OPTIMIZED ALGORITHM FOR LEARNING BAYESIAN NETWORK SUPER-STRUCTURES

Edwin Villanueva and Carlos Dias Maciel
Department of Electrical Engineering, Sao Carlos School of Engineering, University of Sao Paulo, Sao Paulo, Brazil

Keywords: Bayesian networks, Structure learning, Super-structure.

Abstract: Estimating super-structures (SS) as structural constraints for learning Bayesian networks (BN) is an important step of scaling up these models to high-dimensional problems. However, the literature has shown a lack of algorithms with an appropriate accuracy for such purpose. The recent Hybrid Parents and Children - HPC (De Morais and Aussem, 2010) has shown an interesting accuracy, but its local design and high computational cost discourage its use as SS estimator. We present here the OptHPC, an optimized version of HPC that implements several optimizations to get an efficient global method for learning SS. We demonstrate through several experiments that OptHPC estimates SS with the same accuracy than HPC in about 30% of the statistical tests used by it. Also, OptHPC showed the most favorable balance sensitivity/specificity and computational cost for use as super-structure estimator when compared to several state-of-the-art methods.

1 INTRODUCTION

A Bayesian Network (BN) is a powerful tool for representing complex probabilistic knowledge. It have been broadly applied in a variety of fields (Pourret et al., 2008). The knowledge in a BN is intuitively represented via a directed acyclic graph DAG (model structure), where nodes represent domain variables and edges represent dependencies between them.

The main difficulty in building a BN is the induction of its structure from data. A wealth of literature has been produced for this end with two dominant approaches: the constraint-based (CB) and the score-and-search (SS). In the CB approach, the structure is found via conditional independence (CI) tests. In the SS approach, the network is found by optimizing a function that measures how well the network fits the data. Both approaches have drawbacks. CB methods are inaccurate in dense networks with limited data because CI tests become unreliable in such cases. SS methods are more accurate, but they do not scale up to high-dimensional problems due to a super-exponential growth of the search space (Wang et al., 2007). Hybrid methods have emerged to overcome these limitations (Wong and Leung, 2004; Tsamardinos et al., 2006; Perrier et al., 2008; Kojima et al., 2010). In this approach, an undirected graph (super-structure) is estimated first with a CB approach, which is then used in a subsequent SS phase as structural constraints, i.e., the final structure is searched considering only edges on the super-structure.

Despite the expected gain in scalability and accuracy with the hybrid approach, some concerns has been raised regarding the need of specialized methods to estimate super-structures (Perrier et al., 2008; Kojima et al., 2010). Indeed, most available CB methods were designed for one of two purposes: either they were made to get the exact skeleton (as the PC (Spirtes et al., 2000)) or they were made to get a local network around a variable of interest (as MMPC and GetPC (Tsamardinos et al., 2006; Pena et al., 2007)). The problem is that those methods do not control actively the rate of false-negative errors (FNE) when the assumptions in which they rely are not more valid (limited-data cases). Controlling the FNE rate of the super-structure estimation is key in a hybrid approach, since it will be the FNE rate of the whole learning process (Perrier et al., 2008). The recent Hybrid Parents and Children - HPC algorithm (De Morais and Aussem, 2010) was proposed to lower the FNE rate in a local context (it gets the parents and children of a variable). HPC has shown an interesting FNE reduction in limited-data, while being correct in the sample limit. Unfortunately, such reduction comes at an increased computational cost. Also, a direct use of it as super-structure learner would lead to repetitions in CI tests, increasing more the computational cost, which could undermine the benefits of the hybrid approach.

In this paper we present the Optimized Hybrid Parents and Children (OptHPC), an optimized version of HPC for tasks of super-structure estimation. The optimizations were done in order to get a global method and to lower the computational cost of HPC, while maintaining the precision of it. Among the optimizations are: the use of a cache to store dependence calculations of zero and first order and the use of a global structure to consult/store detected zero-first-order CIs. OptHPC is compared against representative state-of-the-art algorithms, including the HPC, the MMPC, the GetPC, and the Heuristics PC (HeuPC) (Wang et al., 2007). Results from several benchmark datasets show OptHPC as a promising super-structure estimator in the hybrid learning approach.

2 BACKGROUND

A BN (Pearl, 1988) is a model $\langle G, \Theta \rangle$ for representing the joint probability distribution P over a set of random variables $\mathbf{U} = \{X_1, \ldots, X_n\}$. G is a directed acyclic graph - DAG (the model structure) whose nodes have a one-to-one correspondence to the random variables in $\mathbf{U}$ (reason why nodes and variables are used indistinctly) and edges represent conditional dependence relationships among variables. Θ is a set of parameters that define for each node X_i a conditional probability distribution (conditioned on the parents of X_i). All BN satisfies the Markov condition (MC) (Pearl, 1988): every node X_i is conditionally independent on any subset of its non-descendants given its parents $\mathbf{Pa}_i$. A BN is said to be faithful with respect to a distribution P if the MC applied on its DAG G entails all and only the CIs in P. A faithful distribution P is one for which exists a faithful BN $\langle G, \cdot \rangle$; G is called a perfect map of P. In this paper we assume faithful distributions to be learned, so the structure learning problem is the induction of a DAG G from a statistical sample D (following distribution P) that is a perfect map of P (Perrier et al., 2008). We use $X \perp Y | \mathbf{Z}$ to denote that variables X and Y are CI given a variable set $\mathbf{Z}$. The order of a CI is the cardinality of the conditioning set. The set of parents and children of a node X is denoted by $\mathbf{PC}_X$, the set of spouses by $\mathbf{SP}_X$ and the Markov blanket by $\mathbf{MB}_X$.

3 OPTIMIZED HYBRID PARENTS AND CHILDREN - OptHPC

The main disadvantage of HPC as super-structure learner is its local design. For instance, when HPC is run on a variable T, many tests are performed to check CIs of the form $T \perp X | \mathbf{Z}$. When HPC is run latter on X, some CI tests involving T (in the form $X \perp T | \mathbf{Z}$) are performed again. These repetitions could be painful in high-dimensional problems, since CI tests are expensive operations. OptHPC overcomes such limitation by using 2 global structures: i) a cache $\mathcal{C}$ to store/recover computed degrees of dependency of zero and first order; and ii) a global graph $\dot{\mathbb{G}}$ to store and get detected CIs of zero and first order (the absence of a link between nodes X, Y in $\dot{\mathbb{G}}$ indicates $X \perp Y | \mathbf{Z}$, $|\mathbf{Z}| < 2$). Only computations of zero and first order are stored in cache because they are the largest portion of CIs in common problems. Also, if higher orders computations are cached, the complexity and time delays of the cache could be significantly exacerbated. The structure $\dot{\mathbb{G}}$ serves as a fast means to save/consult CIs of zero and first order detected along the execution of the algorithm. We do not represent higher order CIs in $\dot{\mathbb{G}}$ for simplicity and because they are less reliable when data is limited.

Algorithm 1: OptHPC.

```
  input : $D$ (dataset, samples of random vars. $\mathbf{U}$)
  output: $\mathbb{S}$ (super-structure)
1 $\mathbb{S} = (\mathbf{U}, \emptyset)$ // Init. super-structure
2 $\dot{\mathbb{G}} = (\mathbf{U}, \mathbf{U} \times \mathbf{U})$ // Init. 0-1-order graph
3 $\mathcal{C}_{\langle X,Y \rangle} = \emptyset, \forall (X — Y) \in \dot{\mathbb{G}}$ // Init. cache
4 for all $T \in \mathbf{U}$ do
5     $[\mathbf{PC}_T, \dot{\mathbb{G}}, \mathcal{C}] = $ HPC*$(T, D, \dot{\mathbb{G}}, \mathcal{C})$
6     for all $X \in \mathbf{PC}_T$ do $\mathbb{S} = \mathbb{S} + (T — X)$
```

OptHPC has the same subroutines than HPC, but optimized to work with the new structures $\dot{\mathbb{G}}$, $\mathcal{C}$ (we rename them by appending * to the original names, i.e., HPC*, DE-PCS*, DE-SPS* and Inter-IAPC*). The main procedure (Algorithm 1) constructs the super-structure $\mathbb{S}$ by calling successively the modified HPC* on each system variable. $\mathcal{C}$ and $\dot{\mathbb{G}}$ are passed to HPC* in each call. $\dot{\mathbb{G}}$ is initialized with a fully-connected graph (edges are removed as CIs are detected). In contrast, super-structure $\mathbb{S}$ is initialized without edges (they are added with each return of HPC*, line 6). The CI testing is carried out through the function *Dep*. It computes from dataset D the degree of dependence between variables X and Y given set $\mathbf{Z}$, $dep_{X,Y|\mathbf{Z}}$. If such degree is lower than a threshold α (significance level) a 0 is returned indicating $X \perp Y | \mathbf{Z}$. If $|\mathbf{Z}| < 2$ the resulting degree is added to the cache $\mathcal{C}$; specifically to the entry $\mathcal{C}_{\langle X,Y \rangle}$, which is a dictionary that save computed degree of dependencies relative to variables X and Y (the key is the conditioning variable Z and the value is $dep_{X,Y|Z}$). Each required degree of dependence of zero or first order is

Algorithm 2: HPC*.

```
   input  : $T$ (target); $D$ (dataset); $\dot{\mathbb{G}}$ and $\mathcal{C}$
   output: $\mathbf{PC}_T$; updated $\dot{\mathbb{G}}$ and $\mathcal{C}$
1  $[\dot{\mathbb{G}},\mathcal{C}] = $ DE-PCS*$(T,D,\dot{\mathbb{G}},\mathcal{C})$
2  $[\dot{\mathbb{G}},\mathcal{C}] = $ DE-SPS*$(T,D,\dot{\mathbb{G}},\dot{\mathbb{G}})$
3  $\mathbf{PCS}_T = Adj(T,\dot{\mathbb{G}})$;   $\mathbf{SPS}_T = Sps(T,,\dot{\mathbb{G}})$
4  $\mathbf{U}_T = T \cup \mathbf{PCS}_T \cup \mathbf{SPS}_T$
5  $[\mathbf{PC}_T,\dot{\mathbb{G}},\mathcal{C}] = $ Inter-IAPC*$(T,D,\mathbf{U}_T,\dot{\mathbb{G}},\mathcal{C})$
6  for all $X \in \mathbf{PCS}_T \setminus \mathbf{PC}_T$ do
7      $[\mathbf{PC}_X,\dot{\mathbb{G}},\mathcal{C}] = $ Inter-IAPC*$(X,D,\mathbf{U}_T,\dot{\mathbb{G}},\mathcal{C})$
8      if $T \in \mathbf{PC}_X$ then  $\mathbf{PC}_T = \mathbf{PC}_T \cup X$
```

Algorithm 3: DE-PCS*.

```
   input  : $T$ (target); $D$ (dataset); $\dot{\mathbb{G}}$ and $\mathcal{C}$
   output: updated $\dot{\mathbb{G}}$ and $\mathcal{C}$
   // Remove edge $T$—$X$ from $\dot{\mathbb{G}}$ if $T \perp X$
1  for all $X \in Adj(T,\dot{\mathbb{G}})$ do
2      $[dep,\mathcal{C}] = Dep(T,X,\emptyset,D,\mathcal{C})$
3      if $dep = 0$ then $\dot{\mathbb{G}} = \dot{\mathbb{G}} - (T$—$X)$
   // Remove $T$—$X$ from $\dot{\mathbb{G}}$ if $T \perp X|Y$
4  for all $X \in Adj(T,\dot{\mathbb{G}})$ do
5      $Ad_T = Adj(T,\dot{\mathbb{G}})$; $Ad_X = Adj(X,\dot{\mathbb{G}})$
6      if $Ad_T \setminus X = \emptyset$ or $Ad_X \setminus T = \emptyset$ then continue
7      $B_1 = Ad_T \cap Ad_X$; $B_2 = Ad_T \setminus B_1$; $B_3 = Ad_X \setminus B_1$
8      for $i = 1$ to 3 do
9          for all $Y \in B_i$ do
10             $[dep,\mathcal{C}] = Dep(T,X,Y,D,\mathcal{C})$
11             if $dep = 0$ then
12                 $\dot{\mathbb{G}} = \dot{\mathbb{G}} - (T$—$X)$;   break 2 loops
```

first checked in the respective cache entry.

HPC* (Algorithm 2) computes the parents and children (PC) of a target variable by using the same basic steps than HPC. However, differently from it, HPC* works in a global context, placing the results of zero-first-order CI testing in the cache $\mathcal{C}$ and structure $\dot{\mathbb{G}}$ to be available along successive calls to it. Subroutine DE-PCS* (Algorithm 3) is called by HPC* to get a superset of PC (SPC) for a target T. DE-PCS* puts its result in $\dot{\mathbb{G}}$ instead of returning it explicitly (as in DE-PCS). The SPC is recovered in HPC* by looking the target's adjacency in $\dot{\mathbb{G}}$ (with function Adj), which avoids repeated computations. Other optimizations in DE-PCS* are: i) (line 6) prevent the search for d-separators on edges T—X with T or X being a leaf node (node whose unique adjacency is the other edge node), since such edges cannot be separated; and ii) start the search for separators in the common adjacency of the edge nodes, and then in the remaining adjacency (lines 7-12). This in practice speeds up the finding of separators, since they (if any) are more likely to be in the common edge adjacency. Subroutine DE-SPS* (Algorithm 4) is called by HPC* to get

Algorithm 4: DE-SPS*.

```
   input  : $T$ (target); $D$ (dataset); $\dot{\mathbb{G}}$ and $\mathcal{C}$
   output: updated $\dot{\mathbb{G}}$ and $\mathcal{C}$
1  for all $X \in Adj(T,\dot{\mathbb{G}})$ do
2      $Ad_X = Adj(X,\dot{\mathbb{G}})$;   $Ad_T = Adj(T,\dot{\mathbb{G}})$
3      $Sp_T^X = Sps(T,X,\dot{\mathbb{G}})$
4      for all $Y \in Ad_X \setminus \{T \cup Ad_T \cup Sp_T^X\}$ do
5          $sep = Sep(T,X,\mathcal{C})$
6          if $sep = X$ then  continue
7          $[dep,\mathcal{C}] = Dep(T,Y,X \cup Sep,D,\mathcal{C})$
8          if $dep \neq 0$ then // make $T \rightarrow X \leftarrow Y$
9              $\dot{\mathbb{G}} = \dot{\mathbb{G}} - \{(X \rightarrow T),(X \rightarrow Y)\}$
10     for all $Y \in Sps(T,X,\dot{\mathbb{G}})$ do
11         for all $Z \in Sps(T,X,\dot{\mathbb{G}}) \setminus Y$ do
12             $[dep,\mathcal{C}] = Dep(T,Y,X \cup Z,D,\mathcal{C})$
13             if $dep = 0$ then // restore $X \leftrightarrow Y$
14                 $\dot{\mathbb{G}} = \dot{\mathbb{G}} + (X \rightarrow Y)$;   break
```

Algorithm 5: Inter-IAPC*.

```
   input  : $T$ (target); $D$ (data); $\mathbf{U}_T$ (variables); $\dot{\mathbb{G}}$; $\mathcal{C}$
   output: $\mathbf{PC}_T$; updated $\dot{\mathbb{G}}$ and $\mathcal{C}$
1  $\mathbf{MB}_T = \emptyset$
2  repeat
       // Add true positives to $\mathbf{MB}_T$
3      for all $X \in \{\mathbf{U}_T \setminus \mathbf{MB}_T \setminus T\}$ do
4          $[dep_X,\mathcal{C}] = Dep(T,X,\mathbf{MB}_T,D,\mathcal{C})$
5          if $dep_X = 0$ and $|\mathbf{MB}_T| < 2$ then
6              $\dot{\mathbb{G}} = \dot{\mathbb{G}} - (T$—$X)$
7      if $max_X(dep_X) = 0$ then break
8      $\mathbf{MB}_T = \mathbf{MB}_T \cup argmax_X(dep_X)$
       // Remove false positives from $\mathbf{MB}_T$
9      for all $X \in \mathbf{MB}_T$ do
10         $[dep,\mathcal{C}] = Dep(T,X,\mathbf{MB}_T \setminus X,D,\mathcal{C})$
11         if $dep = 0$ then
12             $\mathbf{MB}_T = \mathbf{MB}_T \setminus X$
13             if $|\mathbf{MB}_T| < 2$ then $\dot{\mathbb{G}} = \dot{\mathbb{G}} - (T$—$X)$
14 until $\mathbf{MB}_T$ has not changed
15 if $|\mathbf{MB}_T| < 2$ then return
   // Remove spouses of $T$ from $\mathbf{MB}_T$
16 $\mathbf{PC}_T = \mathbf{MB}_T$
17 for all $X \in \mathbf{MB}_T$ do
18     $[fsep,sep,\mathcal{C}] = FindSep(T,X,\mathbf{PC}_T \setminus X,D,\mathcal{C})$
19     if $fsep = true$ then
20         $\mathbf{PC}_T = \mathbf{PC}_T \setminus X$
21         if $|sep| < 2$ then $\dot{\mathbb{G}} = \dot{\mathbb{G}} - (T$—$X)$
```

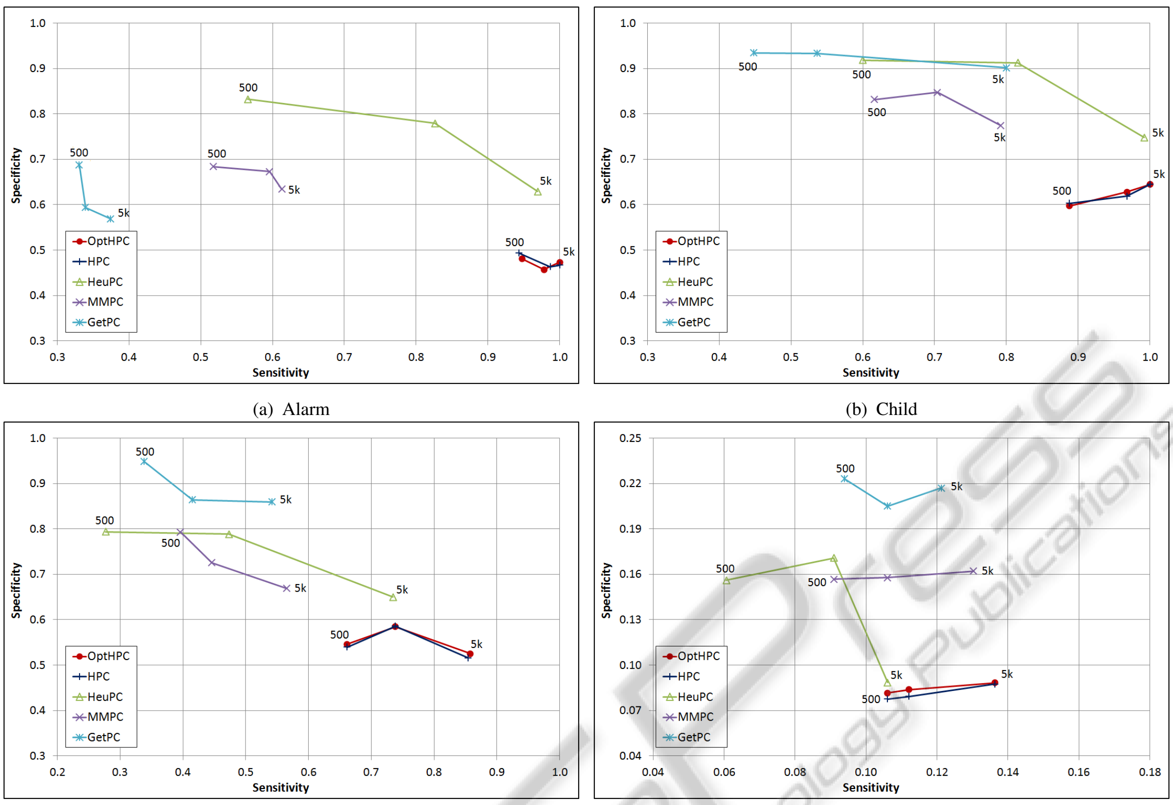
a superset of spouses (SPS) for a target T. As in DE-PCS*, $\dot{\mathbb{G}}$ is used as the working object. The SPS is recovered from $\dot{\mathbb{G}}$ in HPC* with function Sps (when passed a second argument X to it, only the target's spouses with child X are returned, as in lines 3,10-11 in DE-SPS*). Whenever a possible v-structure $T \rightarrow X \leftarrow Y$ is detected (i.e. when Y becomes dependent on T given the current separator and X) it is reflected in $\dot{\mathbb{G}}$ by orienting the respective edges (lines 7-9). In a second phase (lines 10-14) all created v-structures $T \rightarrow X \leftarrow Y$ are reviewed: if any variable Y is found as ancestor or descendant of another target's spouse, it is considerate as non-spouse and the bi-directionality of edge $Y \rightarrow X$ is restored (line 14). Inter-IAPC* (Algorithm 5) is a procedure called by HPC* to get an initial candidate PC set for a target variable T. Inter-IAPC* works only on the set $\mathbf{U}_T$ formed by parents, children and spouses of T returned by DE-PCS* and DE-SPS*, which save computations. $\dot{\mathbb{G}}$ and $\mathcal{C}$ are also used/updated whenever a zero-first-order CI test is required. However, as Inter-IAPC* uses higher order CI testing, the final result is worked on the internal set $\mathbf{PC}_T$: first, the Markov blanket of the target T is obtained (lines 2-14) and then the target's spouses are identified and removed from it (lines 15-21). Function $FindSep$ is used to assist that identification (line 18). It looks for a subset $Z \subseteq \mathbf{Z}$ that makes T and X CI. If found, a flag $fsep$ is returned as "true" along with the found Z (in sep), otherwise $fsep$ is returned as "false". To save computations, we restrict $\mathbf{Z}$ to the current shrinking PC set ($\mathbf{PC}_T \setminus X\}$) instead of the whole $\mathbf{MB}_T$ (as in Inter-IAPC). This is justified because spouse separators always can be found in the true PC set (which is in the current $\mathbf{PC}_T$). Another optimization is in line 7, where the search of the $\mathbf{MB}_T$ is ended as soon as no new candidates exist to enter to it.

4 EXPERIMENTAL EVALUATION

The accuracy and computational cost of OptHPC is compared against some representative state-of-the-art algorithms for skeleton recovery, including the HPC, the MMPC (with all optimizations of the original paper), the Heuristics PC (its original name is algorithmHPC, but in this paper we call it as HeuPC to avoid confusion with HPC) and the GetPC. Accuracy is assessed by the sensitivity ($Sn = TP/(TP+FN)$) and specificity ($Sp = TP/(TP+FP)$) indices (Pena et al., 2007), where TP is the number of edges correctly estimated, FN is the number of missing edges (false negatives), and FP is the number of extra edges. Computational cost is assessed by the number of statistical calls (NSC) used to get the answer. NSC is a fair estimate of the computational efficiency of CB methods, since they spend most of the time doing statistical tests (Tsamardinos et al., 2006). NSC is also independent from the computing platform. Four known benchmark BNs were used for the accuracy evaluation: Alarm, Child, Insurance and Haildfinder. They were chosen to have increasing complexity cases (Alarm has 37 nodes, 46 edges, domain range DR=[2-4] and maximum indegree MI=6; Child has 20 nodes, 25 edges, DR=[2-6] and MI=8; Insurance has 27 nodes, 52 edges, DR=[2-5] and MI=9; Hailfinder has 56 nodes, 66 edges, DR=[2-11] and MI=17). All datasets were taken from the repository at www.dsl-lab.org/supplements/mmhc_paper/mmhc_index.html released by the authors of the MMPC. We choose for each network 15 datasets: 5 with 500 instances, 5 with 1000 instances and 5 with 5000 instances. For the NSC evaluation, we use additionally datasets sampled from tiled versions of the above networks (in the same repository). Tiling is a procedure to construct networks with larger dimensionality by joining several copies of an original network (Tsamardinos et al., 2006). We choose tiling of 3, 5 and 10. All indices are averaged for the same network and sample size. Due to the need of a common platform to make fair comparisons, only author's own implementations of the studied algorithms were used. Every effort was taken to match our implementation to the originals. All algorithms perform the CI testing through function Dep in order to standardize the NSC counting. G^2 is used as the CI test statistics ($\alpha = 0.05$) (Tsamardinos et al., 2006). It is not performed if there are less than 5 samples on average per cell; in that case independence is assumed (Tsamardinos et al., 2006).

Figure 1 shows the sensitivity and specificity values obtained in the 4 networks. Each curve represents an algorithm, where the points correspond to the different sample sizes. It is observed that OptHPC and HPC have very close values in all cases, which confirm that the optimizations in OptHPC do not alter the accuracy of HPC. All algorithms increment the sensitivity with the sample size, as expected, but OptHPC and HPC have values markedly higher than the other in all networks and sample size (more noticeably in small sample size). They, in fact, reach the maximum sensitivity in the simplest networks (Alarm and Child) at sample size 5000. Contrasting with the sensitivity, OptHPC and HPC have the lowest specificity in all networks. This means that more false-positive edges are included in the final structure. However, from the viewpoint of BN hybrid learning, such false-positive increment is of lesser importance than having a good sensitivity. This because the sensitivity of the whole learning process is upper-bounded by the sensitivity of the super-structure estimation phase (since no false negatives can be corrected there), while the specificity can be improved in the SS phase (Wang et al., 2007).

Figure 2 shows the obtained NSC values. Each



(a) Alarm (b) Child

(c) Insurance (d) Hailfinder

Figure 1: Results for sensitivity and specificity indices. The points in the curves (algorithms) correspond to the average values of sensitivity and specificity for the different sample sizes (results for datasets of 500 and 5000 instances are indicated).

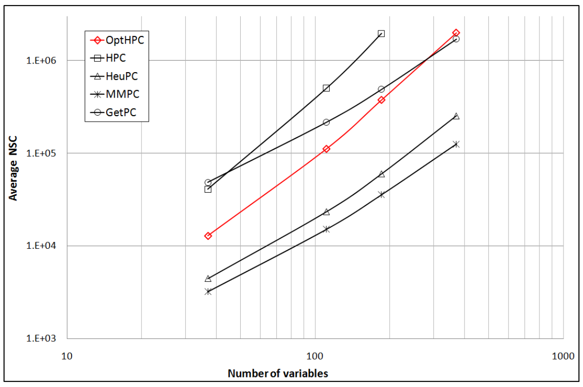
subfigure present results for an original network and its corresponding tiled versions. The points in the curves (algorithms) were plotted in the corresponding dimensionality of the (original or tiled) network and the obtained NSC value (averaged over all sample sizes in the corresponding network). Logarithmic scales are used due to large differences in the results. Values for HPC in networks with tiling 10 are not shown because it took longer time than our imposed tolerance (30 hours single CPU). It can be observed that HPC and GetPC have the highest NSC values among all algorithms, although GetPC has a better scalability to higher dimensions. OptHPC presents a pronounced reduction of the NSC values in all networks with respect to HPC, representing in average about 30% of the NSC used by it. HeuPC and MMPC have the lowest NSC values, but they have a discouraging sensitivity/specificity balance for applications in the hybrid learning approach.

5 CONCLUSIONS

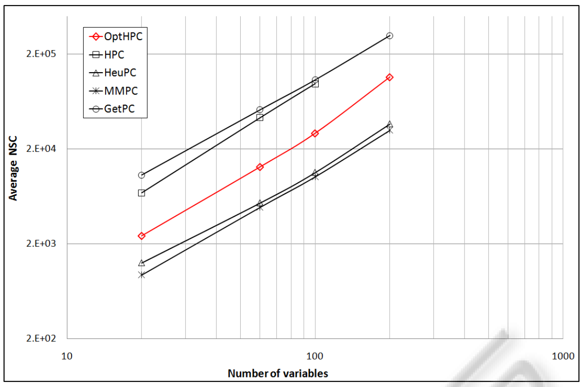
In this paper we presented an optimized version of the recent HPC algorithm, an algorithm for learning the parents and children of a target variable that showed an attractive accuracy for the BN hybrid learning approach. The new algorithm, called OptHPC, implements several optimizations to get an efficient global method for such approach. Results in benchmark datasets showed that OptHPC is effective in reducing the computational cost of HPC, needing on average 30% of the statistical tests used by it without loss of accuracy. Compared to some representative skeleton-recovery algorithms, OptHPC showed the most suitable balance sensitivity/specificity and computational cost for use as super-structure estimator in the hybrid learning approach. We are currently testing OptHPC coupled in a whole hybrid system to asses at what extent the total learning time is reduced.

ACKNOWLEDGEMENTS

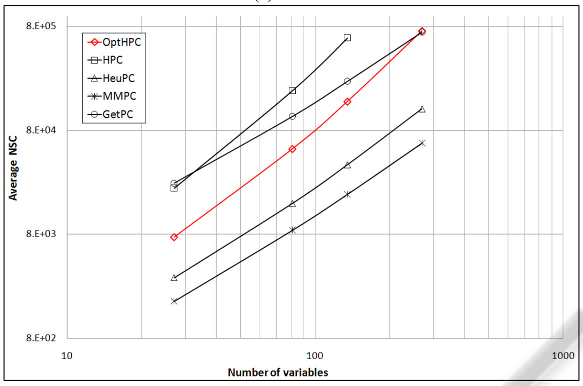
We would like to acknowledge support for this project from the brazilian government agency CAPES.



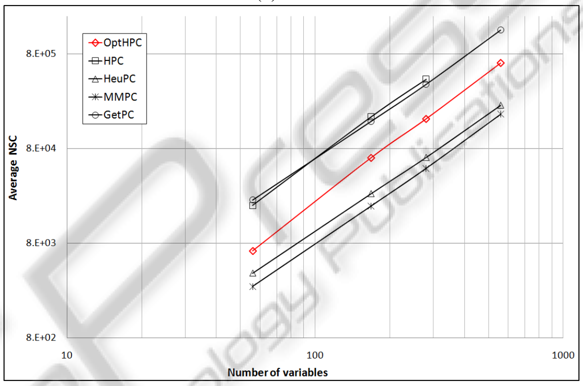
(a) Alarm



(b) Child



(c) Insurance



(d) Hailfinder

Figure 2: Results for the NSC index. Each subfigure shows results from datasets corresponding to the same network (including its tiled versions). The points in the curves (algorithms) were plotted in the corresponding dimensionality of the network and the NSC value (averaged over all sample sizes in that network). Logarithmic scales are used in both axes.

REFERENCES

De Morais, S. R. and Aussem, A. (2010). An efficient and scalable algorithm for local bayesian network structure discovery. In *European conference on Machine learning and knowledge discovery in databases: Part III, ECML PKDD'10*, pages 164–179. Springer-Verlag.

Kojima, K., Perrier, E., Imoto, S., and Miyano, S. (2010). Optimal search on clustered structural constraint for learning bayesian network structure. *J. Mach. Learn. Res.*, 11:285–310.

Pearl, J. (1988). *Probabilistic reasoning in intelligent systems*. Morgan Kaufmann.

Pena, J. M., Nilsson, R., Bjorkegren, J., and Tegner, J. (2007). Towards scalable and data efficient learning of Markov boundaries. *Int. J. Approx. Reasoning*, 45(2):211–232.

Perrier, E., Imoto, S., and Miyano, S. (2008). Finding Optimal Bayesian Network Given a Super-Structure. *J. Mach. Learn. Res.*, 9:2251–2286.

Pourret, O., Nam, P., Naïm, P., Marcot, B., and Na?m, P. (2008). *Bayesian Networks: A Practical Guide to Applications*. Statistics in Practice. John Wiley & Sons.

Spirtes, P., Glymour, C., and Scheines, R. (2000). *Causation, prediction, and search*. The MIT Press, Cambridge, second edition.

Tsamardinos, I., Brown, L. E., and Aliferis, C. F. (2006). The max-min hill-climbing Bayesian network structure learning algorithm. *Mach. Learn.*, 65(1):31–78.

Wang, M., Chen, Z., and Cloutier, S. (2007). A hybrid Bayesian network learning method for constructing gene networks. *Comput. Biol. Chem.*, 31(5-6):361–372.

Wong, M. and Leung, K. (2004). An efficient data mining method for learning Bayesian networks using an evolutionary algorithm-based hybrid approach. *IEEE Trans. Evol. Comput.*, 8(4):378–404.