HAL
open science

Sub-quadratic Markov tree mixture learning based on randomizations of the Chow-Liu algorithm

Sourour Ammar, Philippe Leray, François Schnitzler, Louis Wehenkel

To cite this version:

Sourour Ammar, Philippe Leray, François Schnitzler, Louis Wehenkel. Sub-quadratic Markov tree mixture learning based on randomizations of the Chow-Liu algorithm. PGM 2010, Sep 2010, Helsinki, Finland. pp.17-25. hal-00568028

HAL Id: hal-00568028

https://hal.science/hal-00568028

Submitted on 23 Feb 2011

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Sub-quadratic Markov tree mixture learning based on randomizations of the Chow-Liu algorithm

Sourour Ammar and Philippe Leray
Knowledge and Decision Team
Laboratoire d'Informatique de Nantes Atlantique (LINA) UMR 6241
Ecole Polytechnique de l'Université de Nantes, France
sourour.ammar@univ-nantes.fr, philippe.leray@univ-nantes.fr

François Schnitzler and Louis Wehenkel
Department of EECS & GIGA-Research,
Grande Traverse, 10 - B-4000 Liège - Belgium
fschnitzler@ulg.ac.be, L.Wehenkel@ulg.ac.be

Abstract

The present work analyzes different randomized methods to learn Markov tree mixtures for density estimation in very high-dimensional discrete spaces (very large number n of discrete variables) when the sample size (N) is very small compared to n. Several sub-quadratic relaxations of the Chow-Liu algorithm are proposed, weakening its search procedure. We first study naïve randomizations and then gradually increase the deterministic behavior of the algorithms by trying to focus on the most interesting edges, either by retaining the best edges between models, or by inferring promising relationships between variables. We compare these methods to totally random tree generation and randomization based on bootstrap-resampling (bagging), of respectively linear and quadratic complexity. Our results show that randomization becomes increasingly more interesting for smaller N/n ratios, and that methods based on simultaneously discovering and exploiting the problem structure are promising in this context.

1 Introduction

Directed probabilistic graphical models encode a joint distribution over a set of variables by a product of conditional probability distributions, one for each variable conditionally to its parents in the directed graph. These models may be learned from data and used to perform probabilistic inferences over the encoded distribution (Pearl, 1986). However, exact inference and learning with such models are both NP-hard, unless the skeleton of the graph is constrained (Cooper, 1990). Existing learning algorithms are not scalable to high dimensional spaces because of their excessive computational complexity (Auvray and Wehenkel, 2002).

Markov Trees are an interesting subclass of directed graphical models, whose skeletons are acyclic and for which each node of the graph has (at most) one parent. With Markov trees, the computational complexity of probabilistic inference and parameter learning are linear in the number of variables (Pearl, 1986). Further, Markov tree structures may be learned efficiently by the Chow-Liu algorithm (Section 3.1), quadratic in the number of variables.

While Markov trees impose strong modeling restrictions, mixtures of Markov trees can represent a much wider (actually unlimited) class of probability densities than single Markov trees while retaining their interesting computational properties in terms of inference (Meila and Jordan, 2000), making these models attractive for scaling graphical models to high-dimensional problems. As a matter of fact, these simple graphical models were used for these reasons,

in order to build optimized mixtures of models for probability density estimation (Meila and Jordan, 2000) by using an expectation-minimization algorithm.

In supervised learning, a generic framework which has led to many fruitful innovations is called "Perturb and Combine". Its main idea is to on the one hand perturb in different ways the optimization algorithm used to derive a predictor from a dataset and on the other hand to combine in some appropriate fashion an ensemble of predictors obtained by multiple iterations of the so perturbed search algorithm. This approach may in particular lead to a strong reduction in variance (Breiman, 1996). It was first explored for probability density estimation in (Ammar et al., 2008) by comparing various kinds of large ensembles of simple graphical models in the form of Markov Trees.

The above mentioned algorithms for learning mixtures of Markov trees use the Chow-Liu algorithm (Chow and Liu, 1968). However, since this algorithm is quadratic in the number of variables, these methods do not scale well to very high-dimensional problems, with thousands or even millions of variables. Thus, (Schnitzler et al., 2010; Ammar et al., 2010) tried to investigate mixtures of models learned by using various randomized versions of the Chow-Liu algorithm, with the aim of reducing the computational complexity below the quadratic level, and simultaneously improving accuracy in small (i.e. realistic) sample size conditions by variance reduction. The aim of the present paper is to analyse these methods and compare their results in the same framework.

The rest of the paper is organized as follows. We describe tree models and models of mixtures of trees more formally in section 2, and state in section 3 the different tree mixture learning algorithms that we want to analyse. We explain and discuss our empirical evaluation of these algorithms in section 4, before concluding.

2 Mixtures of Markov trees

Let $X = \{X_1, \ldots, X_n\}$ be a finite set of discrete random variables, and $D = (x^1, \cdots, x^N)$ be a sample (we will use the term "dataset" to denote it) of joint observations $x^i = \{x_1^i, \cdots, x_n^i\}$ independently drawn from some data-generating density $\mathbb{P}_G(X_1, \ldots, X_n)$.

A mixture distribution $\mathbb{P}_{\hat{\mathcal{T}}}(X_1, \ldots, X_n)$ induced by a multiset $\hat{\mathcal{T}} = \{T_1, \ldots, T_m\}$ of m Markov trees is defined as a convex combination of elementary Markov tree densities, i.e.

$$\mathbb{P}_{\hat{\mathcal{T}}}(X) = \sum_{i=1}^{m} \mu_i \mathbb{P}_{T_i}(X),$$

where $\mu_i \in [0, 1]$, $\sum_{i=1}^{m} \mu_i = 1$, and $\mathbb{P}_{T_i}(X)$ is the probability density over X encoded by the graphical model composed of the Markov tree structure S_i and its parameter set $\tilde{\theta}_i$:

$$\mathbb{P}_{T_i}(X) = \mathbb{P}_{S_i, \tilde{\theta}_i}(X) = \prod_{p=1}^{n} P_{\tilde{\theta}_i}(X_p | Pa_{S_i}(X_p)),$$

where $Pa_{S_i}(X_p)$ is the parent variable of X_p in the tree structure S_i.

Several versions of Markov tree mixtures were studied in (Ammar et al., 2009; Ammar et al., 2008) as an alternative to classical methods of density estimation in the context of high-dimensional spaces and small datasets: mixtures of tree structures generated in a totally randomized fashion with linear complexity in the number of variables and ensembles of optimal trees derived from bootstrap replicas of the dataset by the Chow and Liu algorithm (Chow and Liu, 1968) (i.e. bagging of Markov trees).

Other studies tried to relax the Chow-Liu algorithm to reduce its computational complexity while maintaining its accuracy (Schnitzler et al., 2010; Ammar et al., 2010). In the present work we analyze these methods in terms of computational complexity, accuracy, and running time, within a same framework.

3 Panel of learning algorithms

Algorithm 1 describes our general methodology to learn a mixture of m Markov trees from a dataset D.

It may be declined by using different variants of the three subroutines it uses, namely *BuildMarkovTreeStructure*, *LearnPars*, and *Comp-*

Weights. In this paper we focus on the effect of varying only the first one, used to infer the structures S_i of the mixture terms. Next, we describe the different versions of *BuildMarkovTreeStructure* that we have considered in our study. We start by describing the original Chow-Liu method.

Algorithm 1 (Learning a Markov tree mixture).

1. *Repeat for* $i = 1, \cdots, m$:
   (a) $S_i = BuildMarkovTreeStructure(D)$
   (b) $\tilde{\theta}_i = LearnPars(S_i, D)$
2. $(\mu)_{i=1}^m = CompWeights((S_i, \tilde{\theta}_i)_{i=1}^m, D)$
3. *Return* $\left(\mu_i, S_i, \tilde{\theta}_i\right)_{i=1}^m$

3.1 Chow-Liu algorithm

This algorithm learns a Markov tree structure maximizing the likelihood of the training set (Chow and Liu, 1968). Its principle is described by Algorithm 2. It can be decomposed in two steps : Step 1. computes from the dataset the maximum likelihood estimates of the mutual informations between each pair of variables to fill an $n \times n$ symmetrical matrix (MI); Step 2. searches for a maximum weight spanning tree (MWST) in this matrix (e.g. Kruskal's algorithm (Cormen et al., 2001), used here).

Algorithm 2 (Chow-Liu algorithm).

1. $MI = [0]_{n\times n}$; *Repeat for* $k = 1, \cdots, n$:
   Repeat for $j = k+1, \cdots, n$:
   i. $MI[k, j] = CompMI(X_k, X_j, D)$;
   ii. $MI[j, k] = MI[k, j]$.
2. $S = CompKruskal(MI)$; *Return* S.

The first step requires $\mathcal{O}(n^2 N)$ computations, while the second has a complexity of $E \log(E)$ with E the number of considered edges. In the Chow-Liu algorithm $E = n(n-1)/2$, so this second complexity becomes $\mathcal{O}(n^2 \log(n^2))$.

3.2 Randomized edge sampling

To reduce the complexity of the Chow-Liu algorithm, we propose to apply the Perturb and Combine principle by learning each model from an incomplete matrix MI.

The random edge sampling algorithm performs this by randomly selecting a subset of a priori fixed size K of different pairs of variables according to a uniform distribution. These terms are used to partially fill the matrix MI used as input to the MWST algorithm. Algorithm 3 describes this procedure.

Algorithm 3 (randomized edge sampling).

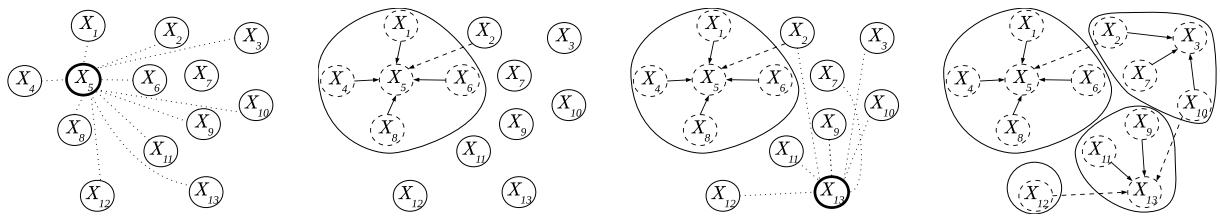
1. $MI = [0]_{n\times n}$; *Repeat for* $k = 1, \cdots, K$:
   (a) *Draw new random pair* $(i_1, i_2) \in \{1, \ldots, n\}^2$;
   (b) $MI[i_1, i_2] = CompMI(X_{i_1}, X_{i_2}, D)$;
   (c) $MI[i_2, i_1] = MI[i_1, i_2]$.
2. $S = CompKruskal(MI)$; *Return* S.

The complexity of Algorithm 3 is loglinear in the number K of edges drawn. Notice that the tree structure that it infers may be disconnected, and that its dependence on the dataset is increasing with the value of K. We will report in this paper simulations and results for two values of the parameter : $K = n\ log(n)$ considered in (Ammar et al., 2010), and $K = 0.33n(n-1)/2$. The first value allows a total complexity of $n \log(n) \log(n \log(n))$, which is sub-quadratic and very close to the quasi-linear. The second corresponds approximately to the edges sampled by the method described in the next section.

3.3 Randomized vertex clustering

Another idea to weaken the Chow and Liu procedure was proposed by (Schnitzler et al., 2010). This method is a less naïve approach to sampling the matrix MI, which targets potentially interesting (i.e. of large weight) edges. Algorithm 4 details this two-step process, that first builds a local structure of the problem, and then focuses on pairs of variables located close to each other in that structure.

The first step consists in an approximate online clustering of the variables based on their mutual information, inspired by leader clustering (a cluster C_p is represented by its leader L_p). As illustrated in figure 1, a sequence C of clusters is created until all variables belong to one. The construction of a cluster C_p is based on two thresholds on mutual information: one cluster-threshold (MI_C) and one neighborhood-threshold (MI_N). The cluster is built by comparing the mutual information of each remaining unclustered variable to the new leader (L_p,



(a) First a leader (here X_5) is chosen at random and compared to all 12 other variables.

(b) Next, the 1st cluster is built. Here it is made of 5 members and has one neighbor.

(c) The 2nd leader (X_{13}, the farthest from X_5) is compared only to 7 variables.

(d) Final result, after 4 iterations. All edges considered are kept for the MWST inference.

Figure 1: Illustration of the vertex clustering algorithm.

chosen first at random, then among unclustered variables by minimizing $\sum_{q<p} MI(L_p, L_q))$. An unclustered variable X is identified as:

1. member of C_p, if $MI(X, L_p) > MI_C$,
2. neighbor of C_p, if $MI_C \geqslant MI(X, L_p) > MI_N$,
3. not related to C_p, otherwise.

Setting those thresholds can be seen as excluding potentially independent variables. This exclusion rate can be controlled, since the maximum likelihood estimate of the mutual information for two independent variables asymptotically follows a χ^2 law. In this work, its percentile 0.5 (respectively 5) was used for MI_C (MI_N).

Algorithm 4 (Vertex Clustering).

1. $\mathcal{V} = \mathcal{X}$; $C = \emptyset$; $MI = [0]_{n\times n}$; *Repeat until* $\mathcal{V} = \emptyset$:
   (a) $L = GetNewLeader(\mathcal{V}, C)$;
   (b) $C, MI \mathrel{+}= MakeCluster(L, \mathcal{V}, MI_N, MI_C)$.
2. $nbClusters = size(C)$;
   Repeat for $p = 1, \cdots, nbClusters$:
   (a) *Repeat for* $i_1, i_2 : X_{i_1}, X_{i_2} \in C_p$:
      i. $MI[i_1, i_2] = CompMI(X_{i_1}, X_{i_2}, D)$;
      ii. $MI[i_2, i_1] = MI[i_1, i_2]$.
   (b) *Repeat* $\forall q < p : C_q \in Neibhors(C_p)$:
      Repeat for $i_1, i_2 : X_{i_1} \in C_p, X_{i_2} \in C_q$:
      A. $MI[i_1, i_2] = CompMI(X_{i_1}, X_{i_2}, D)$;
      B. $MI[i_2, i_1] = MI[i_1, i_2]$.
3. $S = CompKruskal(MI)$; *Return* S.

In the second step of the algorithm, the mutual information of all potentially interesting pairs (X_i, X_j) are computed and used as edge-weights for the MWST algorithm. Interesting pairs (a) are in the same cluster or (b) span two neighboring clusters, i.e one variable of one cluster is a neighbor of the other cluster. In addition, all edges evaluated during the clustering process are used as candidate edges.

The complexity of this algorithm is between linear and quadratic in the number of variables, depending on the numerical values of MI_C and MI_N and the problem structure.

3.4 Inertial search heuristic

This algorithm (Ammar et al., 2010) for computing a sequence of sub-optimal MWST was designed to improve the base method from section 3.2. In this work we also apply it to the vertex clustering algorithm (section 3.3).

The inertial method takes advantage of the Markov tree structure S_{i-1} built in the previous iteration to partially fill the new MI_i matrix. The weights (recomputed in case bootstrap copies of the dataset are used) of the edges of the Markov tree built at the previous iteration $i-1$ are first written in the MI_i matrix of the current iteration i, and a new set of edges generated by the base method (either at random or by vertex clustering) is inserted afterwards. This is described by Algorithm 5.

The complexity of this method is similar to the base method.

Algorithm 5 (Inertial research procedure).

1. $MI_i = [0]_{n\times n}$;
   Repeat for $k = 1, \cdots, nbEdges(S_{i-1})$:
   (a) $(i_1, i_2) = GetIndices(GetEdge(S_{i-1}, k))$;
   (b) $MI_i[i_1, i_2] = CompMI(X_{i_1}, X_{i_2}, D)$;
   (c) $MI_i[i_2, i_1] = MI_i[i_1, i_2]$.
2. *Repeat for* $k = 1, \cdots, nbEdges(BaseMethod)$:
   (a) $(i_1, i_2) =$ *indices of edge* k *from BaseMethod;*
   (b) $MI_i[i_1, i_2] = CompMI(X_{i_1}, X_{i_2}, D)$;
   (c) $MI_i[i_2, i_1] = MI_i[i_1, i_2]$.
3. $S_i = CompKruskal(MI_i)$; *Return* S_i.

3.5 Other variants

Two other variants (baselines) were also considered, namely random trees and bagging.

The first one draws a tree structure totally at random (i.e. independently from the dataset) through the use of Prüfer lists. Complexity is linear in n (Ammar et al., 2008).

Bagging can also be used to increase randomization in a given method, by supplying a bootstrap replica of the original dataset to any tree-structure learning algorithm. This may actually be quite productive in order to randomize tree structures (see the results below). However, as far as accuracy is concerned, it turns out to be preferable to use the full dataset for parameter estimation of each Markov tree generated by this method (Schnitzler et al., 2010).

4 Empirical simulations

We apply the algorithm variants described in Section 3 to synthetic problems to assess their performance. We carried out repetitive experiments for different data-generating (or target) densities as described in Section 4.1; our results are reported in Section 4.2.

4.1 Experimental protocol

We present here results obtained on 10 different target distributions over $n = 1000$ binary variables, and we report them for mixtures and datasets of various sizes.

Target density generation Target densities are synthetic distribution factorizing according to a general directed acyclic graph structure. These models (structure and parameters) are generated by the algorithm described in (Ammar et al., 2008).

Datasets We focus our analysis on rather small datasets ($N = 100, 250, 1000$) with respect to the number $n = 1000$ of variables. This replicates the usual situation in high-dimensional problems, which are the motivation of this work. For each considered sample size and for each target distribution, we generate 5 different datasets.

Mixture learning For a given dataset, and for a given tree structure learning algorithm, we apply the mixture learning algorithm (Algo 1) by generating ensemble models of growing sizes ($m = 1, m = 10, \ldots, m = 150$) in order to appraise the effect of the ensemble size on the quality of the resulting model.

In all our simulations, the parameters of the Markov tree models are learned from the dataset by maximum a posteriori estimation using uniform Dirichlet priors.

In our empirical tests we have always weighted the individual terms uniformly (i.e. $\mu = 1/m$ in Algorithm 1).

Accuracy evaluation We assess the quality of each generated mixture by the Kullback-Leibler divergence (Kullback and Leibler, 1951), an asymmetric measure of similarity of a given distribution $\mathbb{P}_{\hat{\mathcal{T}}}$ to a target distribution $\mathbb{P}_G$, defined by

$$D_{KL}(\mathbb{P}_G \,||\, \mathbb{P}_{\hat{\mathcal{T}}}) = \sum_{X \in \mathcal{X}} \mathbb{P}_G(X) \log_2 \left(\frac{\mathbb{P}_G(X)}{\mathbb{P}_{\hat{\mathcal{T}}}(X)} \right).$$

Since screening all 2^{1000} configurations of $\mathcal{X}$ is not possible, we estimate this quantity by Monte Carlo using a random sample of configurations generated according to $\mathbb{P}_G$:

$$\hat{D}_{KL}(\mathbb{P}_G \,||\, \mathbb{P}_{\hat{\mathcal{T}}}) = \sum_{X \sim \mathbb{P}_G} \log_2 \left(\frac{\mathbb{P}_G(X)}{\mathbb{P}_{\hat{\mathcal{T}}}(X)} \right).$$

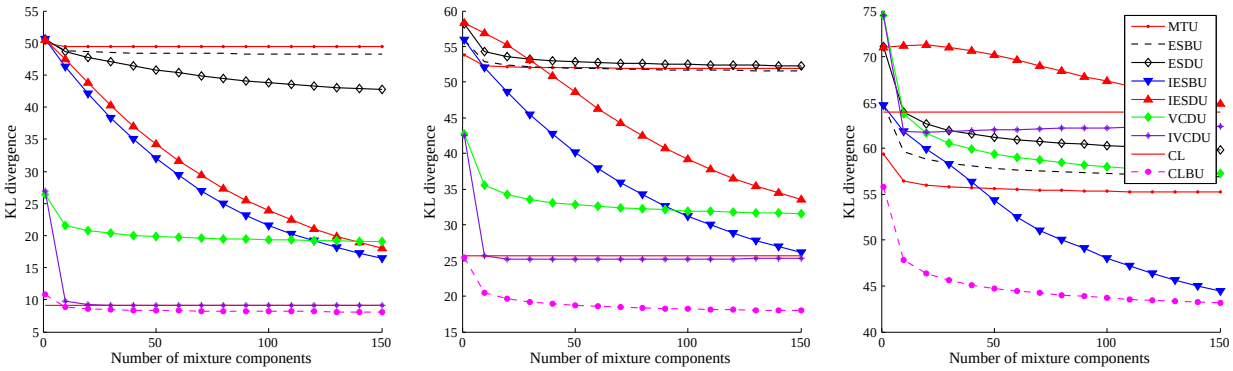
In this work, we generated for each data-generating distribution and each learning algorithm a fixed set of 50000 samples, which is then used for the Monte Carlo estimation of D_{KL} of the models produced by the algorithm applied to the datasets issued from this distribution and with a growing number of mixture terms m.

4.2 Results and discussion

Table 1 describes the algorithm variants that we have evaluated, recalls their computational complexities, and also gives indications of their relative computing times in our implementation. The performances of these algorithms in terms of accuracy (D_{KL} estimates) are reported in Figures 2 and 3. As reference method we use the Chow and Liu single tree method (denoted by CL in the table and figures).

Table 1: In the names of the algorithms we study, D means no alteration to the dataset and B the use of bootstrap replica. m, n, K stand for the number of terms in the mixture, of variables and of sampled pairs of variables. U emphasizes the fact that we are using uniform weights in the mixture. (CPU times are given for $n = 1000$ variables)

Name	Tree generation	Dataset	Complexity	running time (one tree)
MTU	random	D	mn	0.0017
ESBU	rand. Edge Samp.	B	$mK\log(K)$	0.02
ESDU	rand. Edge Samp.	D	$mK\log(K)$	0.02
IESBU	Inertial Edge Samp.	B	$mK\log(K)$	0.02
IESDU	Inertial Edge Samp.	D	$mK\log(K)$	0.02
IESDU%	Inertial Edge Samp.	D	$(K = 165000)$	0.72
VCDU	Vert. Clust	D	up to $mn^2\log(n)$	0.92
IVCDU	Inertial Vert. Clust	D	up to $mn^2\log(n)$	0.92
CL	Chow-Liu	D	$n^2\log(n)$	1
CLBU	Chow-Liu	B	$mn^2\log(n)$	1



(a) 1000 samples. (b) 250 samples. (c) 100 samples.

Figure 2: Average performance of the algorithms described in Table 1 on 5 target distributions of 1000 variables times 10 datasets, with sample sizes decreasing.

Figures 2(a) and 3 display the resulting $\hat{D}_{KL}$ values for growing mixture sizes m on datasets of 1000 samples. From Fig. 2(a) we observe that vertex clustering (VCDU) seems far better than random edge sampling (ESDU).

The comparison is however not fair, because VCDU samples approximately 33% of all edges at each iteration, and ESDU only 1.4%. To provide a more accurate comparison, the latter was modified (ESDU%, in Figure 3 and Table 1) to use the same number of edges than VCDU. The results exposed in Figure 3 with this setting confirm that VCDU is also superior to ESDU%.

The inertial heuristic can be used to enhance both methods (IESDU, IVCDU) with nearly no additional complexity cost (see Table 1), leading to an increase in the improvement rate with the size of the model. Figures 2(a) and 2(b) show that IVCDU converges to a model slightly better than the CL tree (the best edges have been found, so the optimal tree is always learned), while the inertial randomized methods (IRB and IRS) with a complexity close to quasi-linear tend to approach CL when the number of mixture components grows and surpasses IVCDU which complexity is higher.

The same convergence can be observed for IESDU% (IESDU on 33% of edges instead of 1.4%) in Fig. 3, which also converges to the CL tree, albeit slower than IVCDU.

ESDU is degraded by the use of bagging (ESBU), and both methods yield performance only slightly better than random structures (MTU). We therefore conjecture that the low quality (as opposed to the low number) of edges used in ESDU introduces too much randomiza-

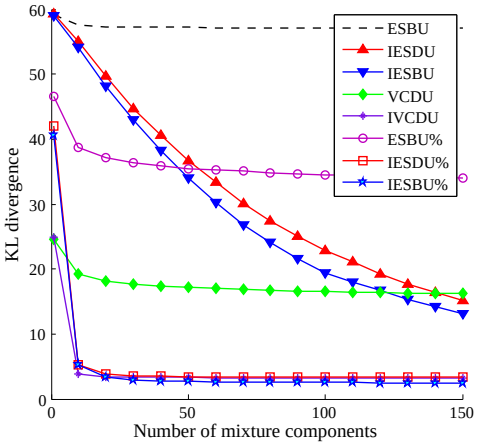


Figure 3: A comparison between vertex clustering and random edge sampling methods, where the latter are modified to use the same number of edges than the first, shows that vertex clustering is still clearly superior. (1000 samples)

tion to benefit from bagging. Indeed, when the quality of the edges considered increases over time (IESDU) bagging actually ameliorates the method. This hints towards that considering the use of bagging methods with CL trees or with the inertial vertex clustering could probably be improved by computing the best edges (i.e. significant on the original dataset) only once instead of repeating it for every new tree.

Experiments performed on sample sets of size 250 and 100 are reported in Fig. 2(b) and 2(c). Decreasing the number of samples from 1000 to 250 does not modify the results very much, as illustrated in Fig. 2(b). The main difference lies in the relative performance of all methods compared to CL algorithm, which seem to improve faster as m increases. This tends to indicate that randomization is increasingly more beneficial as the number of samples decreases.

The observation of the behavior of the bagging methods in Figure 2 further confirms this analysis. The gap between pairs of methods using the same algorithm on the original dataset or on bootstrap replicas (CL - CLBU, ESDU - ESBU, IESDU - IESBU) is widening when the number of samples is decreasing. Bagging of Chow-Liu trees (CLBU) is actually the best method on all dataset sizes. However, using bagging with inertial procedures and edge sampling yields also impressive results: the curve IESBU converges to CLBU in Fig. 2(c), while the IESBU algorithm has a much lower complexity than CLBU.

An alternative way to understand these results is in terms of over-fitting. From Fig. 2(c) we can see that IVCDU first surpasses CL after a few iterations, but the addition of subsequent terms worsens the mixture, which converges to the CL curve. Actually, at that point the model has fully learned the optimal tree, and it is repeatedly added to the model. The rise of the curve at that point signals the over-fitting. In addition, we can notice that MTU, the method using structures drawn independently from the dataset behaves better than most other methods in this context.

Observing the first tree of each model in the same figure, we can observe over-fitting again. IESBU (and ESBU) is better than IESDU (and ESDU). Likewise, the first term of CLBU is better than CL. A tree structure learned on a perturbed dataset leads to a graphical model that is more general.

This behavior is still present at 250 samples, but is no longer noticeable at 1000 : IESBU and IESDU start at the same point, and the first bagged tree is worse than the optimal.

The application of BDeu weights (not reported in this paper) to the methods presented here leads to similar conclusions. But we found out that many methods actually display worse performances in terms of accuracy when combined with such Bayesian weights. This is understandable, since weighting each structure by its posterior probability makes sense for MTU only, since asymptotically only in this context the mixture will converge to a canonical Bayesian method. The methods that benefit the most from those 'Bayesian' weighting scheme are IRSBU and IRSDU. The first terms (built with few information) are gradually eliminated in favor of those identified later on (where the inertial procedure has iteratively improved the quality of the edges).

5 Conclusions and future works

In this paper, we have compared several randomization methods aiming to approximate the Chow-Liu algorithm, with the objective of reducing its computational complexity in the context of learning mixtures of trees, and motivated by the variance reduction potential of randomization in the context of learning in high-dimensional problems.

Based on our results on synthetic experiments, we claim that, in real conditions, i.e. when the number of samples is much smaller than the number of variables, randomization is interesting for probability density estimation in the form of mixtures of Markov Trees. That interest actually increases when the number of samples goes down, or when the dimensionality of the space is increasing.

In addition, we have shown that exploiting the structure of the problem by focussing on strong edges leads to methods able to compete in terms of performance with more time-consuming procedures like bagging.

We therefore plan to keep investigating this approach. In particular, a candidate area for improvement is the transmission of knowledge between terms. Increasing the number of reused edges might speed up the convergence. Another direction of research would be the consideration of continuous variables, and the consideration of a priori known dependency/independency structures for the given problem.

Acknowledgments

This work was supported by FRIA/FNRS Belgium, Wallonie Bruxelles International, the French ministry of foreign and European affairs, the MESR in the framework of Hubert Curien partnerships, the BioMaGNet IUAP network of the Belgian Science Policy Office and the Pascal2 NOE of the EC-FP7. The scientific responsibility rests with the authors.

References

S. Ammar, Ph. Leray, B. Defourny, and L. Wehenkel. 2008. High-dimensional probability density estimation with randomized ensembles of tree structured Bayesian networks. In *Proceedings of the fourth European Workshop on Probabilistic Graphical Models (PGM'08)*, pages 9–16, Hirtshals, Denmark.

S. Ammar, Ph. Leray, B. Defourny, and L. Wehenkel. 2009. Probability density estimation by perturbing and combining tree structured Markov networks. In *Proceedings of the 10th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU 2009)*, pages 156–167, Verona, Italy.

S. Ammar, Ph. Leray, and L. Wehenkel. 2010. Subquadratic Markov tree mixture models for probability density estimation. In *19th International Conference on Computational Statistics (COMPSTAT 2010)*, pages 673–680, Paris, France.

V. Auvray and L. Wehenkel. 2002. On the construction of the inclusion boundary neighbourhood for Markov equivalence classes of Bayesian network structures. In Adnan Darwiche and Nir Friedman, editors, *Proceedings of the 18th Conference on Uncertainty in Artificial Intelligence (UAI-02)*, pages 26–35, S.F., Cal. Morgan Kaufmann Publishers.

L. Breiman. 1996. Arcing classifiers. Technical report, Dept. of Statistics, University of California.

C.K. Chow and C. N. Liu. 1968. Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, 14(3):462–467.

G.F. Cooper. 1990. The computational complexity of probabilistic inference using bayesian belief networks. *Artificial Intelligence*, 42(2-3):393–405, March.

T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. 2001. *Introduction to Algorithms, Second Edition*. MIT Press and McGraw-Hill.

S. Kullback and R. Leibler. 1951. On information and sufficiency. *Annals of Mathematical Statistics*, 22(1):79–86.

M. Meila and M. I. Jordan. 2000. Learning with mixtures of trees. *Journal of Machine Learning Research*, 1:1–48.

J. Pearl. 1986. Fusion, propagation, and structuring in belief networks. *Artificial Intelligence*, 29:241–288.

F. Schnitzler, Ph. Leray, and L. Wehenkel. 2010. Towards sub-quadratic learning of probability density models in the form of mixtures of trees. In *18th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN 2010)*, pages 219–224, Bruges, Belgium.