALSE).

```r
> skel.orig = skeleton(si.hiton.pc(sim, test = "mi", alpha = 0.01,
+                  optimized = FALSE))
> skel.back = skeleton(si.hiton.pc(sim, test = "mi", alpha = 0.01,
+                  optimized = TRUE))
```

Subsequently, we reversed the order of the columns in the data to investigate whether this results in a different skeleton.

```r
> revsim = sim[, rev(seq(ncol(sim)))]
```

After learning the skeleton with (rskel.back) and without backtracking (rskel.orig) from revsim, we compared the output with that from sim using Hamming distance (Jungnickel 2008).

```r
> hamming(skel.orig, rskel.orig)
[1] 0
> hamming(skel.back, rskel.back)
[1] 10
```
Figure 1: Hamming distance between skeletons learned for the ALARM, ANDES, HEPAR II, LINK and MUNIN reference BNs before and after reversing the ordering of the variables, for various n/p ratios and algorithms. Blue boxplots correspond to structure learning without backtracking, green boxplots to learning with backtracking.
Ideally, skel.orig and rskel.orig should be identical and therefore their Hamming distance should be zero. This may not be the case for BNs with deterministic 0-1 nodes, whose structure is unlikely to be learned correctly by any of the considered algorithms; or when conditional independence tests are biased and have low power because of small sample sizes. The difference between the Hamming distance of skel.orig and rskel.orig and that of skel.back and rskel.back gives an indication of the dependence on the ordering of the variables introduced by backtracking.

The results of this simulation are shown in Figure 1. With the exception of ALARM, ANDES, HEPAR II for the GS algorithm, Hamming distance is always greater when backtracking is used. This is true for all the considered sample sizes. In fact, Hamming distance does not appear to converge to zero as sample size increases; on the contrary, large samples make even weak dependencies detectable and thus increase the chances of getting different skeletons. This trend is consistently more marked when using backtracking, as is the range of observed Hamming distances in each configuration of BN, sample size and learning algorithm. The combination of this two facts suggests that backtracking does indeed make learning dependent on the order in which the variables are considered; and that it increases the variability of the learned structure.

3. A Framework for Parallel Constraint-Based Learning

Constraint-based algorithms as described in Algorithm 1 display coarse-grained parallelism: different parts need to be synchronised only three times, in steps 2, 4 and 6. Steps 1, 3 and 5 are embarrassingly parallel, because each $B(X_i)$, each $N(X_i)$ and each $V_l$ can be learned independently from the others. Step 6 on the other hand is inherently sequential because of its iterative formulation. Parallelising Algorithm 1 on a step-by-step basis is therefore very convenient. As shown in Figure 2, the implementation still follows the same steps; and it can scale efficiently because computationally intensive steps can be partitioned in as many as parts as there are variables. Splitting the tests in large batches corresponding to the $B(X_i)$, $N(X_i)$ and $V_l$ also reduces the amount of information exchanged by different parts of the implementation, reducing overhead.

3.1. Simulations on the Reference BNs

All constraint-based learning algorithms in bnlearn have such a parallel implementation thanks to the infrastructure provided by the parallel package (R Core Team 2014). The master R process controls the learning process and distributes the $B(X_i)$, $N(X_i)$ and $V_l$ to the slave processes, executing only steps 2, 4 and 6 itself. Consider, for example, the Inter-IAMB algorithm and the data set generated from the ALARM reference BN shipped with bnlearn.

```r
> data(alarm)
> library(parallel)
> cl = makeCluster(2)
> res = inter.iamb(alarm, cluster = cl)
> unlist(clusterEvalQ(cl, test.counter()))
[1] 3637 3743
> stopCluster(cl)
```
After generating a cluster cl with 2 slave processes with `makeCluster`, we passed it to `inter.iamb` via the `cluster` argument. As we can see from the output of `clusterEvalQ`, the first slave process performed 3637 (49.3\%) conditional tests, and the second 3743 (50.7\%). The difference in the number of tests between the two slaves is due to the topology of the BN: the $B(X_i)$ and $N(X_i)$ have different sizes and therefore require different numbers of tests to learn. This in turn also affects the number of tests required to learn the v-structures $V_i$.

Increasing the number of slave processes reduces the number of tests performed by each of them, further increasing the overall performance of the algorithm.

```r
> cl = makeCluster(3)
> res = inter.iamb(alarm, cluster = cl)
> unlist(clusterEvalQ(cl, test.counter()))
[1] 2218 2479 2683
> stopCluster(cl)
> cl = makeCluster(4)
> res = inter.iamb(alarm, cluster = cl)
> unlist(clusterEvalQ(cl, test.counter()))
[1] 1737 1900 1719 2024
> stopCluster(cl)
```

The running times of the algorithms used in Section 2 are reported in Figure 3 for clusters of 1, 2, 3, 4, 6 and 8 slaves; values are averaged over 10 runs for each configuration using generated data sets of size 20,000. Only the results for LINK and MUNIN are shown, as they are the largest reference BNs considered in this paper. It is clear from the figure that the gains in running time follow the law of diminishing returns: adding more slaves produces smaller and smaller improvements. Furthermore, tests are never split uniformly among the slave

---

Figure 2: Parallel implementation of Algorithm 1 in `bnlearn`. 
Figure 3: Normalised running times for learning the skeletons of the MUNIN and LINK reference BNs with the GS, Inter-IAMB, MMPC and SI-HITON-PC algorithms. Raw running times are reported for backtracking and for parallel learning with 1, 2 and 8 slave processes.
Figure 4: Running times for learning the skeletons underlying the lung adenocarcinoma (Beer et al. 2002) and mice (Valdar et al. 2006) data sets with SI-HITON-PC. Raw running times are reported for backtracking and for parallel learning with 1, 2, 3, 4 and 20 slave processes.
processes and therefore slaves that have fewer tests to perform are left waiting for others to complete (see, for instance, the R code snippets above). Even so, the parallel implementations in `bnlearn` scale efficiently up to 8 slaves. In the absence of any overhead we would expect the average normalised running time to be approximately $1/8 = 0.125$; observed values are in the range $[0.157, 0.191]$. Optimised learning is at best competitive with 2 slaves (MMPC, MUNIN), and at worst may actually degrade performance (LINK, SI-HITON-PC).

3.2. Simulations on the Real-World Data

To provide a more realistic benchmarking on large-scale biological data, we applied SI-HITON-PC on the lung adenocarcinoma gene expression data (86 observations, 7131 variables representing expression levels) from Beer et al. (2002); and on the WTCCC heterogeneous mice SNP data (1940 observations, 4053 variables representing allele counts) from Valdar et al. (2006). Both data sets are publicly available and have been preprocessed to remove highly correlated variables ($\text{COR} > 0.95$) and to impute missing values with the `impute` package (Hastie, Tibshirani, Narasimhan, and Chu 2013). The adenocarcinoma data set has a sample size which is extremely small compared to the number of variables, which is common in systems biology. On the other hand, the mice data has a sample size that is typical of large genome-wide association studies. We ran SI-HITON-PC using 1, 2, 3, 4, 6, 8, 10, 12, 14, 16, 18 and 20 slaves, averaging over 5 runs in each case; the other algorithms we considered in Section 2 did not scale well enough to handle either data set. Variables were treated as continuous, and independence was tested using the Student’s $t$ test for correlation with $\alpha = 0.01$.

As we can see in Figure 4, we observe a low overhead even for 20 slave processes, with normalised running times of 0.062 (mice) and 0.076 (adenocarcinoma) which are very close to the theoretical $1/20 = 0.05$. Similar considerations can be made across the whole range of 2 to 20 slaves, with a measured overhead between 0.02 and 0.08. Surprisingly, overhead seems to decrease in absolute terms with the number of slaves, from 0.04 (adenocarcinoma) and 0.08 (mice) for 3 slaves to 0.012 (adenocarcinoma) and 0.026 (mice) for 20 slaves. However, clusters with 2 slaves have a small overhead (0.021 and 0.037) than those with 3 or 4 slaves. We also note that overhead is comparable to that of the reference BNs in Section 2, suggesting that it does not strongly depend on the size of the BN. Again, the running time of the optimised implementation is comparable with that of 2 slaves.

4. Discussion and Conclusions

In this paper we proposed a general framework to create parallel implementations of constraint-based structure learning algorithms for BNs and implemented it in `bnlearn`. Since all these algorithms trace their roots to the IC algorithm from Verma and Pearl (1991), they share a common layout and can be parallelised in the same way. In particular, steps 1, 3 and 5 are embarrassingly parallel and can be trivially split in independent parts to be executed simultaneously. This is important for two reasons. Firstly, it limits the amount of overhead in the parallel computations due to the need of keeping different parts of the algorithms in sync. As we have seen in Section 3, this allows the parallel implementations to scale efficiently in the number of slave processes. Secondly, it implies that the parallel implementation of each algorithm performs the same conditional independence tests as the original. This is in contrast
with backtracking, which is the only widespread way of improving the sequential performance of constraint-based algorithms. The simulations in Section 2 suggest that backtracking can increase the variability of the DAGs learned from the data. At the same time, speed gains are competitive at most with the parallel implementation with 2 slave processes. Since most computers in recent times come with at least 2 cores, it is possible to outperform backtracking even on commodity hardware while retaining the lower variability of the unoptimized algorithms.

References

Aliferis CF, Statnikov A, Tsamardinos I, Mani S, Xenofon XD (2010). “Local Causal and Markov Blanket Induction for Causal Discovery and Feature Selection for Classification Part I: Algorithms and Empirical Evaluation.” Journal of Machine Learning Research, 11, 171–234.

Andreassen S, Jensen FV, Andersen SK, Falck B, Kjærulff U, Woldbye M, Sørensen AR, Rosenfalck A, Jensen F (1989). “MUNIN - an Expert EMG Assistant.” In Computer-Aided Electromyography and Expert Systems. Elsevier.

Beer DG, Kardia SLR, Huang CC, Giordano TJ, Levin AM, Misek DE, Lin L, Chen G, Gharib TG, Thomas DG, Lizyness ML, Kuick R, Hayasaka S, Taylor JMG, Iannettoni MD, Orringer MB, Hanash S (2002). “Gene-expression Profiles Predict Survival of Patients with Lung Adenocarcinoma.” Nature Medicine, 8, 816–824.

Beinlich IA, Suermondt HJ, Chavez RM, Cooper GF (1989). “The ALARM Monitoring System: A Case Study with Two Probabilistic Inference Techniques for Belief Networks.” In J Hunter, J Cookson, J Wyatt (eds.), Proceedings of the 2nd European Conference on Artificial Intelligence in Medicine (AIME), pp. 247–256. Springer-Verlag.

Chang HH, McGeachie M (2011). “Phenotype Prediction by Integrative Network Analysis of SNP and Gene Expression Microarrays.” In Proceedings of the 33rd Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC), pp. 6849–6852. IEEE Press, New York.

Chickering DM (1995). “A Transformational Characterization of Equivalent Bayesian Network Structures.” In Proceedings of the 11th Conference on Uncertainty in Artificial Intelligence (UAI95), pp. 87–98.

Chickering DM (1996). “Learning Bayesian Networks is NP-Complete.” In D Fisher, H Lenz (eds.), Learning from Data: Artificial Intelligence and Statistics V, pp. 121–130. Springer-Verlag.

Chickering DM, Geiger D, Heckerman D (1994). “Learning Bayesian Networks is NP-Hard.” Technical report, Microsoft Research, Redmond, Washington. Available as Technical Report MSR-TR-94-17.

Colombo D, Maathuis MH (2013). “Order-independent Constraint-based Causal Structure Learning.” ArXiv preprint. URL http://arxiv.org/abs/1211.3295v2.
Conati C, Gertner AS, VanLehn K, Druzdzel MJ (1997). “On-line Student Modeling for Coached Problem Solving Using Bayesian Networks.” In A Jameson, C, C Tasso (eds.), In Proceedings of the 6th International Conference on User Modeling, pp. 231–242. Springer-Verlag.

Cooper GF (1990). “The Computational Complexity of Probabilistic Inference Using Bayesian Belief Networks.” Artificial Intelligence, 42(2–3), 393–405.

Cussens J (2011). “Bayesian Network Learning with Cutting Planes.” In FG Cozman, A Pfeffer (eds.), Proceedings of the 27th Conference on Uncertainty in Artificial Intelligence (UAI 2011), pp. 153–160. AUAI Press.

Dagum P, Luby M (1993). “Approximating probabilistic inference in Bayesian belief networks is NP-Hard.” Artificial Intelligence, 60(1), 141–153.

Daly R, Shen Q (2007). “Methods to Accelerate the Learning of Bayesian Network Structures.” In Proceedings of the 2007 UK Workshop on Computational Intelligence. Imperial College, London.

Friedman N (2004). “Inferring Cellular Networks Using Probabilistic Graphical Models.” Science, 303(5659), 799–805.

Geiger D, Heckerman D (1994). “Learning Gaussian Networks.” Technical report, Microsoft Research, Redmond, Washington. Available as Technical Report MSR-TR-94-10.

Hastie T, Tibshirani R, Narasimhan B, Chu G (2013). impute: imputation for microarray data. R package version 1.36.0.

Heckerman D, Geiger D, Chickering DM (1995). “Learning Bayesian Networks: The Combination of Knowledge and Statistical Data.” Machine Learning, 20(3), 197–243. Available as Technical Report MSR-TR-94-09.

Jansen R, Yu H, Greenbaum D, Kluger Y, Krogan NJ, Chung S, Emili A, Snyder M, Greenblatt JF, Gerstein M (2003). “A Bayesian Networks Approach for Predicting Protein-Protein Interactions from Genomic Data.” Science, 302(5644), 449–453.

Jensen CS, Kong A (1999). “Blocking Gibbs Sampling for Linkage Analysis in Large Pedigrees with Many Loops.” The American Journal of Human Genetics, 65(3), 885–901.

Jungnickel D (2008). Graphs, Networks and Algorithms. 3rd edition. Springer-Verlag.

Koller D, Friedman N (2009). Probabilistic Graphical Models: Principles and Techniques. MIT Press.

Lauritzen SL, Wermuth N (1989). “Graphical Models for Associations between Variables, some of which are Qualitative and some Quantitative.” Annals of Statistics, 17(1), 31–57.

Malovini A, Nuzzo A, Ferrazzi F, Puca A, Bellazzi R (2009). “Phenotype Forecasting with SNPs Data Through Gene-Based Bayesian Networks.” BMC Bioinformatics, 10(Suppl 2), S7.
Margaritis D (2003). Learning Bayesian Network Model Structure from Data. Ph.D. thesis, School of Computer Science, Carnegie-Mellon University, Pittsburgh, PA. Available as Technical Report CMU-CS-03-153.

Meek C (1995). “Causal Inference and Causal Explanation with Background Knowledge.” In P Besnard, S Hanks (eds.), Proceedings of the 11th Conference on Uncertainty in Artificial Intelligence (UAI 2011), pp. 403–410. Morgan Kaufmann.

Nagarajan R, Scutari M, Lèbre S (2013). Bayesian Networks in R with Applications in Systems Biology. Use R! series. Springer.

Onisko A (2003). Probabilistic Causal Models in Medicine: Application to Diagnosis of Liver Disorders. Ph.D. thesis, Institute of Biocybernetics and Biomedical Engineering, Polish Academy of Science, Warsaw.

Pearl J (1988). Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann.

R Core Team (2014). R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria. URL http://www.R-project.org/.

Rauber T, Rünger G (2010). Parallel Programming For Multicore and Cluster Systems. Springer-Verlag.

Russell SJ, Norvig P (2009). Artificial Intelligence: A Modern Approach. 3rd edition. Prentice Hall.

Sachs K, Perez O, Pe’er D, Lauffenburger DA, Nolan GP (2005). “Causal Protein-Signaling Networks Derived from Multiparameter Single-Cell Data.” Science, 308(5721), 523–529.

Scutari M (2010). “Learning Bayesian Networks with the bnlearn R Package.” J. Stat. Soft., 35(3), 1–22.

Sebastiani P, Ramoni MF, Nolan V, Baldwin CT, Steinberg M (2005). “Genetic Dissection and Prognostic Modeling of Overt Stroke in Sickle Cell Anemia.” Nature Genetics., 37(4), 435–440.

Spirtes P, Glymour C, Scheines R (2000). Causation, Prediction, and Search. MIT Press.

Tsamardinos I, Aliferis CF, Statnikov A (2003). “Algorithms for Large Scale Markov Blanket Discovery.” In I Russell, SM Haller (eds.), Proceedings of the 16th International Florida Artificial Intelligence Research Society Conference, pp. 376–381. AAAI Press.

Tsamardinos I, Brown LE, Aliferis CF (2006). “The Max-Min Hill-Climbing Bayesian Network Structure Learning Algorithm.” Machine Learning, 65(1), 31–78.

Valdar W, Solberg LC, Gaugnier D, Burnett S, Klierman P, Cookson WO, Taylor MS, Rawlins JN, Mott R, Flint J (2006). “Genome-Wide Genetic Association of Complex Traits in Heterogeneous Stock Mice.” Nature Genetics, 8, 879–887.

Verma TS, Pearl J (1991). “Equivalence and Synthesis of Causal Models.” Uncertainty in Artificial Intelligence, 6, 255–268.
Yaramakala S, Margaritis D (2005). “Speculative Markov Blanket Discovery for Optimal Feature Selection.” In *ICDM ’05: Proceedings of the Fifth IEEE International Conference on Data Mining*, pp. 809–812. IEEE Computer Society.

**Affiliation:**
Marco Scutari
Genetics Institute (UGI)
University College London (UCL)
WC1E 6BT London, United Kingdom
E-mail: m.scutari@ucl.ac.uk
URL: [http://www.bnlearn.com/about](http://www.bnlearn.com/about)