Mixtures of Tree-Structured Probabilistic Graphical Models for Density Estimation in High Dimensional Spaces

by

François Schnitzler

July 2012

University of Liège,
Department of Electrical Engineering
and Computer Science

A dissertation submitted in partial fulfilment of the requirements for the degree of Doctor in Philosophy (PhD) in engineering science

Abstract

The rapid progress of data acquisition technologies, and in particular the improvements in measurement resolution, allows to observe a stochastic process through the simultaneous monitoring of thousands to millions of random variables. Multimedia, bioinformatics and industrial processes are a few domains where sets of variables of this size are increasingly encountered. Processing such a large quantity of observations can benefit from the use of automatic procedures, in order to create a predictive model or to obtain valuable information about the process.

A widely used strategy to derive a model from observational data is the estimation of a multivariate probability density over the variables of the problem. Such a density can then be used to study the underlying stochastic phenomenon. When the number of variables to model is large, probabilistic graphical models can reduce the number of parameters necessary to encode a joint probability distribution by exploiting independence relationships between variables. However, when there are thousands of variables or more, the use of those models faces two problems. First, both learning these models from a set of observations and exploiting them is computationally problematic. Second, the number of recorded occurrences of the problem may be quite low with respect to the number of variables. This lack of observations might be a source of error when learning a model, because the model constructed may be influenced by the particular sampling of the realisations of the problem, and generalize badly on new, unseen realisations. This source of error is called the variance of the learning algorithm.

Within this context, the problem considered in the present thesis is to study and improve the scaling of probabilistic graphical models on high-dimensional problems, in terms of the number of variables. The approach selected is to use *mixtures of Markov trees*. Markov trees are a class of probabilistic graphical models that are constrained in terms of the independence relationships they can encode. Therefore, they are limited in the probability distributions they can model, but both learning and answering queries with such a model is considered to be computationally tractable. A mixture or an ensemble model is a weighted average of models. Such a mixture can be constructed to reduce the variance of a learning algorithm. In particular, the present thesis explores the possibility to build mixtures of Markov trees by using the *perturb and combine* framework. This approach has been quite successful in some areas of machine learning, and consists in randomizing a learning algorithm and combining the outputs resulting from a repeated application of the randomized algorithm on a given learning set.

There are three main parts in this thesis. In each part, algorithms are first developed and then tested on a set of problems. In the first one, I review existing algorithms for learning a single Markov tree and develop two new randomized algorithms for this task. In the second part, learning algorithms for mixtures of Markov trees are developed. Two different classes of algorithms are constructed. Algorithms of the first class construct a mixture of independent Markov trees. The best algorithm of this first class in terms of accuracy generates Markov trees by applying the Chow-Liu algorithm on bootstrap replicates of the original set of observations. Algorithms of the second class generate a sequence of trees, in the sense that each new Markov tree generated depends on previous models. These algorithms have been developed to approximate the best method of the first class, with the goal of reducing the computational complexity of learning without sacrificing accuracy. The third step of this thesis combines two types of mixtures of Markov trees: mixtures reducing the variance, and mixtures reducing the bias. The bias is another source of error, that originates from the inability of the learning algorithm to select, on average, the best possible model, e.g. because the class of models considered (here Markov trees) cannot encode the true model. In this third part, each tree of a bias-reducing mixture is replaced by a variance-reducing mixture of Markov trees. The objective is to reduce the variance of each term of the bias-reducing mixture, and hence of the bias-reducing mixture itself.

Finally, the ideas developed in this thesis are applied to another class of probabilistic graphical models, called tree-structured conditional random fields. Those models encode a conditional probability distribution rather than a joint probability distribution. A meta-algorithm to learn a mixture of these models is proposed, with the goal of reducing the variance.

Résumé

Le progrès rapide des technologies d'acquisition de données, en particulier l'amélioration de la résolution, permet l'étude d'un système stochastique via l'observation simultanée de miliers voir millions de variables aléatoires. La bioinformatique, le multimédia ou les processus industriels sont quelques-uns des domaines où se rencontrent des problèmes de cet ordre de grandeur. Leur analyse peut être facilitée par des procédures automatiques, permettant l'obtention d'un modèle prédictif ou d'informations sur le système.

Une stratégie répandue pour construire un modèle à partir de données observées est l'estimation d'une densité de probabilité multivariée sur les variables du problème. Celle-ci peut ensuite servir à l'étude du phénomène stochastique sous-jacent. Quand le nombre de variables est élevé, les modèles probabilistes graphiques permettent de réduire le nombre de paramètres nécessaires pour encoder une distribution de probabilité conjointe. Cependant, quand il y a des milliers de variables ou d'avantage, l'utilisation de ces modèles rencontre deux problèmes. Premièrement, tant leur apprentissage que leur utilisation posent des problèmes de complexité de calcul. Deuxièmement, le nombre d'observations du problème peut être faible par rapport au nombre de variables. Ce manque d'observations peut être source d'erreur lors de l'apprentissage : le modèle construit peut être influencé par l'échantillonnage des réalisations du problème, et mal se comporter sur de nouvelles réalisations. Cette source d'erreur est appelée la variance.

Dans ce contexte, le problème considéré dans cette thèse est l'étude et l'amélioration du passage à l'échelle des modèles probabilistes graphiques pour un grand nombre de variables. L'approche choisie est l'utilisation des *mélanges d'arbres de Markov.* Les arbres de Markov sont une classe de modèles probabilistes graphiques fortement contraints en terme des relations d'indépendence qu'ils peuvent encoder. Ils sont donc limités dans les distributions de probabilité qu'ils peuvent modéliser, mais l'apprentissage et l'exploitation de ces modèles sont considérés comme faisables algorithmiquement. Un mélange ou un ensemble de modèles est une moyenne pondérée de modèles. Un mélange peut être construit pour réduire la variance d'un algorithme d'apprentissage. En particulier, la présente thèse explore la possibilité de construire des mélanges d'arbres de Markov selon la technique du *perturb and combine*. Cette approche a eu quelques succès dans certains domaines de l'apprentissage automatique. Elle consiste à rendre un algorithme en partie aléatoire, et à combiner plusieurs résultats obtenus en appliquant plusieurs fois cet algorithme aléatoire sur un ensemble de données fixé.

Cette thèse comporte trois parties principales. Dans chacune, de nouveaux algorithmes sont développés et évalués empiriquement. Dans la première partie, je passe en revue les algorithmes de la littérature construisant un arbre de Markov, et puis j'en développe deux versions randomisées. Dans la seconde partie, des algorithmes pour apprendre un mélange d'arbres de Markov sont développés. Deux types d'algorithmes sont mis au point. Les algorithmes du premier type construisent un mélange d'arbres de Markov indépendants. Le meilleur algorithme de cette classe (en terme de précision) construit des arbres en appliquant l'algorithme de Chow-Liu sur des réplicats bootstrap de l'ensemble de données. Les algorithmes du second type construisent une séquences d'arbres, dans le sens où chaque nouvel arbre construit dépend des modèles précédemment obtenus. Ces algorithmes ont été développés pour approximer la meilleure méthode du premier type, dans le but de réduire le temps de calcul sans sacrifier la précision. La troisième partie de la thèse combine deux types de mélanges d'arbres de Markov : des mélanges réduisant la variance et des mélanges réduisant le biais. Le biais est une autre source d'erreur, causée par une incapacité de l'algorithme d'apprentissage à produire, en moyenne, le meilleur modèle, par exemple parce que la classe de modèles considérés (ici les arbres de Markov) ne peut encoder le vrai modèle. Dans cette troisième partie, chaque arbre d'un mélange réduisant le bias est remplacé par un mélange d'arbres de Markov réduisant la variance. Le but est de réduire la variance de chaque terme du mélange réduisant le biais, et donc la variance du mélange lui-même.

Enfin, les idées développées dans cette thèse sont appliquées à une autre classe de modèles probabilistes graphiques, les champs de Markov conditionnels en forme d'arbre. Ces mélanges encodent une distribution de probabilité conditionnelle au lieu d'une distribution conjointe. Un méta-algorithme pour apprendre des mélanges de ces modèles est proposé, avec l'objectif d'une réduction de variance.

Acknowledgments

First and foremost, I would like to extend my thanks to Prof. Louis Wehenkel, my thesis adviser, for offering me the opportunity to work on a PhD thesis and patiently guiding me towards its achievement. During these four years, he has shared with me his scientific and human experience, and has introduced me to the world of research. Even though he gave me the freedom to create and pursue my ideas, the present thesis would not have been possible without his suggestions, availability, kindness, help and passion.

I would like to express my gratitude to Prof. Philippe Leray and to his (then) PhD student Sourour Ammar, from the University of Nantes. They have been working on the same topic than myself, and a large part of my thesis benefited from the collaboration, encouragements, and critics they provided. I also want to thank Prof. Leray for welcoming me several times in Nantes, giving me the opportunity to have in-depth discussions with him.

I am also very grateful to Prof. Pierre Geurts and Prof. Damien Ernst from the University of Liège, who made me benefit from their experience in their respective areas of expertise. They were always eager to discuss any idea or answer any question I might have, offering precious suggestions and references. They have generously shared their enthusiasm and creativity.

I am also thankful to Dr. Jan Lemeire and Dr. Stijn Meganck from the Vrije Universiteit Brussel. They were kind enough to welcome me several times for a one-day visit to their lab in Brussels. I really appreciated these opportunities to discuss with them and other researchers in the field of probabilistic graphical models. Their spirit and ideas really inspired me.

In addition, I want to thank all the SYSTMOD unit, the Department of Electrical Engineering and Computer Science, the GIGA and the University of Liege. I was lucky to do my PhD in such a great research environment. I want to thank the academic staff, in particular Prof. Quentin Louveaux, Prof. Rodolphe Sepulchre and Prof. Eric Bullinger. I would like to acknowledge my colleagues, post-doctoral researchers or PhD students, in particular Bertrand Cornélusse, Raphaël Fonteneau and Florence Belmudes, but also Vincent Auvray, Julien Becker, Vincent Botta, Tom Cattaert, Anne Collard, Boris Defourny, Julie Dethier, Renaud Detry, Guillaume Drion, Dirk Fey, Fabien Heuze, Samuel Hiard, Van-Anh Huynh-Thu, Alexandre Irrthum, Arnaud Joly, David Lupien St-Pierre, Francis Maes, Alejandro Marcos Alvarez, Alexandre Mauroy, Gilles Meyer, Laurent Poirrier, Jonathan Pisane, Pierre Sacré, Alain Sarlette, Marie Schrynemackers, Olivier Stern, Laura Trotta, François Van Lishout and Da Wang. I also would like to thank the administrative staff of the department, in particular Diane Zander and Marie-Berthe Lecomte for their help.

I would also like to express my gratitude towards Dr. David Colignon, the manager of the computer cluster of the University of Liège. He was always available to talk about solutions, and he greatly facilitated my experiments.

I want to thank all the researchers I have met during the conferences I have attended, in particular Raphaël Mourad, Merel Rietbergen, Linda Van Der Gaag, Steven Woudenberg, Robert Castelo, Teemu Roos, Cassio P. de Campos, Yi Wang, Mark van Heeswijk and Amaury Lendasse.

For funding my research, I am very grateful to the FRIA (Fonds pour la Formation à la Recherche dans l'Industrie et l'Agriculture) from the Belgium fund for scientific research, to the PASCAL2 network, to WBI and to the PAI (Poles d'Attraction Interuniversitaires) BIOMAGNET (Bioinformatics and Modeling: from Genomes to Networks). Finally, I would like to thank any colleague I might have forgotten, and hope they can forgive me.

I also want to express my deepest gratitude to my parents, for the education they provided me with and for the love they keep giving me. I also want to thank my three sisters, my grandparents and all the other members of my family for their support throughout my studies and this thesis.

I am also grateful to my dear friends Bertrand, Pierre, Vincent, Manu, Lucie and Sophie for all those gaming sessions shared. It is undoubtedly with you that I have developed an interest for probability theory. Many thanks as well to Roxanne, Frédérique and Elisabeth for their continuous support from afar. To them, to all the others, thank you for all those sometimes difficult moments where you lifted my spirit.

And off course, thank you to Eglantine for her recent but still greatly appreciated support.

Contents

List of Figures

List of Algorithms

List of Notations

$\mathcal{X}, \mathcal{Y}_i$	A single variable.
x, y_i^j	A single configuration that a single variable can take.
$\boldsymbol{\mathcal{X}}$	A set of variables. It may also be equivalent to $Val(\boldsymbol{\mathcal{X}})$, depending on the context.
$\mathbf{x}, \mathbf{y}$	A possible configuration of a set of variables.
$Val(\mathcal{X})$, $Val(\boldsymbol{\mathcal{X}})$	The set of all configurations that a single variable or a set of variables can take.
$\vert . \vert$	Cardinality of a set or tuple.
$\mathcal{G}$	A graph (or a probabilistic graphical model).
$\mathcal{T}$	A tree graph (or a tree-structured probabilistic graphical model).
$E(\mathcal{G})$	The edges of $\mathcal{G}$.
$\boldsymbol{\mathcal{G}}$	A tuple of graphs (or of probabilistic graphical models).
$\boldsymbol{\mathcal{P}}a_{\mathcal{G}}^{\mathcal{X}_i}$	The set of parent variables of $\mathcal{X}_i$ in $\mathcal{G}$.
θ	The set of parameters of a probability distribution, in particular of a probabilistic graphical model.
$\theta_{i,x_i^j \vert \mathbf{a}}$	$\mathbb{P}_{\mathcal{G}}(\mathcal{X}_i = x_i^j \vert \boldsymbol{\mathcal{P}}a_{\mathcal{G}}^{\mathcal{X}_i} = \mathbf{a})$.
$\theta_{i,\mathcal{X}_i \vert \mathbf{a}}$	$\{\theta_{i,x_i^j \vert \mathbf{a}}\}_{j=1}^{Val(\mathcal{X}_i)}$.
$\theta_{i,\mathcal{X}_i \vert \boldsymbol{\mathcal{P}}a_{\mathcal{G}}^{\mathcal{X}_i}}$	$\{\theta_{i,\mathcal{X}_i \vert \mathbf{a}^j}\}_{j=1}^{Val(\boldsymbol{\mathcal{P}}a_{\mathcal{G}}^{\mathcal{X}_i})}$.
$\phi(\boldsymbol{\mathcal{D}})$	A factor defined on the variables $\boldsymbol{\mathcal{D}}$.
Z	A normalization constant for a factor graph.
$\mathbb{P}$	A probability distribution.

$\mathbb{P}_{\mathcal{G}}$	A probability distribution encoded by the probabilistic graphical model $\mathcal{G}$, where the parameters of the model are left implicit.
$\mathbb{P}_{\mathcal{G}}(.\vert\theta)$	A probability distribution encoded by the probabilistic graphical model $\mathcal{G}$, where the parameters are explicitly specified.
$\mathbb{P}_{\boldsymbol{\mathcal{T}}}$	A probability distribution encoded by a mixture $\boldsymbol{\mathcal{T}}$ of Markov trees, where the parameters of the mixtures are left implicit.
m	In general, the number of Markov trees in a mixture.
p	The number of variables.
N	The number of observations in a learning set.
D	A learning set of observations.
$\mathbf{x}_{D_i}$	The configuration of values taken by the variables $\boldsymbol{\mathcal{X}}$ in the i^{th} observation of D.
$\mathbb{P}_D(.)$	The empirical probability distribution observed in the learning set D.
$N_D(\mathbf{x})$	The number of observations in D where $\boldsymbol{\mathcal{X}} = \mathbf{x}$.
ρ	The type I error associated to a hypothesis test.
λ, μ	In general, the weights of a mixture model.
$H_D(\boldsymbol{\mathcal{X}})$	The entropy of $\boldsymbol{\mathcal{X}}$, according to $\mathbb{P}_D(\boldsymbol{\mathcal{X}})$.
$I_D(\boldsymbol{\mathcal{X}};\boldsymbol{\mathcal{Y}})$	The mutual information between $\boldsymbol{\mathcal{X}}$ and $\boldsymbol{\mathcal{Y}}$, according to $\mathbb{P}_D(\boldsymbol{\mathcal{X}},\boldsymbol{\mathcal{Y}})$.
$D_{KL}(\mathbb{P}_1\vert\vert\mathbb{P}_2)$	The Kullback–Leibler divergence of $\mathbb{P}_2$ with respect to $\mathbb{P}_1$.
$\text{logll}_D(.)$	The log-likelihood, estimated based on the set D of observations. The subscript may be omitted.
$\text{nlogll}_D(.)$	The negative log-likelihood, estimated based on the set D of observations. The subscript may be omitted.

Chapter 1

Introduction

The main preoccupation of this thesis is discrete probability density estimation. More precisely, the research presented in this document is the application of the perturb and combine principle to automatic learning of probabilistic graphical models, an encoding for a probability distribution. Bound together, those two frameworks lead to the construction of mixtures of simple probabilistic models. Both topics are introduced in Section 1.1, where this research is also motivated. The objectives of the thesis are presented in Section 1.2. The organisation of the present manuscript is detailed in Section 1.3. This introduction is closed by spelling out the main contributions of the present thesis, in Section 1.4.

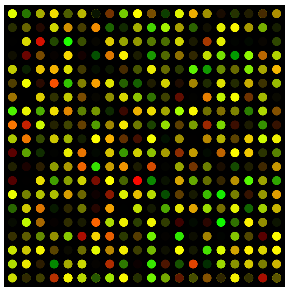
1.1 Context and Motivation

1.1.1 Machine Learning

Machine learning is a subfield of artificial intelligence that focuses on the automatic (i.e. by computers and algorithms) exploitation of empirical data, whether they come from databases or from sensors. Machine learning procedures can replace or assist experts in the study of the problem of concern, which can be overly complex for a human or a team of humans to tackle alone.

Applying machine learning algorithms to a data set can serve various purposes, e.g. to unveil the underlying structure, behavior or properties

Microarray technology



A DNA microarray can measure the expression level of thousands of genes in a single experiment. It is composed of thousands of microscopic DNA dots attached to a substrate. Each dot is made of a small quantity of a known and specific DNA sequence.
To measure gene expression levels in a medium, a copy of interesting genetic material it contains is made, using building blocks attached to a fluorescent molecule. These copies are then applied on the microarray, and each molecule binds onto a complementary sequence if it composes one dot of the microarray. Once the solution is washed away, the fluorescence of the dots can be measured to quantify gene expression level. Two different mediums (such as healthy and sick) can be compared by coloring each one with a different color. The result is illustrated in the above figure.

of a given system, or to build a black box capable of predicting valuable information in any new (unseen) situation.

Consider as an example the study of a complex genetic disease. With the developments of the microarray and sequencing technology, it is increasingly easy to identify hundreds of thousands of genetic variations in the DNA code of a human being and to associate them to his medical condition in a database, or to measure the expression level of thousands of genes simultaneously. However dealing with this massive amount of information is hardly possible without the assistance of automatic learning procedures.

Machine learning algorithms could exploit such a database containing the genetic variations from several individuals to achieve several objectives. It could be used to extract knowledge. Can the data be expressed as a function of a smaller number of elements? What are the genes involved in the disease? Can it be partitioned into different pathologies? But the same information can also be processed to build a predictive model. Based on his genetic variations, what is the probability that a new individual is or will be affected by the disease? What treatment is the most likely to be successful on him?

In a machine learning framework, a data set is often characterised in terms of its number of samples or observations, i.e. the number of problem instances that were measured, and in terms of its number of variables (dimension), i.e. the values measured in each sample. In the example, the number of samples would be the number of individuals and the variables would be the genetic variations.

One way to perform inductive reasoning is through the use of one or several joint probability distribution(s). Indeed, such a distribution can be used for inference, e.g. to obtain the most probable configuration of variables or to compute the conditional probability distribution of some unobserved variables given known values for others. Therefore constructing a distribution from a set of samples has received considerable attention in the machine learning community, crossing with the field of statistical estimation. In this case a distribution is typically defined on all the problem variables and encodes the distribution of said variables as observed in the samples available.

For example, one approach to predicting a binary variable (the target) based on the values taken by the others (then called the features) is to learn two joint distributions on all the features, i.e. one distribution for each value of the target variable. Each distribution is learned conditionally on the associated value of the target variable. This distribution is constructed using only the observations in which the target variables take the value considered. When a prediction must be made, one possibility is then to output the target value associated to the distribution giving the highest probability to the new values observed for the features.

1.1.2 Probabilistic Graphical Models

Probabilistic graphical models (PGM) can efficiently encode a joint probability distribution over a large set of variables. For this reason, they have seen an increasing use in machine learning, where the number of variables can be challenging. A PGM is composed of a graphical structure and a set of parameters. The graph defines a compact representation of the distribution based on the relationships between the variables, and the parameters quantify it [Pea88, NWL$^+$07, Dar09, KF09].

In this work, I will use such models extensively, and in particular Bayesian networks (BN). In this class of PGMs, the nodes of the graph $\mathcal{G}$ are labeled by the variables $\boldsymbol{\mathcal{X}}$ of the problem and provide information about the struc-

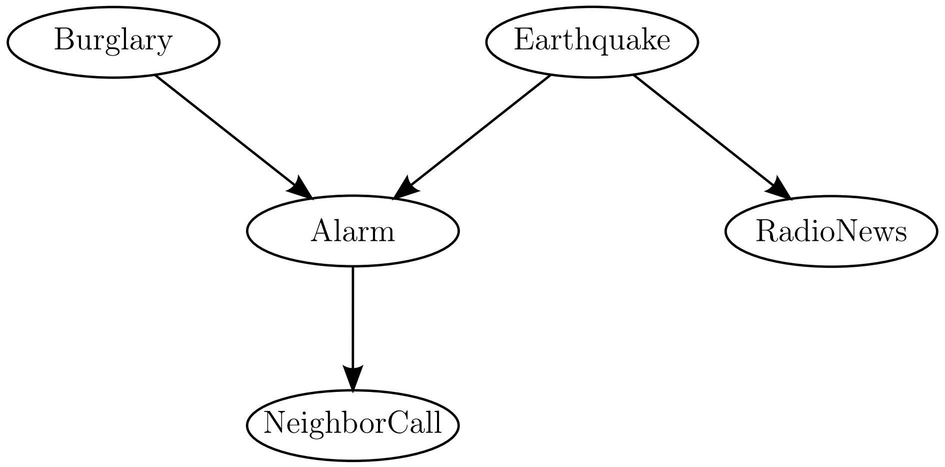


Figure 1.1: The burglary network is a Bayesian network modelling a home alarm system.

ture (in terms of probabilistic independencies) of the problem, and the parameters quantify the conditional probability distribution of each variable $\mathcal{X}_i$ conditioned on its parents $\boldsymbol{\mathcal{P}a}_{\mathcal{G}}^{\mathcal{X}_i}$ in the graph. Hence, a Bayesian network encodes a factorization of the joint probability distribution by a product of conditional distributions:

$$\mathbb{P}_{\mathcal{G}}(\mathcal{X}) = \prod_{i=1}^{p} \mathbb{P}_{\mathcal{G}}(\mathcal{X}_i | \boldsymbol{\mathcal{P}a}_{\mathcal{G}}^{\mathcal{X}_i}) \ . \tag{1.1}$$

Consider as an illustration the burglary network, due to [Pea88] and displayed in Figure 1.1. It is composed of 5 binary variables and models a sensitive home alarm system that can be triggered by either a burglary or an earthquake, however minor. When the alarm rings, a neighbor may call the home owner at work, and an earthquake may lead to a radio announcement.

Due to this encoding, a visual inspection of the graph of the BN can provide insight into the inner structure of the distribution. In the burglary example, the network reveals that the variables “Alarm” and “RadioNews” are independent given the variable “Earthquake”, because the latter is the parent of the two former and there is only one path between them. The probability that the neighbor calls when the alarm rings is also not influenced by the occurrence of an earthquake or of a burglary.

Since the numeric encoding of the distribution exploits its structure, the number of parameters needed is smaller than in a contingency table, a table containing the probability of every joint configuration of the variables. The example contains five binary variables. Therefore a contigency table for the joint probability distribution would contain $2^5 = 16$ parameters, minus one since they must sum to one. However the Bayesian network stores only 10 independent probability values.

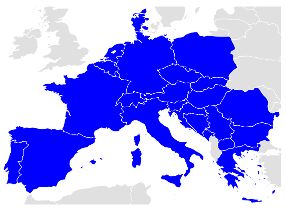
In practice, exploiting the structure of the distribution to facilitate its encoding is necessary for large sets of variables. Storing a probability value on a computer as a floating point number typically uses 4 bytes of memory. If the variables considered are limited to binary values and assuming it can be completely used, a RAM memory with a capacity of 1 GB can store 2^{28} probability values, and can therefore store a contingency table for up to 28 binary variables. Under similar hypotheses, a hard drive of 1TB can hold the contingency table of 38 variables. Finally, today's worldwide hard drive capacity can be estimated at 1ZB (2^{70}B)[1]. Assuming it could all be used to store a single contigency table, it could not be defined on more than 68 variables. These numbers are rather small, so more efficient encoding schemes such as Bayesian networks are required.

A BN can encode any joint distribution, ranging from a set of independent variables (by a structure without edges) to the opposite case where all variables directly influence each other (completely connected graph). As with any probability distribution, it is possible to compute from a Bayesian network marginal probabilities (the probability distribution of a subset of variables) or conditional probabilities (i.e. the probability distribution of a subset of variables given the values of another subset of variables). In the framework of PGMs, these and similar operations are designated by the generic term of inference.

A BN can be specified by an expert, but also automatically constructed from a set of observations by a machine learning algorithm. When the latter method is chosen, performing inference on a problem can be seen as a two-step process: learning the network based on the observations, and performing inference. A BN can also be used to extract the independence relationships between the variables in the problem, but this thesis only focuses on learning probabilistic models for inference.

[1]This estimation is based on the methodology of [HL11]. The cumulated hard drive capacity produced in last five years is 1.2ZB (based on estimations made in 2006).

Synchronous grid of Continental Europe



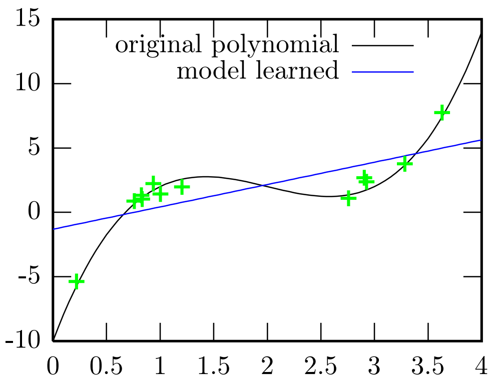
The synchronous grid of Continental Europe is the interconnected electrical grid of 24 European countries. The whole network is synchronized at 50Hz. Connecting national grids is a mean to share production, more easily balance load and consumption (and thus maintain frequency) and to better exploit available energy resources. When a country lacks power generation capacity, it can easily import power from its neighbours.

However, applying these techniques is challenging on problems where the number of variables is large (a few thousands). For example, one may wish to model a probability density over the voltages at the nodes of a power network (at the extra-high voltage level, there are about 16 000 transmission nodes in the synchronous grid of Continental Europe) or to model a probability density over the expression level of genes or proteins (there are approximately 21 000 genes and 1 000 000 proteins in a human being).

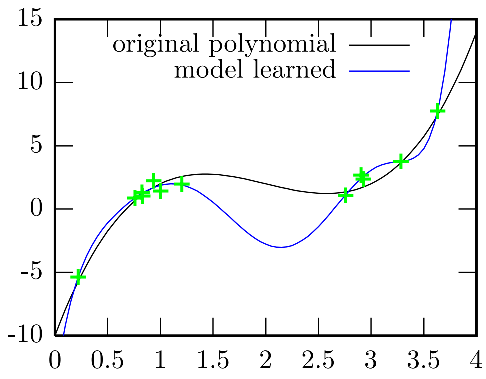
Learning and inference, which will both be presented with more details in Chapter 3, are computationally very expensive to perform in general for large problems (actually they belong to the class of non-deterministic polynomial-time hard problems). In practice, they are not applicable on problems of more than a thousand variables [Auv02, EG08]. Constraining graphical structures of BN can improve algorithmic complexity, but decreases modelling power. However, this decrease may be beneficial in terms of accuracy of the estimated distribution with respect to the original distribution. A constrained model is likely to be less sensitive to noise in the observations.

1.1.3 Bias and Variance

In the context of learning, a problem is the usually extremely low number of observations with respect to the number of variables. Intuitively one would wish to have an infinite number of observations. However, obtaining at least one occurrence of every possible configuration implies that the number of observations would possibly increase exponentially with the number of



(a) The learned model is a line.



(b) The learned model is a polynomial of degree 7.

Figure 1.2: The overfitting problem is illustrated on a regression problem. A polynomial of degree 3 is sampled to generate a learning set of 12 noisy observations. Two models of different complexity are learned on each of these learning sets by minimizing the mean square error to the observations. The first model is too simple, whereas the second is too complex and suffers from overfitting.

variables. However, the larger the number of variables the more expensive it is to acquire and store observations, and in very high dimensional settings the number of observations is typically smaller than the number of variables. This is called “the curse of dimensionality”. As a consequence of the lack of observations, selecting the best model can be very difficult. Discriminating between different alternatives is not easy, and special care must be taken that the one selected does not *overfit* the data. Overfitting means that the model does not generalize well, and will behave worse during inference on new, unobserved observations than a simpler model. This phenomenon is illustrated in Figure 1.2, where a polynomial is approximated by a line (left) and a polynomial of degree 7 (right). In the latter case, the model used is too complex and fits the observations too closely, leading to a large error, e.g. in the middle of the figure.

To limit the negative impact of the lack of samples, a classical solution is to impose constraints on the model structure, to lower the number of parameters. Constraining the model is a balance between its capacity to model information and the risk it fits the observations too much. This is referred to as the **bias-variance trade-off**.

Bias refers to the difference between the average model learned (over all possible sets of observations of a given size) and the true model. This results from the inability of the learning algorithm to recover this true model because

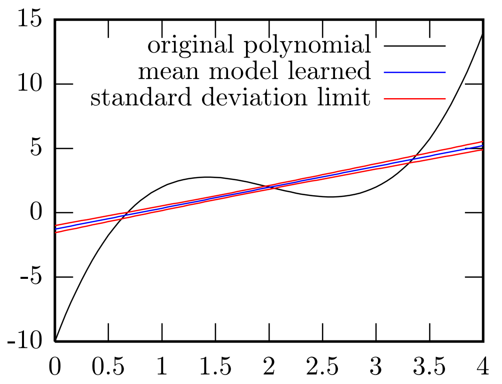
1. the models considered by the algorithm do not contain the true model,
2. the learning algorithm does not identify the best model among those considered, and instead returns a worse model.

Variance is related to the phenomenon of overfitting that was mentioned above. When the model considered has many degrees of freedom, and when few observations are available, it is possible to construct a model that perfectly fits these observations. Applying the same algorithm on another set of observations of the same problem would also fit them perfectly. The two models, however, are likely to be quite different. This is the variance. Such models do not generalize well on unseen instances because they tend to model sampling noise in the observations. They are said to be overfitting.

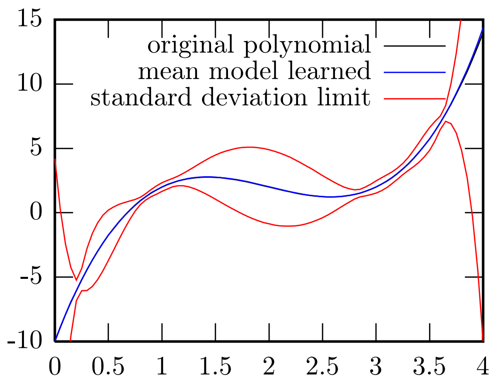
Bias appears when the models considered are not adaptable enough, variance when they are overly adaptable. Selecting the adaptability of the model is therefore a trade-off between both sources of errors, with an intermediate adaptability giving the best result. If it is increased, variance grows more than the bias decreases and the overall error gets bigger, and vice versa. This is illustrated in Figure 1.3. On the regression problems already considered, several models belonging to the classes of polynomials of degrees respectively 1 and 7 are constructed on different learning sets. Both the averaged model learned and the variability of the models are displayed. When the models are simple (degree 1), the variance is small but the bias large, and when the models are complex (degree 7), it it the opposite.

1.1.4 Mixture Models and Perturb and Combine

The bias-variance trade-off applies to the learning of probabilistic graphical models. An additional consideration relevant to the complexity of the class of models considered is algorithmic complexity of both learning and



(a) The learned model is a line. The bias is large.



(b) The learned model is a polynomial of degree 7. The variance is important.

Figure 1.3: The bias variance trade-off is illustrated on the regression problem already studied in Figure 1.2. 1000 learning sets are generated by repeatedly sampling the polynomial on 12 given abscissa.
Two models of different complexity are learned on each of these learning sets by minimizing the mean square error to the observations. The first model (a) is too simple, whereas the second one (b) is too complex. The mean and the standard deviation of the set of 1000 models are illustrated, for both complexities. The distance between the red lines equals twice the standard deviation.

inference. As mentioned above, reducing the set of candidate structures can make learning algorithmically easier, and the search can also be limited to structures for which inference can be performed under a given time budget.

In this thesis a further step is taken and mixtures of simple PGMs are considered. Instead of using one potentially complex structure, the problem is represented by a combination of simple PGMs (typically Markov trees), each modelling a joint probability density on all variables. To each one of those m terms is associated a (positive) weight, here denoted by w. The probability density defined by the mixture is the weighted average of the different densities encoded by each term:

$$\mathbb{P}(\boldsymbol{\mathcal{X}}) = \sum_{i=1}^{m} w_i \mathbb{P}_i(\boldsymbol{\mathcal{X}}), \tag{1.2}$$

$$\sum_{i=1}^{m} w_i = 1 \text{ and } \forall i : w_i \geq 0. \tag{1.3}$$

This approach reduces the shortcoming of both simple and complex graph structures: the structures are simple, therefore the algorithms may remain simple as well. However, combining several simple models allows in principle to represent richer classes of densities.

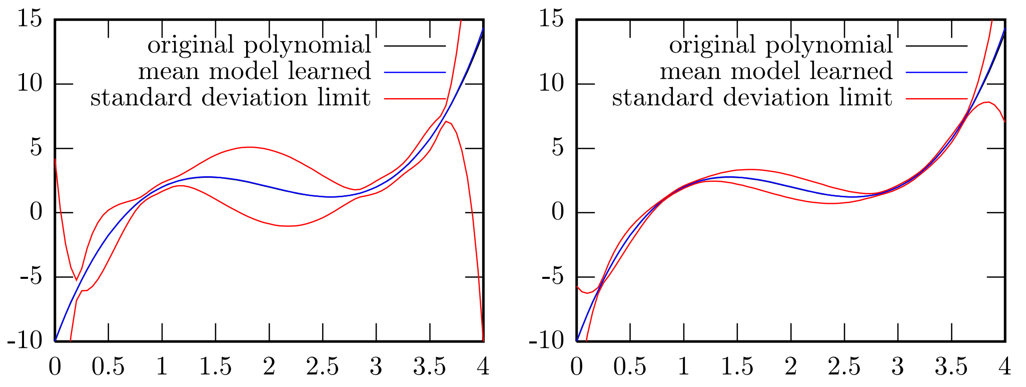
Such mixtures can be constructed to either reduce the bias or the variance. Each individual model can be viewed as an alternative distribution and averaging them as considering either different mechanisms of the problem, a way to deal with a complex problem, or different possibilities between which the lack of samples does not allow to discriminate. In the first setting, the mixture is an attempt to bridge the gap between the class of simple models and the complex structure they cannot represent, i.e. to reduce the bias. In the second, the mixture is trying to decrease the error introduced by the lack of samples, the variance. This thesis focuses on mixtures for reducing the variance, and in particular on mixtures constructed based on the perturb and combine method.

Perturb and combine is a generic framework for building mixture models based on that second principle. Its main idea is to, on the one hand, randomize the learning algorithm used to construct a model from a learning set and, on the other hand, to combine in some appropriate fashion an ensemble of models obtained by multiple iterations of the perturbed algorithm over the given set of observations.

Figure 1.4 illustrates the intuition behind the model averaging approach on the regression problem already considered. While the 10 different polynomials of degree 7 all suffer from a larger variance, averaging them results in a closer approximation of the original function.

1.2 Objectives of the Thesis

The research presented in this thesis concerns learning mixtures of simple PGMs for inference. The two main objectives of this research were to extend this framework by developing new learning algorithms, especially by approaches related to the perturb and combine framework, and to assess the interest of both the existing and new methods in realistic problems.



(a) The learned model is a polynomial of degree 7, as in Figure 1.3b.

(b) The learned model is an average of 10 polynomials of degree 7, the standard deviation decreases.

Figure 1.4: The reduction of variance due to a mixture model is illustrated on the regression problem already studied in Figure 1.3. 1000 polynomials of degree 7 are learned, each based on a different set of 12 observations (with fixed abscissa). The standard deviation of these 1000 models is contrasted against the standard deviation of 1000 mixtures, constructed by averaging 10 of these polynomials. The distance between the red lines equals twice the standard deviation.

1.2.1 Improvements in Mixtures of Simple PGMs

In order to develop new mixtures of simple PGMs for high-dimensional learning, several points are addressed in this thesis. Scalability of the algorithms, both for inference and learning, and accuracy of the distribution estimated are the main focus of the research.

Perhaps the most fundamental question to address is the selection of the class of models composing the mixtures. What is the most suitable class of models, and what are the alternatives, given the focus on high-dimensional applications? Because of the complexity of the algorithms associated to these models, Markov trees, a class of BNs where each variable has at most one parent, are very appealing models, and have already received much attention. Indeed, algorithms for learning such a model are roughly quadratic in the number of variables, while inference is of linear complexity. The present thesis will therefore focus on these models. More details about that will be given in Chapter 4.

As a consequence, the quest for new learning methods starts by a review of existing methods for building such mixtures, for methods targeting either the bias, the variance or both.

Based on the strength and weaknesses of other methods uncovered in the aforementioned analysis, three main targets are identified for the perturb and combine procedure: the single tree learning algorithm, the tree learning algorithm in the context of a variance reduction mixture and the mixture learning algorithm. The randomization of each of these three levels is investigated in respectively Chapters 5, 6 and 7 of this manuscript.

1. Learning the structure of a single Markov tree is an essential building block of most algorithms for learning a mixture of Markov trees. The algorithmic complexity of this operation is roughly quadratic in the number of variables and is driven by the number of pairs of variables considered. In the context of mixtures, this algorithm will be applied multiple times. As a consequence, the resulting complexity can be problematic when the number of variables increases. Therefore randomizations were carefully designed to decrease the complexity while retaining as much accuracy as possible. The key idea is to guess which pairs of variables are interesting for the tree. The resulting algorithm is compared to the original one, both for learning a single tree and a mixture. While in both situations the new algorithm does not match the precision of the unmodified version, it outperforms a random selection of candidate edges both in accuracy and in complexity.

2. Considering the generation of the mixture as a whole rather than the repetitive learning of a tree opens up the possibility to transfer information from the trees already generated to the learning procedure creating an additional tree. By transposing in this context the concept of selecting interesting pairs of variables, several randomized algorithms are developed. A comparison to other randomized methods shows an accuracy close to the state of the art for a much lower learning cost, and a far better accuracy than other randomized methods with a similar complexity.

3. The methods studied in the two previous steps focus on improving over a single Markov tree, and in particular on reducing the variance. However, it is possible to go one step further and to consider improving

the other class of mixtures of Markov trees, i.e. those targeting the bias of Markov trees. In those mixtures, each tree can be viewed as a distinct mechanism of the problem. The variance of each of these Markov trees may increase the error of the model. To attempt to decrease the variance of each tree of the model, the two classes of mixtures can be combined by replacing each tree of the bias reducing mixture by a variance-reducing mixture, leading to a two-level mixture of Markov trees.

1.2.2 Empirical Evaluation of Algorithms

The proposed algorithms must be evaluated, both from the point of view of computational complexity and from the point of view of modeling accuracy. In this work, this evaluation is done mostly in an empirical way, by applying the proposed learning algorithms on both synthetic and realistic datasets, and by comparing the resulting models with those obtained by other methods.

There exist measures of similarity between distributions to evaluate the fit of a model to a given probability density. They are used in this thesis to quantify the quality of the models learned by the algorithms developed, and of the alternative approaches. The scores obtained for both are compared to establish the merits of both methods. These alternative methods are existing algorithms for learning mixtures of Markov trees, but also regularization of a single Markov tree, another method for reducing the variance. This latter method attempts to reduce the complexity of a single model to lower its variance.

1.2.3 Application to Multitarget Classification

Applying the algorithms developed to solve realistic problems could give an additional insight into the interest of the approaches studied as a tool for machine learning. Therefore, this thesis proposes to apply the techniques developed to multitarget classification.

Multitarget classification is a class of machine learning problems where the value of a set of several output variables must be predicted based on the known values of a set of input variables. Taking into account the relationships between output variables is mandatory to obtain good predictive performance. To do so it is possible to model a joint probability distribution on those output variables, conditioned on the inputs. Conditional random fields are a specialized class of probabilistic graphical models that encode a conditional joint distribution. They can thus be used for multi-target classification. However, learning such a model is even more complex than learning a BN.

As a last contribution of this thesis, I propose in Chapter 8 to adapt the concepts developed in the present thesis, in order to build mixtures of tree-structured conditional random fields. The implementation and evaluation of this approach is left for future work.

1.3 Organization

This manuscript is divided in three parts, plus some appendices. The first part is mostly a background section, introducing general notions of machine learning, PGMs and mixture models. The second part focuses on the development of new methods for learning mixtures of Markov trees and on their empirical evaluation. The third part proposes an approach to apply those methods to multitarget prediction. Finally the conclusion gives a synthetic overview of the main scientific advancements contained in the thesis and presents some possible ways to further expand this research. The contents of the three main parts and the appendices are described below.

1.3.1 Part I: Background

Chapter 2 presents several machine learning problems. The evaluation of machine learning algorithms is also discussed. Important notions such as bias and variance are introduced more formally. Mixtures of models are also introduced. After these models are defined, two frameworks for building them are distinguished based on the component of the error they target, i.e. either bias or variance. Those notions are then illustrated by a few machine learning examples. This chapter ends by relating the topic of this thesis with respect to the information provided.

Chapter 3 provides an overview of probabilistic graphical models. Different general classes of models are first reviewed and their capacity of representation discussed. Then, both learning and inference are discussed:

problem statement, complexity and typical approaches. Finally, the content of this thesis is positioned with respect to existing work on PGMs.

Chapter 4 is concerned with the application of mixtures to Bayesian networks. This chapter provides a closer look at mixtures of simple probabilistic graphical models, and in particular mixtures of Markov trees, making the transition to part II.

1.3.2 Part II: New Algorithms for Learning Mixtures of Markov Trees

This part of the thesis first explores three different applications of the perturb and combine principle to build mixtures of Markov trees, before briefly considering other models. Note that the problems on which the algorithms are evaluated are precisely described in the appendices.

In Chapter 5, this randomization focuses on the construction of a single tree. The chapter starts by a review of existing algorithms for learning Markov trees and assesses their strengths and interest in the context of high-dimensional density estimation. First, regularization of the Chow-Liu algorithm is considered, as it is an alternative to mixtures for reducing the variance. Standard methods based on regularization are developed to compete against mixtures during the evaluation. Two new randomized algorithms are then presented. Their accuracy and algorithmic complexity are evaluated against the original algorithm for learning a tree.

Going one level higher, Chapter 6 contains algorithms for learning mixtures of Markov trees for variance reduction. After a general meta-algorithm is described, this chapter is divided into two parts. The first part focuses on mixtures of independent trees, while the second studies randomization schemes where the construction of each new tree exploits the computations performed for building the previous trees. Existing randomization schemes are studied, and based on that analysis new algorithms are developed. These algorithms are then compared against existing methods and against the regularization standard developed in the previous chapter, on both synthetic and more realistic problems.

Chapter 7 still focuses on mixtures of Markov trees, but merges the two frameworks for building them defined in Chapter 4. Rather than replacing one Markov tree by a mixture of such trees, as in previous chapters, 2-level mixtures of Markov trees are now constructed: the first level reduces the bias while the second reduces the variance. This meta-heuristic is first motivated and discussed before this concept is instantiated, tested on synthetic and realistic problems and compared to traditional mixtures.

1.3.3 Part III: Perspectives for Multitarget Prediction

This part of the thesis is composed of a single chapter providing a proposal for applying the proposed algorithms of Part II to multitarget prediction.

Chapter 8 considers conditional random fields (CRF) and proposes a meta-algorithm for building mixtures of tree-structured CRFs. The CRF models, specialized for the joint prediction of several output variables, are first described. Then, the different strategies for learning them are presented, and the interest of tree-structured CRFs is discussed. Next, a meta-algorithm for learning mixtures of such models is proposed and its main ingredients are discussed, in particular the question of aggregation. The implementation and validation of these ideas is however left for future work.

1.3.4 Appendices

The appendices provide the following information apart from the core content to avoid disrupting reading.

Appendix A characterizes the different problem instances used to evaluate the algorithms studied in this thesis. The generation process of synthetic problems is described, and the origin of realistic problems specified.

Appendix B provides elements of graph theory and related terminology. Different types of graphs and algorithms used in this text are described, and this chapter can be used as a reference.

Appendix C provides a short introduction to the theoretical notion of computational complexity, and reports on the main results from the literature concerning the computational complexity of various problems of learning and inference with probabilistic graphical models.

1.4 Contributions

In this section the main contributions of the thesis are stated precisely. While a full discussion about these points is of course present in the main body of this manuscript, a quick comparison with existing work is already given here to clearly highlight the novelty of the developments presented.

- In Section 2.4, I provide a detailed analysis of the bias-variance compromise in the context of density estimation. Our analysis, based on the comparison of two different decompositions of the Kullback-Leibler divergence, provides a theoretical motivation for arithmetic (rather than geometric) averaging of probability density models, when the objective is to reduce their variance. These ideas are still unpublished at the moment of submitting this thesis.

- Two algorithms for learning a randomized maximum likelihood tree are detailed in Chapter 5. While the original Chow-Liu algorithm [CL68] has already been accelerated for sparse samples [Mei99] and approximated for learning sets containing many samples using probabilistic bounds [PM06], this is to the best of my knowledge the first time a heuristic has been developed to approximate the Chow-Liu algorithm by discarding edges either randomly or based on previously considered edges. Note that random edge sampling was also proposed in [ALW10b] and generation of random tree structures has been considered in [ALDW08].

- These algorithms are used to construct mixtures of Markov Trees based on the perturb and combine framework. This led to one publication [SLW10] and is described in Chapter 6.

- Several approximations of the mixture of bagged Markov trees are proposed in Chapter 6. The algorithm approximated can be found in [ALDW09b]. Other approximations have also been developed in [ALW10b], but the experiments presented in the chapter show the interest of my new methods in terms of accuracy, although their runtime is for some of them slightly more difficult to control. Those results were published in [SAL$^+$11].

- The algorithms mentioned above are extensively evaluated. To the best of my knowledge, I was the first to:

  - analyze the effect of the number of samples (for a fixed number of variables) on the performance of variance-reducing mixtures of

  Markov trees. This analysis has been performed e.g. in [SLW10, ALSW10, SAL$^+$11].
  - evaluate them on realistic data sets, whereas previous works focused only on synthetic data sets [ALW10b, ALDW09b].
  - compare them to a regularized Markov tree, a single tree penalized for its complexity during learning, so that the complexity of the model is tuned to the number of observations available.

- The two frameworks are combined in Chapter 7, resulting in a 2-level mixtures. The two types of mixtures had already been combined in [KS07], but their approach is of cubic complexity whereas mine is quadratic. The approach is different as well: here a maximum-likelihood mixture is generalized by replacing each term by a variance reducing mixture while in [KS07], they generate a variance reducing mixture of a bias reducing mixtures. An early version of this work was presented in [SW11], and a paper has been recently submitted to the Sixth European Workshop on Probabilistic Graphical Models (PGM 2012).

- The meta-algorithm for learning mixtures of Markov trees is adapted to tree-structured conditional random fields (CRF) in Chapter 8. Mixtures of tree-structured CRFs have already been discussed in [CG10, POB09] in the context of multitarget classification. However, [CG10] actually conditions the structure of the tree to the features and thus performs inference on one tree only, and [POB09] builds those trees as an approximate method for inference in a more complex CRF. The novelty of my approach is that a mixture of tree-structured CRFs will be learned from a set of observations, that inference will be carried on on all CRFs constructed, and that the results will be aggregated.

Part I

Background

Chapter 2

Machine Learning

The goal of this chapter is not to give a complete overview of machine learning, but rather to provide a sufficient understanding of the notions relevant for this thesis.

Machine learning is first presented generally (Section 2.1). Then, different categories of machine learning algorithms are discussed in order to define the scope of this thesis (Section 2.2). Next I discuss the desirable properties of these algorithms, how they can be compared against each other and how their respective usefulness can be evaluated (Section 2.3).

Afterwards, the error or the difference between the output of an algorithm and the true solution is discussed. In particular, the bias and the variance, two components of this error, are defined and discussed in Section 2.4. Mixtures, the aggregation of different models, are then presented and motivated based on the bias and variance trade-off (Section 2.6).

Finally, the content of this thesis is positioned with respect to machine learning, and validation procedures for the developed algorithms are discussed.

2.1 General Machine Learning Framework

Machine learning is the scientific discipline concerned with the development and the study of algorithms to process a learning set D, generally obtained by the observation of a phenomenon, in order to derive and/or exploit a useful model of the phenomenon (or its part of interest). This field is also called statistical learning or automatic learning.

In this section, the notions of 'learning set' and 'useful model' are formally described.

2.1.1 Learning Set

In this context, a learning set is a collection of observations. Each observation results from the monitoring of the problem studied, usually at a very low-level, without refined processing.

More formally, the phenomenon is observed through a set of p *random variables* taken in some order:

$$\boldsymbol{\mathcal{X}} = \{\mathcal{X}_1, \ldots, \mathcal{X}_p\}\,, \tag{2.1}$$

whose joint probability density is denoted by $\mathbb{P}(\boldsymbol{\mathcal{X}})$. Each of those variables can be either categorical or numerical, continuous or discrete.

An *observation* or *sample* $\mathbf{x}$ corresponds to a tuple containing the values of those variables when the sample is acquired and the variables measured:

$$Val(\boldsymbol{\mathcal{X}}) \ni \mathbf{x}_D = \mathbf{x} = (\mathcal{X}_1 = x_1, \ldots, \mathcal{X}_p = x_p) \sim \mathbb{P}(\boldsymbol{\mathcal{X}}) \ . \tag{2.2}$$

$Val(\boldsymbol{\mathcal{X}})$ is the set of all values $\boldsymbol{\mathcal{X}}$ can take. I will also use $\boldsymbol{\mathcal{X}}$ to denote this set when no confusion is possible. The values of some variables can be unknown. In that case, the sample is said to contain missing values. However, this case will not be considered in this thesis.

A *data set*, *observation set* or *sample set* D is a sequence of N samples:

$$D = \mathbf{x}_{D_1}, \ldots, \mathbf{x}_{D_N} \ . \tag{2.3}$$

A core hypothesis made in this thesis is that these observations are all generated by the same density and that this density does not change in time. Moreover, these samples are considered to be independent from each other.

In that case, these observations are denoted *iid*, for independent and identically distributed, and the probability of a data set D is

$$\mathbb{P}(D) = \prod_{i=1}^{N} \mathbb{P}(\boldsymbol{\mathcal{X}} = \mathbf{x}_{D_i}) \ . \tag{2.4}$$

In this context, the order of the observations is irrelevant: D is a (multi)set.

2.1.2 Learning Algorithm

A *learning algorithm* takes as input a data set, called in this context the training or *learning set*, and outputs a *model*, capable to answer queries about the problem of interest. This process is illustrated in Figure 2.1. Two important features of a learning algorithm are the search space and the search strategy.

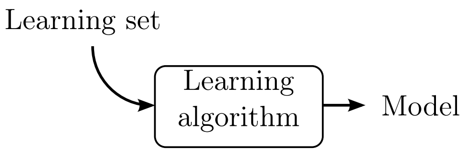


Figure 2.1: A learning algorithm takes as input a learning set and outputs a model.

The *search space* is the set of all the models the algorithm can possibly output.

The *search strategy* describes how the algorithm selects its output in the search space, given a learning set. This process can be deterministic or stochastic. In the first case the algorithm always produces the same model when provided with a given learning set. In the second case, the model varies from one application of the algorithm to another on an identical learning set. Possible strategies include: an exhaustive enumeration of all possible models and the selection of the best one; a stochastic walk in the search space, either random, based on the quality or on another characteristic of the models, and the output of the best model encountered; a gradient descent from an initial point to optimize a function of the model and of the learning set, so as to reach a model corresponding to a local minimum of this function; the elimination of models not satisfying constraints derived from the learning set until a single model remain etc.

The model can be either informative, predictive or both. Informative models provide information about the problem, e.g. which variables are influencing each other, whether different subgroups of observations can be identified etc. Predictive models on the other hand provide information about new realizations of the problem. A typical example is the following question: based on the values of only a subset of the variables (e.g. a set of products previously bought by an individual) in a new sample of the target density of probability, what is the most likely configuration of another, unobserved variable (e.g. the sale of another product to this individual)? Some additional examples will be given in Section 2.2.

2.2 Subdivisions of Machine Learning Problems

Machine learning algorithms and models can be used to solve or contribute to the answer of a wide range of questions. Algorithms and models are generally specialized for only a few of these tasks.

In this section, a few machine learning problems are presented. Each class of problems is illustrated by an example and a simple algorithm. The first division is between supervised and unsupervised learning problems, the latter set being the one (mostly) considered in this thesis.

2.2.1 Supervised Learning

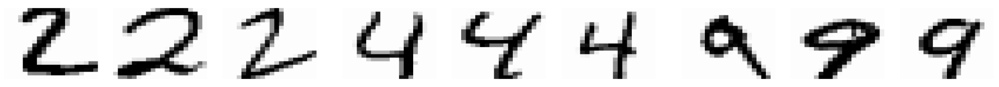
In supervised learning, the set of variables is composed of two separate user-defined sets $\boldsymbol{\mathcal{X}}$, the features or inputs, and $\boldsymbol{\mathcal{C}}$ the output, label or class variables. This distinction is made because only some variables, the outputs, are interesting in the problem context. Typically, the goal of the machine learning procedure is to predict the values of the outputs based on the values of the inputs (predictive setting), or to identify the input variables influencing the outputs (informative setting). The resulting model is a function

$$M_D : Val(\boldsymbol{\mathcal{X}}) \to Val(\boldsymbol{\mathcal{C}}) \ , \tag{2.5}$$

where $Val(\boldsymbol{\mathcal{X}})$ denotes the set of all possible configurations of $\boldsymbol{\mathcal{X}}$.

In the classic setting, the output comprises only one variable. In a predictive setting, a further distinction is usually made based on the type of the output variable. When the variable is categorical, the supervised learning problem is called **classification**. When its type is numerical, the problem is called **regression**.

Consider as a first example the problem of handwritten digit recognition. The problem is to identify the correct digit based on an image composed of a certain number of pixels. A learning set for this problem is typically



(a) 2 (b) 4 (c) 9

Figure 2.2: The problem of handwritten digit recognition consists in correctly associating a digit handwritten representation to the correct digit in [0-9]. These examples are composed of 20x20 pixels, and are taken from the MNIST data base [LBBH98].

composed of different such images (see Figure 2.2), each associated to the correct digit. In that case, the features are the different pixels intensities and the label of the image is the output. Since it is categorical, this is a classification problem.

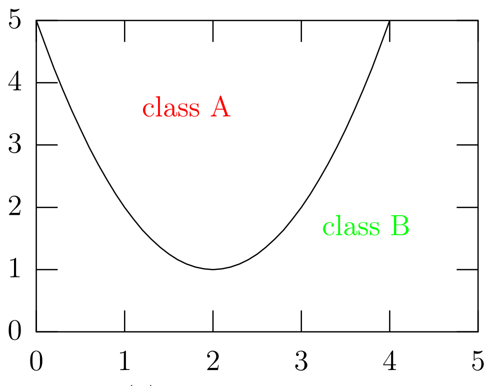
A simple algorithm providing an answer to this problem is the *Nearest Neighbour* algorithm [CH67]. The model corresponds to the learning set, hence there is no real learning. A new problem instance is classified by computing the distance of this instance to each learning observation, and associating to the new input the label of the closest learning observation. Hence the algorithm implicitly computes a decision boundary, as illustrated in Figure 2.3 for a binary target variable. This algorithm can be extended to associate to a new instance the label most frequently associated to the k nearest neighbors of the observation.

Regression is illustrated in Figure 2.4. In this example, a polynomial $\hat{\mathcal{C}} = \hat{f}(\mathcal{X})$ of degree 7 is learned by minimizing the mean square error to a learning set $D = \left\{(x_{D_j}, c_{D_j})\right\}_{j=1}^{12}$ containing 12 observations. These observations are noisy samples of the original model, a polynomial $f(\mathcal{X})$ of degree 3:

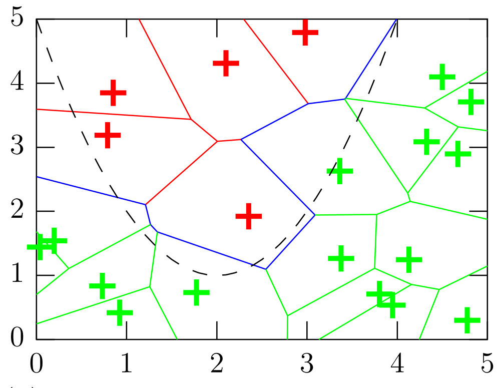
$$\mathcal{C} \sim f(\mathcal{X}) + \mathcal{N}(0, 0.5) \ , \tag{2.6}$$

where $\mathcal{N}(0, 0.5)$ is a normal density of mean 0 and standard deviation 0.5. The mean square distance to minimize is here

$$\sum_{j=1}^{12} |\hat{f}(x_{D_j}) - c_{D_j}|^2 \ . \tag{2.7}$$



(a) Original model



(b) Learning observations are marked by crosses. A new sample "above" the blue continuous line will be labeled as class A.

Figure 2.3: The nearest neighbor algorithm assigns to any new observation the class of the learning observation that is the closest to that new observation. The space partition induced by this method is therefore based on Voronoi cells. In this example, there are two features, and the output is a binary class.

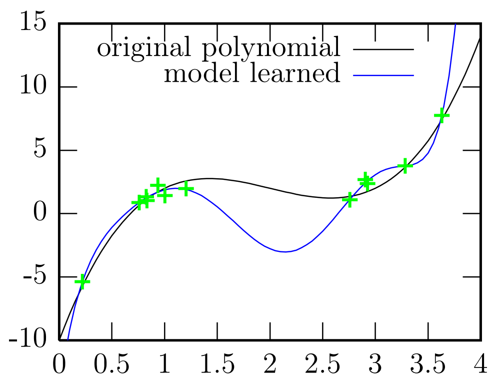


Figure 2.4: In this regression problem, a polynomial of degree 7 is learned by minimizing its mean square error to 12 observations.

This problem was already presented in the introduction of the thesis.

2.2.2 Unsupervised Learning

In unsupervised learning, all variables have a priori the same status. The focus is on the global structure of the problem rather than only on the prediction of a selected subset of variables based on the values taken by another selected subset, as in supervised learning. It is however possible to exploit an unsupervised model to solve a supervised problem: from a model specifying the interactions between all variables, it is e.g. possible to obtain the variables influencing a specific output variables. However this specialization of the model can be done when needed, meaning it could be adapted to a changing problem. This latter approach to a prediction task is referred to as *generative learning*, since it could be used to generate observations similar to the input. *Discriminative learning*, on the other hand, refers to the direct modelling of the input $\rightarrow$ output mapping.

Many different unsupervised learning problems exist. In this work the goal is the modeling of a probability density $\mathbb{P}(\boldsymbol{\mathcal{X}})$ for inference. Nevertheless, a few different unsupervised learning problems are described next.

Clustering

Clustering or cluster analysis targets the identification of different subgroups in the learning set. Their number might be fixed by the user, or automatically adjusted by the algorithm. A cluster can be a group of observations or of variables, and the goal of the learning procedure is to associate each observation or each variable to a group, so that two elements belonging to the same group are more similar than two elements belonging to different groups.

The key notion here is the definition of a measure of similarity between subsets of samples or of variables. Figure 2.5 contains an example of hierarchical clustering applied to observations of a single numerical variable. The variant illustrated here builds a hierarchy of clusters. It starts with one cluster per observation, and iteratively merges the two closest clusters.

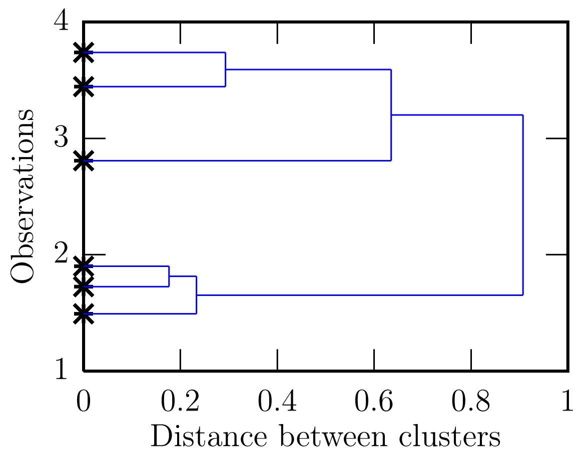


Figure 2.5: 6 observations (black star) of a single variable are analyzed by hierarchical clustering. The distance between two clusters is the minimum distance between two elements belonging each to one cluster. Each blue vertical line represents a merge between two clusters.

Density Estimation and Probabilistic Inference

Density estimation was originally studied in statistics. Its aim is to obtain from a set D of observations an estimate $M_D = \widehat{\mathbb{P}}(\boldsymbol{\mathcal{X}})$ of the probability density that generated the samples.

Such an estimate can be used to answer various queries about the original probability density, a process called probabilistic inference, e.g. to compute

- a conditional density $\widehat{\mathbb{P}}(\boldsymbol{\mathcal{C}}|\boldsymbol{\mathcal{X}}')$ and to use it for predicting the values of the output $\boldsymbol{\mathcal{C}}$ based on some inputs $\boldsymbol{\mathcal{X}}'$;
- the most likely configuration $\mathbf{x}_{\text{ML}} = \arg\max_{\mathbf{x}} \widehat{\mathbb{P}}(\mathbf{x})$;
- the likelihood of observing a problem instance $\mathbf{x}$.

Density estimation is widely used as a subroutine to tackle machine learning problems, and therefore it has been extensively studied in machine learning. Note that further example of the possibilities of probabilistic inference will be developed in Section 3.4.

In addition to using it, machine learning can also play a role in performing density estimation. As will be discussed in Chapter 3, machine learning can infer the structure of a density from a learning set, and exploiting this structure, the relationships between variables, can radically reduce the number of parameters quantifying the probability density, hence making the resulting density more tractable (in terms of storage), and more accurate.

Obtaining a density is also useful to generate new observations, e.g. in the context of simulations.

2.3 Validation of Machine Learning Approaches

An important aspect of machine learning is to estimate the quality of a solution, to assess the confidence one can have in the information provided. Two validation problems can be distinguished: the validation of an algorithm in general and the validation of a model for a given problem. They are discussed in turn below.

Other important features such as interpretability of the resulting model and algorithmic complexity are not discussed here.

2.3.1 Validation of an Algorithm

Section 2.3.2 will discuss how a model provided by an algorithm can be evaluated on a given problem. Suppose such a measure is available. A machine learning scientist may also be concerned by the evaluation of an algorithm, especially a new one, against other algorithms or for a generic purpose.

There are two means to evaluate an algorithm with respect to others.

The first is theoretical analysis. Bounds can be provided on the accuracy or the run time of an algorithm, either on a finite set of observations or asymptotically, as the number of observations grows to infinity.

The second is empirical analysis. The algorithms are applied on a selection of representative problems and the results are evaluated. To reduce the noise and the variance in the analysis, the more problems the better. Several problems and learning sets are therefore considered. Moreover, to limit the variance of the measurements, it is better to use the same learning sets for all algorithms, and to use the same test sets as well.

2.3.2 Validation of a Model

The second problem considered is the validation of a given model on a particular problem instance. A common practice to build an application is to select a set of different algorithms suited to the task considered, to apply them all on the learning set and to compare the corresponding models against each other. The best model is then chosen. This section discusses how to quantify the quality of a model.

Only models suited for classification, regression and density estimation problems will be discussed here. These are chosen either because they are related to this thesis (density estimation) or because they facilitate the explanation of some concepts. Note that evaluating the result of some other machine learning algorithms, such as clustering or informative models, can indeed be much more difficult and subjective. I will also limit the discussion to a finite and discrete variable space, which makes both notations and discussions easier. The elements presented here can however be carried over to continuous variables.

The quality of a predictive model M_D is often evaluated in terms of its error. The true error of such a model is the average error on all possible configurations of the variables:

$$\mathrm{Err}(M_D) = \sum_{\mathbf{x}} \mathrm{Err}(M_D, \mathbf{x})\mathbb{P}(\mathbf{x}) \ , \tag{2.8}$$

where $\mathrm{Err}(M_D, \mathbf{x})$ is the error of the model for a given variable configuration $\mathbf{x}$. This quantity must be chosen in accordance with the problem considered.

For example, a standard choice for classification problem is the loss function

$$L(M_D, \mathbf{x}, \mathbf{c}) = \begin{cases} 0 & \text{if } M_D(\mathbf{x}) = \mathbf{c}, \\ 1 & \text{otherwise,} \end{cases} \tag{2.9}$$

where, in this case, $\mathbf{x}$ and $\mathbf{c}$ are respectively the values of the input or output variables. The error is null if the output is correctly predicted, and the model is penalized for an incorrect prediction.

For a regression problem, the mean square error is often used:

$$\mathrm{MSE}(M_D, \mathbf{x}, \mathbf{c}) = (M_D(\mathbf{x}) - \mathbf{c})^2 \ . \tag{2.10}$$

In density estimation problems, the focus of this thesis, a suitable error is the likelihood ratio of a given variable configuration $\mathbf{x}$ according to the true density and the one encoded by the model. Hence the inadequacy between the two densities is measured by the well-known Kullback-Leibler (KL) divergence [KL51], also called information divergence or relative entropy:

$$\begin{aligned} \mathrm{Err}(M_D) &= D_{KL}(\mathbb{P}||M_D) & (2.11) \\ &= \sum_{\mathbf{x}} \mathbb{P}(\mathbf{x}) \log \frac{\mathbb{P}(\mathbf{x})}{M_D(\mathbf{x})} \ . & (2.12) \end{aligned}$$

This divergence can be interpreted as the expected number of extra bits required to encode samples from the original density by a code based on M_D, rather than one based on $\mathbb{P}$. When the two densities are equal, the divergence is null, otherwise it is positive.

The negative log-likelihood according to the model is also used to quantify an error:

$$\mathrm{nlogll}(M_D, \mathbf{x}) = -\log M_D(\mathbf{x}) \ , \tag{2.13}$$

making the model error the cross entropy:

$$H(\mathbb{P}, M_D) = \sum_{\mathbf{x}} -\mathbb{P}(\mathbf{x}) \log M_D(\mathbf{x}) \ . \tag{2.14}$$

The relationship between the two quantities is as follow:

$$\begin{aligned} D_{KL}(\mathbb{P}||M_D) &= \sum_{\mathbf{x}} \mathbb{P}(\mathbf{x}) \log \frac{\mathbb{P}(\mathbf{x})}{M_D(\mathbf{x})} & (2.15) \\ &= \sum_{\mathbf{x}} \mathbb{P}(\mathbf{x}) \left[\log \mathbb{P}(\mathbf{x}) - \log M_D(\mathbf{x})\right] & (2.16) \\ &= -H(\boldsymbol{\mathcal{X}}) + H(\mathbb{P}, M_D) \ . & (2.17) \end{aligned}$$

The difference between the two, the entropy $H(\boldsymbol{\mathcal{X}})$, is constant for a given problem. Therefore the difference between the respective scores of two models on a given problem would be identical if the KL divergence or the negative log-likelihood are used.

However, in practice, those quantities can usually not be computed exactly. The first reason is that machine learning is applied precisely on problems where the true density $\mathbb{P}(\boldsymbol{\mathcal{X}})$ is unknown. The second reason is that the variable set is usually large, so that computing the sum over all configurations $\mathbf{x}$ might be infeasible in a reasonable time.

Therefore those measures are estimated based on a set D' of N' observations, called the *test set*, using a Monte-Carlo like procedure:

$$\widehat{\mathrm{Err}}_{D'}(M_D) = \frac{1}{N'}\sum_{i=1}^{N'} \mathrm{Err}(M_D, \mathbf{x}_{D'_i}) \ . \tag{2.18}$$

The error used to quantify the accuracy of a model is not necessarily directly related to the score explicitly or implicitly maximized by the learning algorithm, as presented in Section 2.1.2. However both are related to the quality of a model in terms of accuracy, and typically, when the score of a model increases, its error decreases on the learning set.

Overfitting and Cross-validation

If the error of a model is evaluated on the same observations that were used to learn it, this error is likely to be underestimated. Indeed, this does not penalize the model for being overly complex and modelling sampling noise, which may lead to bad results on unseen configurations of the variables. This is particularly important, because the model will be applied on new, unseen samples, and the essence of a good machine learning model is that it generalizes well.

The *complexity of a model* refers to its adaptivity, its capacity to fit the learning set, its number of degrees of freedom. The way this complexity is quantified depends on the model type. For the few models described above, complexity can be defined as follows.

- For the k nearest neighbor algorithm, complexity increases when fewer neighbors are considered.
- For regression by least-square fitting of a polynomial, complexity increases with the degree of the polynomial.
- For clustering, complexity increases with the number of clusters.
- For density estimation, complexity increases with the number of independent parameters used to represent a model.

When the complexity of a model increases, it can generally fit the learning set increasingly better, and the error measured on the learning set decreases. However this is not always beneficial, because the error on unseen

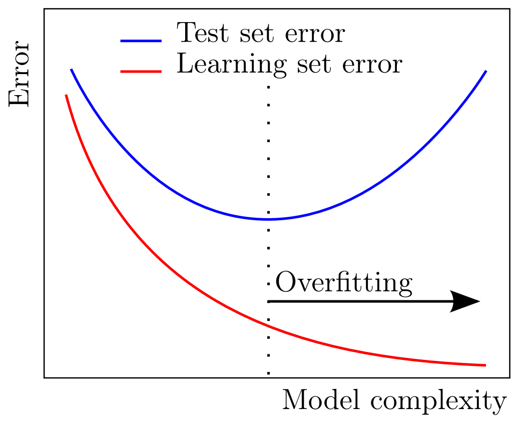


Figure 2.6: The concept of overfitting is illustrated. Usually, the error of a model is a decreasing function of model complexity when measured on the learning set. However when it is measured on a test set independent from the learning set, it can start increasing when complexity grows too large.

samples can increase and more than counterbalance the gain in accuracy on the learning set. This is called overfitting, and is illustrated in Figure 2.6. The figure represents a typical evolution of two measures of the error of a model of increasing complexity. These measures are either based on the learning set or of an independent set of observations, the test set. While the error measured on the learning set is a decreasing function of complexity, the test set error first decreases but starts increasing when complexity reaches a threshold.

A method to detect overfitting follows naturally: the observations used to evaluate the quality of the model must be different from the ones used for learning. This can also be used to select the complexity of the model. Note that overfitting is directly related to the bias-variance trade-off, which will be discussed in Section 2.4.

A machine learning practitioner is usually not provided with a learning set and a test set but only with a set of observations. *Cross-validation* denotes the process of splitting that original set to obtain distinct learning and validation sets. However, a measure obtained from two such sets is dependant on the split, and vary from one split to another. In order to reduce the variance of that estimated measure of performance, different values can be computed from several different splits and averaged, leading to a more robust estimation.

K-fold cross-validation is a procedure where the original sample set is partitioned into K subsets. Each of these subsets is used as test set for evaluating the model learned on the $K-1$ remaining subsets. Leave-one-out cross-validation denotes the special case where $K = N$.

2.4 Bias-variance Compromise

Both in the introduction and in the previous section the necessity to select a model of an appropriate complexity was mentioned. In the present section this is discussed in more length, and the important notions of bias and variance are introduced in the process. These concepts are first derived for regression problems, and then extended to the context of density estimation.

For any learning set, it is possible to select a class of models sufficiently complex so that at least one of them can match the observations perfectly, a model producing a minimal error. This error may not be zero, e.g. in a classification problem where the same configuration of the input variables is associated to two different configurations of the output variables because of noise.

However, perfectly matching the learning set is not necessarily desirable, even without taking algorithm complexity into account. Indeed, the model should not only be good on that learning set, but also on new, unseen problem instances. This is called the *generalisation* of the model. As mentioned above, overfitting happens when a model is too specialized on the learning set (see Figure 2.6).

In order to express these notions mathematically, the error resulting from a model learning on a random sample set can be studied. This error is a random variable, function of the learning set that has been obtained/generated, as described in Section 2.1.1.

2.4.1 Bias and Variance in Regression Problems

The error

$$\mathrm{Err}(M_D) = \mathbb{E}_{\boldsymbol{\mathcal{X}},\boldsymbol{\mathcal{C}}} \left(M_D(\mathbf{x}) - \mathbf{c}\right)^2 \tag{2.19}$$

of the model M_D learned from a given sample set D is a random variable that depends on that sample set. The expected value of that error (with respect to the different learning sets) can be decomposed as:

$$\mathbb{E}_D \mathrm{Err}(M_D) = \mathbb{E}_D \mathbb{E}_{\boldsymbol{\mathcal{X}},\boldsymbol{\mathcal{C}}} \left(M_D(\mathbf{x}) - \mathbf{c}\right)^2 \tag{2.20}$$

$$= \mathbb{E}_{\boldsymbol{\mathcal{X}}} \mathbb{E}_D \mathbb{E}_{\boldsymbol{\mathcal{C}}|\mathbf{x}} \left(M_D(\mathbf{x}) - \mathbf{c}\right)^2 \ . \tag{2.21}$$

$\mathbb{E}_D \mathbb{E}_{\boldsymbol{\mathcal{C}}|\mathbf{x}} \left(M_D(\mathbf{x}) - \mathbf{c}\right)^2$ is the average error made when predicting a value for a given input, on all sample sets and for all possible outputs. This expression can be simplified by introducing the Bayes model M_b that associates to any input $\mathbf{x}$ the expectation of the output $\mathbb{E}_{\boldsymbol{\mathcal{C}}|\mathbf{x}}\mathbf{c}$. M_b is known to be optimal with respect to the square error.

Therefore,

$$\mathbb{E}_D \mathbb{E}_{\boldsymbol{\mathcal{C}}|\mathbf{x}} \left(M_D(\mathbf{x}) - \mathbf{c}\right)^2 = \mathbb{E}_D \mathbb{E}_{\boldsymbol{\mathcal{C}}|\mathbf{x}} \left(M_D(\mathbf{x}) - M_b(\mathbf{x}) + M_b(\mathbf{x}) - \mathbf{c}\right)^2 \tag{2.22}$$

$$\begin{aligned} &= \mathbb{E}_D \mathbb{E}_{\boldsymbol{\mathcal{C}}|\mathbf{x}} \left(M_D(\mathbf{x}) - M_b(\mathbf{x})\right)^2 + \mathbb{E}_D \mathbb{E}_{\boldsymbol{\mathcal{C}}|\mathbf{x}} \left(M_b(\mathbf{x}) - \mathbf{c}\right)^2 \\ &\quad + \mathbb{E}_D \mathbb{E}_{\boldsymbol{\mathcal{C}}|\mathbf{x}} \left[2(M_D(\mathbf{x}) - M_b(\mathbf{x}))(M_b(\mathbf{x}) - \mathbf{c})\right] \end{aligned} \tag{2.23}$$

$$\begin{aligned} &= \mathbb{E}_D \left(M_D(\mathbf{x}) - M_b(\mathbf{x})\right)^2 + \mathbb{E}_{\boldsymbol{\mathcal{C}}|\mathbf{x}} \left(M_b(\mathbf{x}) - \mathbf{c}\right)^2 \\ &\quad + 2\mathbb{E}_D \left[(M_D(\mathbf{x}) - M_b(\mathbf{x}))\mathbb{E}_{\boldsymbol{\mathcal{C}}|\mathbf{x}}(M_b(\mathbf{x}) - \mathbf{c})\right] \end{aligned} \tag{2.24}$$

where that last term is equal to zero, since $M_b(\mathbf{x}) = \mathbb{E}_{\boldsymbol{\mathcal{C}}|\mathbf{x}}\mathbf{c}$.

Moreover, the first term can be further developed by the introduction of the expected predicted value over the different models, averaged over all possible learning sets $\bar{M}(\mathbf{x}) \triangleq \mathbb{E}_D(M_D(\mathbf{x}))$:

$$\mathbb{E}_D \left(M_D(\mathbf{x}) - M_b(\mathbf{x})\right)^2 = \mathbb{E}_D \left(M_D(\mathbf{x}) - \bar{M}(\mathbf{x}) + \bar{M}(\mathbf{x}) - M_b(\mathbf{x})\right)^2 \tag{2.25}$$

$$\begin{aligned} &= \mathbb{E}_D \left(M_D(\mathbf{x}) - \bar{M}(\mathbf{x})\right)^2 + \mathbb{E}_D \left(\bar{M}(\mathbf{x}) - M_b(\mathbf{x})\right)^2 \\ &\quad + \mathbb{E}_D \left[2(M_D(\mathbf{x}) - \bar{M}(\mathbf{x}))(\bar{M}(\mathbf{x}) - M_b(\mathbf{x}))\right] \end{aligned} \tag{2.26}$$

$$\begin{aligned} &= \mathbb{E}_D \left(M_D(\mathbf{x}) - \bar{M}(\mathbf{x})\right)^2 + \left(\bar{M}(\mathbf{x}) - M_b(\mathbf{x})\right)^2 \\ &\quad + 2(\bar{M}(\mathbf{x}) - M_b(\mathbf{x}))\mathbb{E}_D \left(M_D(\mathbf{x}) - \bar{M}(\mathbf{x})\right) . \end{aligned} \tag{2.27}$$

Once again, the last term vanishes.

Therefore, the expected value of the error for a given input $\mathbf{x}$ is a sum of three terms:

$$\mathbb{E}_D \mathbb{E}_{\boldsymbol{\mathcal{C}}|\mathbf{x}} \left(M_D(\mathbf{x}) - \mathbf{c}\right)^2 = \sigma^2(\mathbf{x}) + \mathrm{bias}^2(\mathbf{x}) + \mathrm{var}(\mathbf{x}) \ , \tag{2.28}$$

with

$$\sigma^2(\mathbf{x}) = \mathbb{E}_{\mathcal{C}|\mathbf{x}} \left(M_b(\mathbf{x}) - \mathbf{c}\right)^2 \tag{2.29}$$

$$\text{bias}^2(\mathbf{x}) = \left(\bar{M}(\mathbf{x}) - M_b(\mathbf{x})\right)^2 \tag{2.30}$$

$$\text{var}(\mathbf{x}) = \mathbb{E}_D \left(M_D(\mathbf{x}) - \bar{M}(\mathbf{x})\right)^2 \quad . \tag{2.31}$$

$\sigma^2(\mathbf{x})$ is called the residual error, and is the lower bound of the error of any model. It is caused by a residual uncertainty in the value of the output even when both the inputs and the relationships between input and output are known.

The algorithm and the associated class of models selected influence the other two components of the error.

The term bias$^2(\mathbf{x})$ is the mismatch between the average model obtained from a learning set and the optimal Bayes model. This mismatch is due to a combination of a search space too simple to contain the true model (model bias) and an incapacity of the search strategy to identify in this space the model best matching the observations (estimation bias).

The variance var$(\mathbf{x})$ originates from the variability in the learning set. As the number of samples increases, it usually shrinks. It is related to overfitting.

These errors are schematically depicted in Figure 2.7. The true model is outside the search space, and the search strategy only covers part of this space, leading respectively to a model and estimation bias. Moreover, the learning set used as input for the learning algorithm can vary, which is a source of variance.

This decomposition is also illustrated in Figure 2.8 on the regression problem presented in Section 2.2.1. 1000 sets of 12 noisy observations of a polynomial are generated, and each set is used to learn polynomials of increased degree. The variance, bias and error of these models are displayed as a function of this degree. The error is minimal when this degree is identical to the degree (=3) of the target. The bias and the variance are respectively the main component of the error when the degree of the estimations is respectively smaller and higher than 3. $\sigma^2(\mathbf{x})$ is constant.

So far, the learning algorithm has been considered deterministic. The analysis performed above can be extended to include stochastic algorithms.

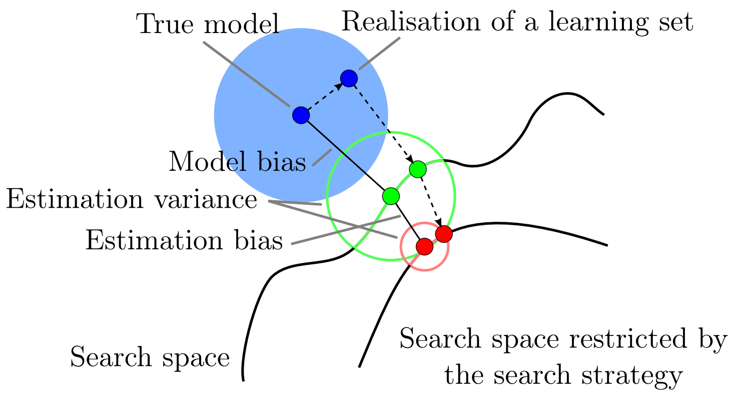
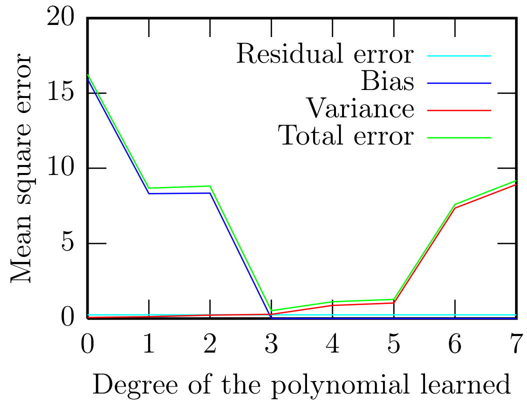
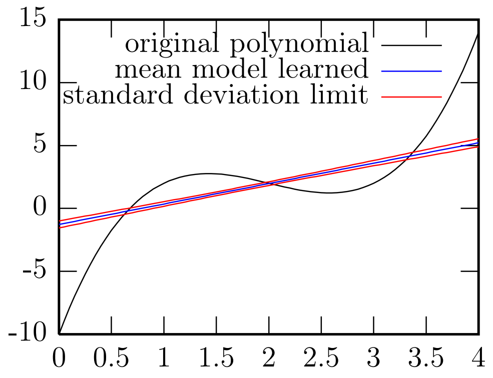

True model
Realisation of a learning set
Model bias
Estimation variance
Estimation bias
Search space
Search space restricted by the search strategy

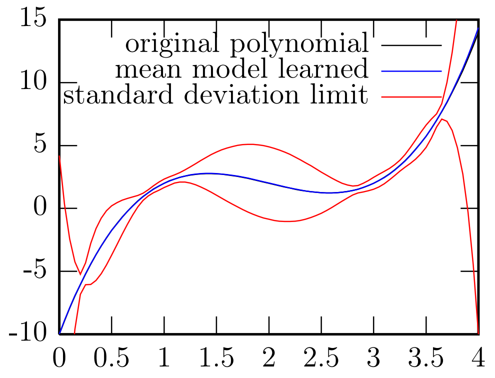
Figure 2.7: This schematic representation illustrates the concept of bias and variance. It was greatly inspired by figure 7.2 from [HTF01].
If the true model was known, the best possible choice among the search space would be the green dot linked to the true model. However the search strategy may not consider all models in the search space, or may on average output another model and so the algorithm would output the model corresponding to the red dot linked to the green one mentioned above. The model bias corresponds to the difference between the true and the best model in the search space, the estimation bias to the difference between this best model and the average estimate output by the algorithm. Together, they constitute the bias.
In practice however, information about the true model is available only through a learning set, a noisy picture of the true model. This noise is represented by the blue disk. There is one model in the search space (second green dot) and in the restricted space (second red dot) that best correspond to this noisy picture. Since the learning set vary, this leads to a second source of error, the variance (green and red circles). By selecting an appropriate search strategy, the estimation bias may be smaller than the reduction in variance, leading to a smaller average error.



(a) Evolution of the error with the degree of the estimated model.



(b) The degree of the polynomial is 1. The difference between the average model and the target (the bias) is large.)



(c) The degree of the polynomial is 7. The variability of the model learned (the variance) is large.

Figure 2.8: The bias-variance decomposition is illustrated on a regression problem. The true model is a polynomial of degree 3 and a sampling noise. A learning set contains 12 observations, with fixed abscissae. 1000 independent learning sets are generated, and these learning sets are used in a Monte-Carlo approximation of $\mathbb{E}_D$ in the error, the variance and the bias, computed for different degrees of the estimated polynomials. These quantities are measured on the smallest x interval containing the observations. In this problem, $M_b = f$ and $\sigma^2(\mathbf{x}) = 0.5^2$.

One possibility to do so is to add in the variable set a variable $\mathcal{S}$ that accounts for the stochasticity of the algorithm.

2.4.2 Density Estimation

In statistical estimation, different errors can be considered. The first kind of error is at the parameter level. For a given model, how well are its parameters estimated? The second kind of error is based on the comparison of the densities directly, whatever the evaluated model is.

Both are relevant in the context of Bayesian network learning. The first one is used for studying parameter learning for a given probabilistic model structure, and the second is at the core of this thesis.

Parameter Estimation

One possibility to compare two models, one estimated and one original model, is to compare the values of a common parameter between these two models.

In the case where the two models are similar and for a parameter whose true value is denoted by θ, the mean square error of an estimate $\widehat{\theta}_D$ is composed of the bias and variance:

$$\operatorname{bias}(\widehat{\theta}_D) = \mathbb{E}_D(\widehat{\theta}_D) - \theta, \tag{2.32}$$

$$\operatorname{var}(\widehat{\theta}_D) = \mathbb{E}_D\left(\widehat{\theta}_D - \mathbb{E}_{D'}\widehat{\theta}_{D'}\right)^2 \quad . \tag{2.33}$$

Consider as an example the problem of estimating the mean μ of a univariate normal density $\mathbb{P}(\mathcal{X}|\mu) = \mathcal{N}(\mu, \sigma)$, with $\sigma > 0$ the standard deviation of the density, supposed to be known. Two estimators are considered, first the maximum likelihood (ML) estimator and then a maximum a posteriori (MAP) estimator.

A usual estimator for μ is the maximum likelihood estimator, which corresponds to the mean of the observations $\{x_{D_j}\}_{j=1}^N$:

$$\widehat{\mu}_D^{ML} = \arg\max_{\hat{\mu}} \sum_{j=1}^{N} \log \mathbb{P}(x_{D_j}|\hat{\mu}) \tag{2.34}$$

$$= \frac{1}{N}\sum_{j=1}^{N} x_{D_j} \quad . \tag{2.35}$$

This estimator of the mean μ is unbiased since

$$\mathbb{E}_D \widehat{\mu}_D^{ML} = \mathbb{E}_D \frac{1}{N} \sum_{j=1}^{N} x_{D_j} \tag{2.36}$$

$$= \frac{1}{N} \sum_{j=1}^{N} \mathbb{E}_D x_{D_j} \tag{2.37}$$

$$= \frac{1}{N} \sum_{j=1}^{N} \mu \ , \tag{2.38}$$

and its variance is equal to

$$\mathrm{var}(\widehat{\mu}_D^{ML}) = \mathbb{E}_D \left(\widehat{\mu}_D^{ML} - \mathbb{E}_{D'} \widehat{\mu}_{D'}^{ML}\right)^2 \tag{2.39}$$

$$= \mathbb{E}_D \left(\widehat{\mu}_D^{ML} - \mu\right)^2 \tag{2.40}$$

$$= \mathbb{E}_D (\widehat{\mu}_D^{ML})^2 - 2 * \mu \mathbb{E}_D \widehat{\mu}_D^{ML} + \mu^2 \tag{2.41}$$

$$= \frac{1}{N^2} \mathbb{E}_D \left(\sum_{j=1}^{N} x_{D_j} \right)^2 - \mu^2 \tag{2.42}$$

$$= \frac{1}{N^2} \mathbb{E}_D \left(\sum_{j=1}^{N} (x_{D_j} - \mu) \right)^2 \tag{2.43}$$

$$= \frac{\sigma^2}{N} \ . \tag{2.44}$$

Another estimator for the mean μ is the maximum a posteriori estimator. A prior density over μ is specified as $\mathbb{P}_0(\mu) = \mathcal{N}(\mu_0, \sigma_0)$. The MAP estimate for μ is the value

$$\hat{\mu}_D^{MAP} = \arg\max_{\hat{\mu}} \mathbb{P}_0(\hat{\mu}) \prod_{j=1}^{N} \mathbb{P}(x_{D_j} | \hat{\mu}) \tag{2.45}$$

$$= \frac{\frac{\sigma^2}{N} \mu_0 + \sigma_0^2 \hat{\mu}_D^{ML}}{\frac{\sigma^2}{N} + \sigma_0^2} \ . \tag{2.46}$$

The bias of this estimator is not zero if $\mu_0 \neq \mu$:

$$\mathbb{E}_D \widehat{\mu}_D^{MAP} = \frac{\frac{\sigma^2}{N}\mu_0 + \sigma_0^2 \mathbb{E}_D \hat{\mu}_D^{ML}}{\frac{\sigma^2}{N} + \sigma_0^2} \tag{2.47}$$

$$= \frac{\frac{\sigma^2}{N}\mu_0 + \sigma_0^2 \mu}{\frac{\sigma^2}{N} + \sigma_0^2} \tag{2.48}$$

$$\text{bias}(\widehat{\mu}_D^{MAP}) = \mathbb{E}_D \widehat{\mu}_D^{MAP} - \mu \tag{2.49}$$

$$= \frac{\frac{\sigma^2}{N}(\mu_0 - \mu)}{\frac{\sigma^2}{N} + \sigma_0^2} \ . \tag{2.50}$$

The variance of the MAP estimator is:

$$\text{var}(\widehat{\mu}_D^{MAP}) = \mathbb{E}_D \left(\widehat{\mu}_D^{MAP} - \mathbb{E}_{D'} \widehat{\mu}_{D'}^{MAP}\right)^2 \tag{2.51}$$

$$= \mathbb{E}_D \left(\frac{\frac{\sigma^2}{N}\mu_0 + \sigma_0^2 \hat{\mu}_D^{ML}}{\frac{\sigma^2}{N} + \sigma_0^2} - \frac{\frac{\sigma^2}{N}\mu_0 + \sigma_0^2 \mu}{\frac{\sigma^2}{N} + \sigma_0^2} \right)^2 \tag{2.52}$$

$$= \left(\frac{1}{\frac{\sigma^2}{N} + \sigma_0^2} \right)^2 \mathbb{E}_D \left(\sigma_0^2 \hat{\mu}_D^{ML} - \sigma_0^2 \mu\right)^2 \tag{2.53}$$

$$= \left(\frac{\sigma_0^2}{\frac{\sigma^2}{N} + \sigma_0^2} \right)^2 \mathbb{E}_D \left(\hat{\mu}_D^{ML} - \mu\right)^2 \tag{2.54}$$

$$= \left(\frac{\sigma_0^2}{\frac{\sigma^2}{N} + \sigma_0^2} \right)^2 \frac{\sigma^2}{N} \ . \tag{2.55}$$

To summarize, $\text{bias}(\hat{\mu}_D^{MAP}) \geq \text{bias}(\hat{\mu}_D^{ML})$ and $\text{var}(\hat{\mu}_D^{MAP}) < \text{var}(\hat{\mu}_D^{ML})$. The MAP estimation may therefore be more interesting than the ML one. Even though the former is biased (when $\mu_0 \neq \mu$), its lower variance may result in a lower total error than the ML estimator.

Density Estimation

The problem of statistical estimation amounts to a regression problem on the function

$$\mathbb{P} : \boldsymbol{\mathcal{X}} \to [0,1] : \sum_{\mathbf{x} \in \boldsymbol{\mathcal{X}}} \mathbb{P}(\mathbf{x}) = 1 \tag{2.56}$$

when the variables are discrete, and on the probability density function for continuous variables. Hence the mean square error decomposition from Section 2.4.1 is also valid. The function of interest is here not noisy, $M_b = \mathbb{E}_{\boldsymbol{\mathcal{C}}|\mathbf{x}}\mathbf{c} = \mathbb{P}(\mathbf{x})$ and the residual error is null. Thus the mean square error decomposes into bias and variance only:

$$\begin{aligned}&\mathbb{E}_{\boldsymbol{\mathcal{X}}}\mathbb{E}_D \left(M_D(\mathbf{x}) - \mathbb{P}(\mathbf{x})\right)^2 \\ &\qquad = \mathbb{E}_{\boldsymbol{\mathcal{X}}} \left(\bar{M}(\mathbf{x}) - \mathbb{P}(\mathbf{x})\right)^2 + \mathbb{E}_{\boldsymbol{\mathcal{X}}}\mathbb{E}_D \left(M_D(\mathbf{x}) - \bar{M}(\mathbf{x})\right)^2 \ .\end{aligned} \tag{2.57}$$

A more classical choice to evaluate the quality of a density estimate is the Kullback-Leibler divergence (see Section 2.3.2). A similar decomposition can be performed for this divergence, see e.g. [Hal87]. Such a decomposition has also been made in [WS97] in the context of classification, where the quantity decomposed is the log of the conditional probability density of a target variable, given a configuration of the input variables. As in Section 2.3.2, I consider discrete and finite variables for simplicity, but the developments can be extended to continuous variables:

$$\begin{aligned}\mathbb{E}_D D_{KL}(\mathbb{P}||M_D) &= \mathbb{E}_D \sum_{\mathbf{x} \in \boldsymbol{\mathcal{X}}} \mathbb{P}(\mathbf{x}) \log \frac{\mathbb{P}(\mathbf{x})}{M_D(\mathbf{x})} && (2.58)\\ &= \mathbb{E}_D \sum_{\mathbf{x}} \mathbb{P}(\mathbf{x}) \log \frac{\mathbb{P}(\mathbf{x})\bar{M}(\mathbf{x})}{M_D(\mathbf{x})\bar{M}(\mathbf{x})} && (2.59)\\ &= \mathbb{E}_D \sum_{\mathbf{x}} \mathbb{P}(\mathbf{x}) \left[\log \frac{\mathbb{P}(\mathbf{x})}{\bar{M}(\mathbf{x})} + \log \frac{\bar{M}(\mathbf{x})}{M_D(\mathbf{x})}\right] && (2.60)\\ &= D_{KL}(\mathbb{P}||\bar{M}) + \mathbb{E}_D[H(\mathbb{P}, M_D)] - H(\mathbb{P}, \bar{M}) \ . && (2.61)\end{aligned}$$

The first term $D_{KL}(\mathbb{P}||\bar{M})$ can be interpreted as a bias, as the mismatch of the average model with respect to the true model. To conform to the bias plus variance decomposition, $\mathbb{E}_D[H(\mathbb{P}, M_D)] - H(\mathbb{P}, \bar{M})$ is defined as the variance of the model. Moreover, this variance is positive. Indeed, it can be rewritten as

$$\mathbb{E}_D \sum_{\mathbf{x}} \mathbb{P}(\mathbf{x}) \log \frac{\bar{M}(\mathbf{x})}{M_D(\mathbf{x})} = \sum_{\mathbf{x}} \mathbb{P}(\mathbf{x}) \mathbb{E}_D \left[\log \bar{M}(\mathbf{x}) - \log M_D(\mathbf{x})\right] \quad (2.62)$$

$$= \sum_{\mathbf{x}} \mathbb{P}(\mathbf{x}) \left[\log \bar{M}(\mathbf{x}) - \mathbb{E}_D \log M_D(\mathbf{x})\right] \quad (2.63)$$

$$= \sum_{\mathbf{x}} \mathbb{P}(\mathbf{x}) \left[\log \left(\mathbb{E}_D M_D(\mathbf{x})\right) - \mathbb{E}_D \log M_D(\mathbf{x})\right] \quad (2.64)$$

which is positive since since $\log\left(\mathbb{E}_D M_D(\mathbf{x})\right) - \mathbb{E}_D \log M_D(\mathbf{x}) \geq 0$ by Jensen's inequality: for any concave function ϕ,

$$\phi(\mathbb{E}_s s) \geq \mathbb{E}_s \phi(s) \ . \quad (2.65)$$

An alternative decomposition was proposed by [Hes98] and is based on the geometric mean

$$\bar{M}^G(\mathbf{x}) = \frac{1}{Z} \exp(\mathbb{E}_D \log M_D(\mathbf{x})) \ , \quad (2.66)$$

where Z is a normalization constant. This geometric mean is motivated by the desire to minimize the variance of the models in the decomposition, i.e. to find the model M minimizing the mean KL divergence of a learned model to M [Hes98]:

$$\bar{M}^G = \arg \min_{M : \sum M(\mathbf{x}) = 1} \mathbb{E}_D D_{KL}(M || M_D) \ . \quad (2.67)$$

Solving this constrained optimization problem can be done by the method of Lagrange multipliers.

The quantity $\mathbb{E}_D D_{KL}(\bar{M}^G || M_D)$ is therefore defined as the variance of the learning problem. It is related to the normalization constant Z:

$$\begin{aligned}
\log Z &= \log\left[\frac{\exp(\mathbb{E}_D \log M_D(\mathbf{x}))}{\bar{M}^G(\mathbf{x})}\right] \qquad \forall \mathbf{x} : \bar{M}^G(\mathbf{x}) > 0 & (2.68)\\
&= \mathbb{E}_D(\log M_D(\mathbf{x})) - \log \bar{M}^G(\mathbf{x}) & (2.69)\\
&= \mathbb{E}_D\left[\log \frac{M_D(\mathbf{x})}{\bar{M}^G(\mathbf{x})}\right]\left[\sum_{\mathbf{x}} \bar{M}^G(\mathbf{x})\right] & (2.70)\\
&= \mathbb{E}_D\left[\sum_{\mathbf{x}} \bar{M}^G(\mathbf{x}) \log \frac{M_D(\mathbf{x})}{\bar{M}^G(\mathbf{x})}\right] & (2.71)\\
&= \mathbb{E}_D\left[-D_{KL}(\bar{M}^G||M_D)\right] & (2.72)\\
&= -\text{variance} \ . & (2.73)
\end{aligned}$$

The bias is then $D_{KL}(\mathbb{P}||\bar{M}^G)$, the mean KL divergence of a learned model to the true density, minus the variance:

$$\begin{aligned}
\text{bias} &\triangleq \mathbb{E}_D D_{KL}(\mathbb{P}||M_D) - \text{variance} & (2.74)\\
&= \mathbb{E}_D \sum_{\mathbf{x}} \mathbb{P}(\mathbf{x}) \log \frac{\mathbb{P}(\mathbf{x})}{M_D(\mathbf{x})} + \log Z & (2.75)\\
&= \sum_{\mathbf{x}} \mathbb{P}(\mathbf{x}) \left[\log \mathbb{P}(\mathbf{x}) - \mathbb{E}_D \log M_D(\mathbf{x})\right] + \log Z & (2.76)\\
&= \sum_{\mathbf{x}} \mathbb{P}(\mathbf{x}) \left[\log \mathbb{P}(\mathbf{x}) - \log(\exp(\mathbb{E}_D \log M_D(\mathbf{x})))\right] + \log Z & (2.77)\\
&= \sum_{\mathbf{x}} \mathbb{P}(\mathbf{x}) \left[\log \mathbb{P}(\mathbf{x}) - \log \bar{M}^G(\mathbf{x})\right] & (2.78)\\
&= D_{KL}(\mathbb{P}||\bar{M}^G) \ . & (2.79)
\end{aligned}$$

To summarize, the mean KL divergence equals:

$$\begin{aligned}
\mathbb{E}_D D_{KL}(\mathbb{P}||M_D) &= D_{KL}(\mathbb{P}||\bar{M}^G) + \mathbb{E}_D D_{KL}(\bar{M}^G||M_D) & (2.80)\\
&= D_{KL}(\mathbb{P}||\bar{M}^G) + \mathbb{E}_D \sum_{\mathbf{x}} \bar{M}^G(\mathbf{x}) \log \frac{\bar{M}^G(\mathbf{x})}{M_D(\mathbf{x})} \ . & (2.81)
\end{aligned}$$

It is interesting to note that a decomposition similar to what was done for the arithmetic mean leads to

$$\mathbb{E}_D D_{KL}(\mathbb{P}||M_D) = D_{KL}(\mathbb{P}||\bar{M}^G) + \mathbb{E}_D \sum_{\mathbf{x}} \mathbb{P}(\mathbf{x}) \log \frac{\bar{M}^G(\mathbf{x})}{M_D(\mathbf{x})} \ . \qquad (2.82)$$

There is no contradiction however: any density can be substituted to the first $\bar{M}^G(\mathbf{x})$ in Equation 2.70. Substituting $\mathbb{P}(\mathbf{x})$ leads to Equation 2.82.

For both averaging scheme, the error of the mean model can be expressed as the mean error of a model minus the variance, based on respectively Equations 2.60 and 2.80:

$$D_{KL}(\mathbb{P}||\bar{M}) = \mathbb{E}_D D_{KL}(\mathbb{P}||M_D) - \mathbb{E}_D \sum_{\mathbf{x}} \mathbb{P}(\mathbf{x}) \log \frac{\bar{M}(\mathbf{x})}{M_D(\mathbf{x})} \tag{2.83}$$

$$D_{KL}(\mathbb{P}||\bar{M}^G) = \mathbb{E}_D D_{KL}(\mathbb{P}||M_D) - \mathbb{E}_D D_{KL}(\bar{M}^G||M_D) \ . \tag{2.84}$$

Because the two variances (the rightmost term) are positive, the error of any of the two mean models is smaller than the average error of a single model. If such a mean model could be generated, it would have on average a better accuracy than a model learned on a single learning set, because there would be no variance term in its error.

In order to identify whether one averaged model has a smaller error than the other, their respective variances must be compared. The difference between these two variances is

$$\Delta = \mathbb{E}_D \sum_{\mathbf{x}} \mathbb{P}(\mathbf{x}) \log \frac{\bar{M}(\mathbf{x})}{M_D(\mathbf{x})} - \mathbb{E}_D \sum_{\mathbf{x}} \mathbb{P}(\mathbf{x}) \log \frac{\bar{M}^G(\mathbf{x})}{M_D(\mathbf{x})} \tag{2.85}$$

$$= \mathbb{E}_D \sum_{\mathbf{x}} \mathbb{P}(\mathbf{x}) \log \frac{\bar{M}(\mathbf{x})}{\bar{M}^G(\mathbf{x})} \tag{2.86}$$

$$= \mathbb{E}_D \sum_{\mathbf{x}} \mathbb{P}(\mathbf{x}) \log \frac{\mathbb{E}_{D'} M_{D'}(\mathbf{x})}{\frac{1}{Z} \exp(\mathbb{E}_{D'} \log M_{D'}(\mathbf{x}))} \tag{2.87}$$

$$= \underbrace{\mathbb{E}_D \sum_{\mathbf{x}} \mathbb{P}(\mathbf{x}) \log \frac{\mathbb{E}_{D'} M_{D'}(\mathbf{x})}{\exp(\mathbb{E}_{D'} \log M_{D'}(\mathbf{x}))}}_{\geq 0} + \underbrace{\log Z}_{\leq 0} \ . \tag{2.88}$$

The first term is positive because the geometric mean is lower or equal to the algebraic mean, the second is negative because Z is positive and smaller or equal to one. It is therefore unclear whether one variance is smaller than the other. Although $\bar{M}^G$ is the probability density M minimizing $\mathbb{E}_D D_{KL}(M||M_D)$, it may not minimize $\mathbb{E}_D \sum_{\mathbf{x}} \mathbb{P}(\mathbf{x}) \log [M(\mathbf{x})/M_D(\mathbf{x})]$.

However, if the models M_D are good estimators of the real probability density $\mathbb{P}$, a combination of those estimators can be expected to be close to $\mathbb{P}$: $\mathbb{P}(\mathbf{x}) \approx \bar{M}(\mathbf{x})$. In this case, $\mathbb{E}_D \sum_{\mathbf{x}} \mathbb{P}(\mathbf{x}) \log \left[\bar{M}(\mathbf{x})/M_D(\mathbf{x})\right] \approx$

$\mathbb{E}_D D_{KL}(\bar{M}||M_D)$. Therefore, the variance associated to $\bar{M}^G$ is likely to be smaller than the one associated to the arithmetic mean, because $\bar{M}^G$ minimizes $\mathbb{E}_D D_{KL}(M||M_D)$.

This would also imply that the bias obtained in the decomposition based on this arithmetic mean model is lower:

$$D_{KL}(\mathbb{P}||\bar{M}^G) \geq D_{KL}(\mathbb{P}||\bar{M}) \ . \tag{2.89}$$

Therefore, combining all models (assuming they can be generated) using the arithmetic mean could lead to a lower error than combining them using the geometric mean.

These two bias-variance decompositions are illustrated in Figure 2.9. A Bernouilli density is estimated based on 60 observations and for different values of its parameter. The different quantities involved in the decompositions are computed analytically, but the expectations on D are truncated to remove "extreme" learning set containing all 0 or all 1, because those learning sets would lead to an infinite error. The parameters of the different estimations are inferred by the maximum likelihood principle, introduced in Section 3.3.3. This example shows that the arithmetic mean indeed leads to a smaller bias and a larger variance than the geometric mean.

2.5 High Dimensional Learning

Machine learning problems encountered in practice are continuously increasing in size, both in the number of samples and in the number of variables: high-throughput techniques enables the measurement of hundreds of thousands of genes or protein levels at the same time, hundreds of millions of smartphones collect data, multimedia content available on the web explodes...

Since the improvement in computational power is surpassed by the increase in data, this leads to computational complexity problems. And since the number of variables is often increasing faster than the number of samples, handling those problems becomes more challenging even without considering computational complexity.

In this section these two aspects will be briefly presented, with a focus on a high number of variables since this thesis aims at developing new methods

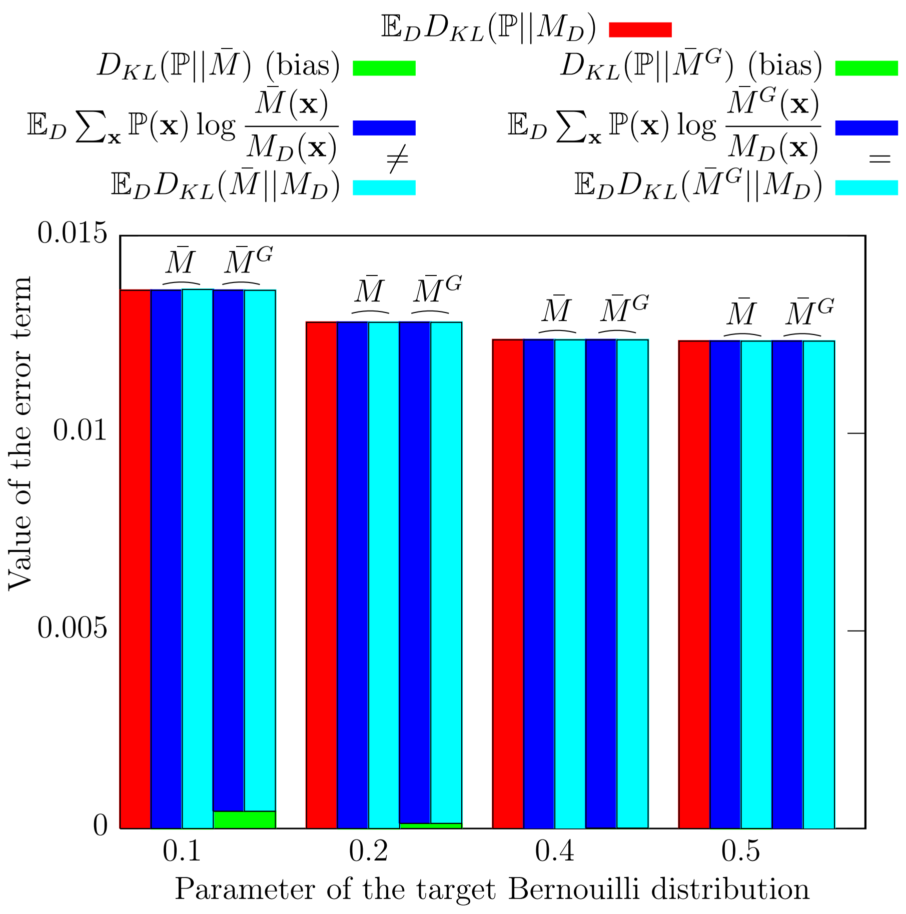


Figure 2.9: Two bias variance decompositions for the Kullback-Leibler divergence are illustrated through the estimation of a Bernouilli density. Sets of 60 observations and maximum-likelihood estimation are considered. The different quantities involved are computed analytically. However, $\mathbb{E}_D$ does not average over all possible learning sets: sets containing only 0 or 1 are discarded, because they would lead to an infinite value of the average error $\mathbb{E}_D D_{KL}(\mathbb{P}||M_D)$. For each value of the parameter of the Bernouilli density, the first column is this average error, and the last four a bias (bottom) plus a variance term (top) in a decomposition.
The 2^{nd} column is based on the arithmetic mean (Equation 2.60), the 4^{th} and 5^{th} (Equations 2.80 and 2.82) on the geometric mean. In the 3^{rd}, $\mathbb{E}_D D_{KL}(\bar{M}||M_D)$ is not part of any decomposition, but is provided here for comparison with $\mathbb{E}_D D_{KL}(\bar{M}^G||M_D) = \mathbb{E}_D \sum_{\mathbf{x}} \mathbb{P}(\mathbf{x}) \log(\bar{M}^G(\mathbf{x})/M_D(\mathbf{x}))$, the variance of the decomposition based on the geometric mean. The density used to average on $\mathbf{x}$ is irrelevant for this mean. The variance based on the arithmetic mean is $\mathbb{E}_D \sum_{\mathbf{x}} \mathbb{P}(\mathbf{x}) \log(\bar{M}(\mathbf{x})/M_D(\mathbf{x}))$.
Notice how the bias is larger and the variance smaller for the geometric mean than for the arithmetic mean.

for density estimation for inference on high-dimensional problems (in the number of variables).

2.5.1 Large Number of Samples N, Small p

A larger learning set means a lower variance (see Section 2.4) and hence more accurate results. However it also increases learning time, since more samples have to be processed.

Special care should thus be taken when programming the algorithms. Techniques used to reduce the load include optimizing memory access, preprocessing the data to compress it or learning on a subset of samples only.

2.5.2 High Number of Variables p, Small N

A high number of variables leads to similar algorithmic complexity problems, although generally in different parts of the algorithms.

However another possible issue in such a problem stems from the learning set. There are in this context too few observations. As an example, recent developments in data acquisition technology such as microarrays have drastically increased the amount of variables that can be measured at once for a given problem. The number of observations collected has however not increased by the same order of magnitude, and it is common to work with N in the order of p, or smaller (often denoted as $p >> N$).

This low number of observations is called "the curse of dimensionality". In high-dimensional spaces, with thousands or more variables, the observations become "sparse". This lack of data makes it difficult to assess statistical significance, and selecting the best model can be very difficult, since many can fit the learning set. The variance in these situations also increases. This is illustrated in Figure 2.10 for a binary classification problem with two continuous features and one observation from each class. An infinite number of linear separators between the two classes perfectly classifying the learning set can be constructed. The partition of the plane induced by these separators vary a lot, and it is possible to construct two separators (2 lines arbitrarily close to each observation, but on opposite sides) leading to two opposite classifications of nearly every point of the plane.

Special techniques therefore have to be developed to constrain model complexity and limit overfitting. One such technique is regularization,

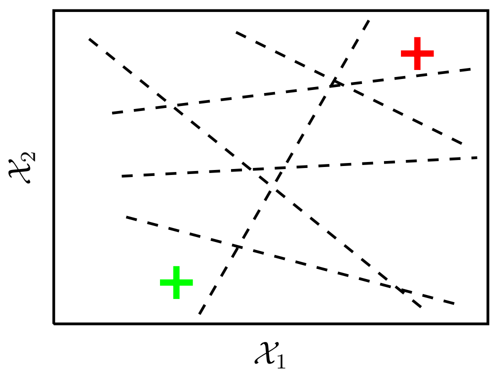


Figure 2.10: Learning a linear boundary for binary classification based on two feature variables is difficult when only $p = N = 2$ observations are available, one for each class. Indeed, the set of all possible boundaries perfectly classifying the learning set covers (almost) the whole plane. This is called the "curse of dimensionality".

which modifies a score to include a new term penalizing complexity. Another is the use of mixtures, i.e. averaging several different models to reduce the variance. A first hint about this technique has been given in Section 2.4, and it is described in the next section more thoroughly.

Both methods will be studied in this thesis for graphical models (mostly for Markov trees): mixtures because they constitute the main topic of this thesis, regularization as a competing method. These notions will be covered in the following order: mixtures in the next section, probabilistic graphical models in the next chapter. Then, in Chapter 4, mixtures of probabilistic graphical models, and in of particular Markov trees, will be presented. Regularization of these Markov trees will be discussed in Section 5.1.1.

2.6 Mixture Models

A mixture model (also called ensemble model) is composed of a set of models, each of them capable to provide on its own an answer for the problem at hand, and an averaging scheme for combining the predictions of the different models into one common answer, usually a convex combination. In particular, probability densities are commonly averaged. In that case the probability of a set of variables $\boldsymbol{\mathcal{X}}$ is

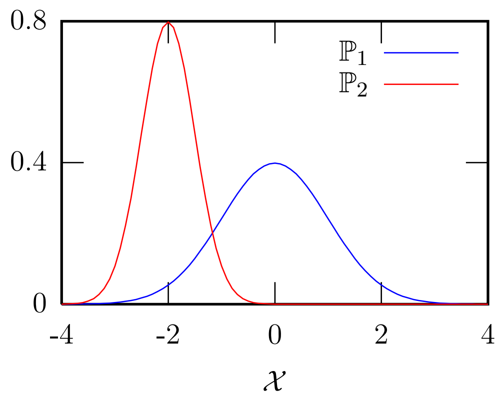
$$\mathbb{P}(\boldsymbol{\mathcal{X}}) = \sum_{i=1}^{m} \lambda_i \mathbb{P}_i(\boldsymbol{\mathcal{X}}) \ , \tag{2.90}$$

where $\mathbb{P}_i(\boldsymbol{\mathcal{X}})$ is the probability density encoded by the ith term of the mixture and $\sum_{i=1}^{m} \lambda_i = 1$, $\lambda_i > 0 \ \forall i$.

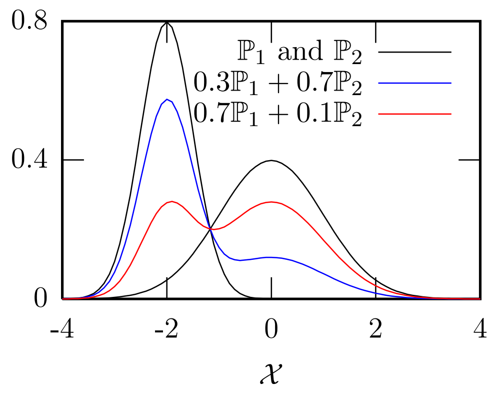
A well known example is the mixture of normal densities, a convex combination of m such densities $\mathcal{N}(\mu_i, \sigma_i)$, each weighted by λ_i and characterized by a mean μ_i and a variance σ_i^2:

$$\sum_{i=1}^{m} \lambda_i \mathcal{N}(\mu_i, \sigma_i) \ . \tag{2.91}$$

An illustration of a mixture of two univariate normal densities is displayed in Figure 2.11. Combining the two densities creates a more complex density function, whose shape is controlled by the weights of the mixture (and the parameters of the terms).



(a) The two terms of the mixture are $\mathbb{P}_1 = \mathcal{N}(0, 1)$ and $\mathbb{P}_2 = \mathcal{N}(-2, 0.5)$.



(b) The weights of the mixture control its shape.

Figure 2.11: Two normal densities are combined in a mixture, according to Equation 2.90.

Aggregating models in a mixture can be performed to reduce either the bias or the variance. As a short reminder, the error of a model with respect to the best possible model is a function of both the number of samples and the complexity of the class of candidate models. This decomposition of the error was illustrated in Figure 2.7 and contains two terms, the bias and the variance. Learning in a more complex class of models tends to increase the variance, learning in less complex one to increase the bias.

A mixture model can be built to either decrease the bias or the variance. The term *mixture model* tends to be used when referring to a model decreasing the bias, and the term *ensemble model* when referring to a model decreasing the variance. However, there are exceptions (such as boosting often being referred to as an ensemble method). In this thesis, all these models will be denoted by the term *mixture*.

These two types of mixtures will be discussed in respectively Sections 2.6.1 and 2.6.2. Finally, selected applications of mixtures in machine learning (but not for density estimation) will next be presented in Section 2.6.3. Mixtures of probabilistic models and in particular mixtures of Markov trees will be covered in Chapter 4.

Mixtures decreasing both have also been developed, e.g. [Bre99, Web00], but will not be discussed here.

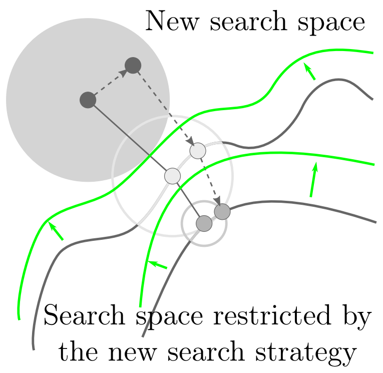


Figure 2.12: Optimizing the mixture as a global model reduces the bias. The mixture has more degrees of freedom, and the search space is expanded: the model bias decreases. Note that the new points representing the model in the new search spaces are not represented. See Figure 2.7 for the original learning problem.

2.6.1 Mixtures as a Mean to Reduce the Bias

In the first framework, learning the mixture is viewed as a global optimization problem aiming at modelling the set of observations as well as possible. Building a mixture extends the modelling capacity of an original model while allowing easier interpretation and manipulation of the resulting complex model, since it is constituted by combining simpler entities. Rather than a potentially complex model, the problem is represented by a combination of simple models, usually chosen for their tractability. As illustrated in Figure 2.12, the mixture is in this case considered as a single model, a model more complex than one of its term. The search space is expanded. Therefore, the model bias is reduced. The variance however may increase.

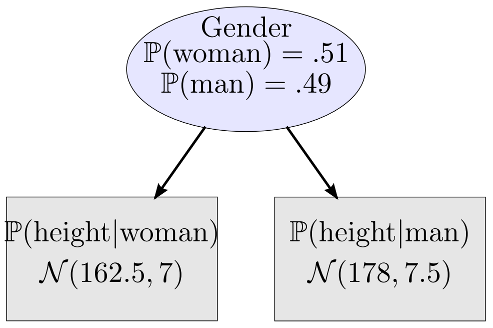
This combination can also be motivated by a problem which by itself presents different subgroups, or different phenomena. In that setting, each model is a different alternative solution, a different mechanism of the problem. In this case, each observation is supposed to have been generated by one term of the mixture. The weight of each term corresponds to the probability that the particular problem is the one active. The weights can also be seen as the different probabilities of a categorical hidden variable conditioning the different models, as illustrated in Figure 2.13. In this example, the height of a human is supposed to depend on the gender and to be normally distributed conditionally on the gender:

$$\begin{aligned} \mathbb{P}(\text{height}) &= \mathbb{P}(\text{woman})\mathbb{P}(\text{height}|\text{woman}) + \mathbb{P}(\text{man})\mathbb{P}(\text{height}|\text{man}) \quad (2.92) \\ &= 0.49\ \mathcal{N}(162.5, 7) + 0.51\ \mathcal{N}(178, 7.5)\ . \quad (2.93) \end{aligned}$$

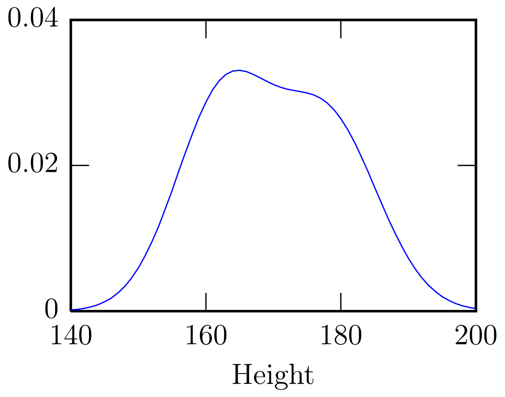
Modelling $\mathbb{P}(\text{height})$ with a single normal density would result in a bias. The mixture expands the modelling capacity of the model.

In the context of statistical estimation, another reason to use mixture models comes from their flexibility: a mixture can approximate any density arbitrarily well, provided the number of terms is large enough [MP99, MP00]. In that case the terms of the mixture are not interpreted as alternative problem mechanisms, but they attempt to locally approximate the probability density, e.g. by using one term per mode of the density.

No matter what the motivation for building this mixture is, increasing the number of terms of the mixture means increasing its flexibility, the complexity of the model space where the best fit for the learning sample is



(a) The mixture can be represented as a model with a hidden variable (gender), whose probabilities are the weights of the mixture.



(b) The resulting probability density has 2 maximums.

Figure 2.13: An (artificial) density of human height based on a mixture of normal distributions, is viewed as a model with a latent variable Z selecting among the male or female densities. Male heights are supposed to be of mean 178 and variance 7.5, female heights of mean 162.5 and variance 7.

searched. Hence this number of terms controls the complexity of the model: a larger number of terms may decrease the model bias.

An excellent reference on that type of mixture is [MP00]. Learning of such a mixture based on a set of observations is now considered.

Learning a Finite Mixture of Densities

A popular method to learn a finite mixture of densities is by maximizing the likelihood of the learning set through an iterative process: the Expectation-Maximization (EM) algorithm [DLR77, MK08]. This algorithm selects the parameters ψ of a statistical model where only some variables $\boldsymbol{\mathcal{X}}$ are observed, and $\boldsymbol{\mathcal{Z}}$ denotes those that are unobserved.

If all variables $\boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{Z}}$ were observed in the learning set D, the parameters ψ could be found by maximizing the complete log likelihood:

$$\log L_c(\psi) = \sum_{j=1}^{N} \log \mathbb{P}(\boldsymbol{\mathcal{X}} = \mathbf{x}_{D_j}, \boldsymbol{\mathcal{Z}} = \mathbf{z}_{D_j} | \psi) \ , \tag{2.94}$$

i.e. by setting ψ so that

$$\nabla_\psi \log L_c(\psi) = 0 \ . \tag{2.95}$$

However since only $\boldsymbol{\mathcal{X}}$ is observed, the quantity to be maximized is the likelihood

$$\log L(\psi) = \sum_{j=1}^{N} \log \mathbb{P}(\mathbf{x}_{D_j}|\psi) \tag{2.96}$$

$$= \sum_{j=1}^{N} \log \int_{\boldsymbol{\mathcal{Z}}} \mathbb{P}(\boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{Z}}|\psi) d(\boldsymbol{\mathcal{Z}}) \ . \tag{2.97}$$

The EM algorithm attempts to solve that problem by maximizing the complete log likelihood. Since computing it requires the values $\mathbf{z}_{D_j}$, the algorithm alternates between constructing estimates of the unobserved value $\mathbf{z}_{D_j}$ based on both the observations and the current value of the parameters (E step), and optimizing ψ on the complete log-likelihood based on the observations and these estimated values for $\boldsymbol{\mathcal{Z}}$ (M step).

More formally, the E step consists in computing the expectation (over $\boldsymbol{\mathcal{Z}}$) of $\log L_c(\psi)$, using the current value of the parameters ψ^t:

$$Q(\psi; \psi^t) = \mathbb{E}_{\boldsymbol{\mathcal{Z}}|\psi^t} \left\{ \log L_c(\psi) \right\} \ ; \tag{2.98}$$

and in the M step a new value ψ^{t+1} of the parameters is obtained by maximizing this quantity:

$$\psi^{t+1} = \arg\max_{\psi} Q(\psi; \psi^t) \ . \tag{2.99}$$

Applying those two steps can only increase $L(\psi)$, and so they are repeated until the likelihood increase is small enough:

$$L(\psi^{t+1}) - L(\psi^t) \leq \epsilon \ , \tag{2.100}$$

with $\epsilon \in \mathbb{R}^+$ a user specified tolerance.

This algorithm can be applied to the estimation of a mixture of m models by considering one hidden variable $\mathcal{Z}$ conditioning the different terms of the mixture. In that case, the mixture density is rewritten as:

$$\mathbb{P}(\boldsymbol{\mathcal{X}}) = \sum_{i=1}^{m} \mathbb{P}(\mathcal{Z} = i) \mathbb{P}(\boldsymbol{\mathcal{X}}|\mathcal{Z} = i) \ , \tag{2.101}$$

where $\mathbb{P}(\mathcal{Z} = i) = \lambda_i$ is the weight of the i^{th} term of the mixture and $\mathbb{P}(\boldsymbol{\mathcal{X}}|\mathcal{Z} = i)$ is the probability density encoded by this term. The value of $\mathcal{Z}$ can be understood as the label of the term that generated a given sample.

The parameters ψ can be divided into two categories: the weights $\lambda_1,\ldots,\lambda_m$ and the parameters of the different terms $\theta_1,\ldots,\theta_m$. The complete likelihood of the learning set augmented by the unknown variable is

$$\begin{aligned}\log L_c(\psi) &= \sum_{j=1}^{N} \log \mathbb{P}(\mathbf{x}_{D_j}, z_{D_j}|\psi) &(2.102)\\ &= \sum_{j=1}^{N} \log \left[\mathbb{P}(z_{D_j}|\psi)\mathbb{P}(\mathbf{x}_{D_j}|z_{D_j},\psi)\right] &(2.103)\\ &= \sum_{j=1}^{N} \left[\log \mathbb{P}(z_{D_j}|\psi) + \log \mathbb{P}(\mathbf{x}_{D_j}|z_{D_j},\psi)\right] &(2.104)\\ &= \sum_{j=1}^{N}\sum_{i=1}^{m} \delta(z_{D_j} = i)\left(\log\lambda_i + \log \mathbb{P}_i(\mathbf{x}_{D_j}|\theta_i)\right) \quad . &(2.105)\end{aligned}$$

Hence,

$$\begin{aligned}Q(\psi;\psi^t) &= \mathbb{E}_{\boldsymbol{\mathcal{Z}}|\psi^t}\left\{\log L_c(\psi)\right\} &(2.106)\\ &= \sum_{j=1}^{N}\sum_{i=1}^{m} \mathbb{E}_{\boldsymbol{\mathcal{Z}}|\psi^t}\left\{\delta(z_{D_j} = i)\right\}\left(\log\lambda_i + \log \mathbb{P}_i(\mathbf{x}_{D_j}|\theta_i)\right) \quad . &(2.107)\end{aligned}$$

By denoting $\mathbb{E}_{\boldsymbol{\mathcal{Z}}|\psi^t}\left\{\delta(z_{D_j} = i)\right\}$ as $\tau_i^t(j)$, the two steps of the EM algorithms are:

E-step:

$$\tau_i^t(j) = \frac{\lambda_i^t \mathbb{P}_i(\mathbf{x}_{D_j}|\theta_i^t)}{\sum_{k=1}^{m} \lambda_k^t \mathbb{P}_k(\mathbf{x}_{D_j}|\theta_k^t)} \tag{2.108}$$

This corresponds to a soft assignment of each sample to the different terms of the mixture.

M-step: λ_i^{t+1} is obtained $\forall i$ by

$$\lambda_i^{t+1} = \arg\max_{\lambda_i} \sum_{j=1}^{N} \tau_i^t(j) \log \lambda_i \ \ , \text{with} \sum_{i=1}^{m} \lambda_i = 1 \tag{2.109}$$

$$= \frac{1}{N} \sum_{j=1}^{N} \tau_i^t(j) \ \ , \tag{2.110}$$

and θ_i^{t+1} is defined as

$$\theta_i^{t+1} = \arg\max_{\theta} \sum_{j=1}^{N} \tau_i^t(j) \log \mathbb{P}_i(\mathbf{x}_{D_j} | \theta_i) \ \ , \tag{2.111}$$

i.e. the values of the parameters θ_i maximizing the likelihood of a learning set D_i^t composed of the original samples, each weighted by $\tau_i^t(j)$. D_i^t corresponds to the samples assigned to that particular term of the mixture in the E step. Depending on the class of densities considered, this may be computable in closed form.

While the EM algorithm converges towards a local maximum of the likelihood function, there is no guarantee that this local maximum is unique. Therefore it is common to run the algorithm from several different random initialization points and to keep only the best value.

Number of components: The EM algorithm optimizes a mixture for a given number of components m, but choosing this number is non-trivial, since it controls the complexity of the model. When $m = 1$, where there is no mixture but a single term. When $m = N$, the number of samples, the EM algorithm may attribute one and only one sample to each term of the mixture. In that case and depending on the learning algorithm for each term, the mixture may be equivalent to a kernel estimate of the density, where each term of the mixture is one density (the kernel), positioned on each sample.

Several techniques have been developed to select that number. A general method to fit the complexity of the model is to penalize the optimization criterion, here the likelihood function, by adding a term depending on the number of parameters in the model. Another class of methods use the likelihood as a test statistic to perform a hypothesis test. Bootstrapping and cross-validation can be used to refine those analysis [MP00].

On a sidenote, a Bayesian approach to the selection of the number of components is also possible. A prior density is defined on the number of components. Mixtures of different sizes can be considered together. Note that this approach is akin to defining a new mixture of the variance reduction type (as discussed in Section 2.6.2) on top of the ML mixture.

2.6.2 Mixtures as a Mean to Reduce the Variance

Mixtures have been shown to be capable of reducing the bias. In this section, they are used to reduce the other component of the error, the variance. The variance was defined (see Section 2.4) as the expected difference between a model learned on a learning set and the average of that model.

From this definition, one can get the intuition that averaging models leads to a reduction in variance. Indeed, should we be able to average over all possible leaning sets,

$$\int M_D(.)\mathbb{P}(D)dD \ , \tag{2.112}$$

the variance would be zero. This was expressed more formally in Equations 2.83 and 2.84 for bias-variance decomposition based respectively on the arithmetic mean and on the geometric mean. The KL divergence of the average model to the target distribution is the bias, and is equal to the mean KL divergence of a model learned on a given learning set minus the positive variance. The KL divergence of the average model is therefore smaller than the mean KL divergence of a model learned on a given learning set.

This has long been observed in practice: asking a question to several individuals and averaging their answers usually leads to a better answer than a single expert would provide. This phenomenon is called the "wisdom of the crowd". A well-known and perhaps the first illustration is reported in [Gal07]: participants at an annual West of England Fat Stock and Poultry Exhibition were given for a fee the possibility to guess how much a live ox would weight after being slaughtered and "dressed". The most accurate estimates were granted rewards. The error of the mean value of the guesses was only 1% of the real weight, while the mean error was 3.1%.

The former example can be seen as a practical illustration of the property mentioned just above. Each individual participating in the contest uses a different (internal) model of the problem, and averaging the predictions of those models yields a result closer to the true value than the mean error.

Obtaining the probability density over the learning set (or over the models), or sampling models directly, as in that particular example, is usually not possible in machine problem. The density on the models or the learning set is unavailable, since only a single set of observations is available.

In the remaining part of this section two methods for approximating the ideal mixture described above are discussed: the Bayesian and the perturb and combine framework, the latter being the focus of this thesis. Boostrap aggregation, a method belonging to the perturb and combine framework, is also described.

Bayesian Approach to Mixtures

As opposed to the traditional approach of using the best model, this technique consists in averaging all possible models, weighted by their respective probability according to the learning set D:

$$\int M(.)\mathbb{P}(M|D)dM \ , \tag{2.113}$$

where $\mathbb{P}(M|D)$ is the probability of the model according to the learning set D and is provided by the Bayes theorem:

$$\mathbb{P}(M|D) = \frac{\mathbb{P}(D|M)\mathbb{P}(M)}{\mathbb{P}(D)} \ . \tag{2.114}$$

$\mathbb{P}(M)$ is called the prior probability of a model and can be used to incorporate some extra information about the problem (such as expert knowledge) into the learning process, or to regularize (by assigning a lower prior probability to complex models). The probability of the learning set $\mathbb{P}(D)$ is a normalization constant computed by averaging over all possible models:

$$\mathbb{P}(D) = \int \mathbb{P}(D|M)\mathbb{P}(M)dM \ . \tag{2.115}$$

Depending on the prior and posterior density, the resulting expression may be closed-form.

Perturb and Combine

A pure Bayesian approach may be impractical: the integral of the posterior density may be intractable or sampling from it may be too expensive.

In this section, a different way of generating several models from a given learning set is described: the perturb and combine framework. It has led to many successful algorithms in supervised learning (e.g. [Fre95, Bre96, Bre01, EGWL05, GEW06]).

Suppose a stochastic algorithm is available to generate a model depending on its input learning set. Because the algorithm is stochastic, this model will differ (with high probability) from the output of a new run of the algorithm on the same input. This algorithm could be repeatedly applied to generate an ensemble of distinct models, which could be aggregated in a mixture.

Some machine learning algorithms are inherently stochastic, e.g. algorithms starting from a random initialization of their parameters and performing a gradient descent on an optimization criterion containing multiple local maxima. Those algorithms can be used to generate an ensemble of models. Not all machine learning algorithms are inherently stochastic.

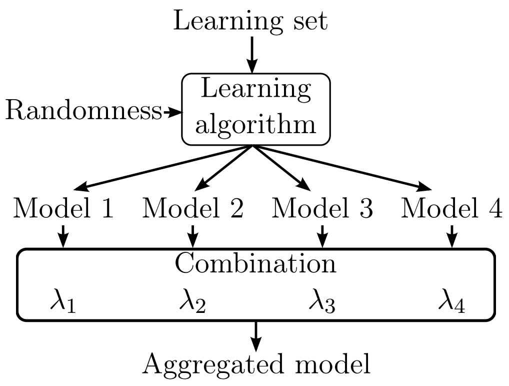
The perturb and combine framework randomize the algorithms so that they become stochastic and can be used to generate mixtures. It consists in:

- introducing randomization in the algorithm, which can be deterministic or a stochastic one where more randomization is desired,
- applying this perturbed algorithm several times on the learning set,
- combining in some way the predictions of the resulting ensemble of models.

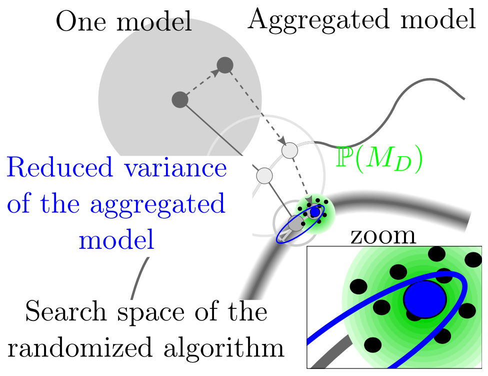
This procedure results in the construction of a mixture model and is illustrated in Figure 2.14, along with its effect on the error. Modifying the search strategy may increase both the variance and the estimation bias of a single model (black dot). Combining several models however reduces the variance. This reduction increases with the number of terms generated, and may outweigh the increased error of a single model.

Randomizing the algorithm can be achieved through changes to the algorithm itself, e.g. replacing a deterministic choice by a stochastic one, or through modifications to the data set. As for any mixture, the combination of the models can be an arithmetic mean, a majority vote etc.

The challenge of this framework is to select appropriate randomization and averaging schemes in order to obtain a reduction in variance without



(a) The perturb and combine framework perturbs an algorithm to make it stochastic, repeatedly applies it on the learning set to generate several models, and combines them.


One model
Aggregated model
Reduced variance of the aggregated model
$\mathbb{P}(M_D)$
zoom
Search space of the randomized algorithm

(b) Perturbing the search strategy of the learning algorithm can modify the search space. In addition, several different models may be output for a single learning set. Several models are generated, and then aggregated. The variance of the result is usually smaller (blue ellipse).

Figure 2.14: The concept of the perturb and combine framework, and its effect on the error are illustrated.

deteriorating the bias too much. An increase in the bias is tolerable provided the gain in variance surpasses it.

To illustrate the perturb and combine framework, bootstrap aggregation will now be described.

Bootstrap Aggregation

Bootstrap aggregation or bagging [Bre96] is a meta-algorithm belonging to the perturb and combine framework described in the previous section (2.6.2). The randomness introduced in the learning algorithm consists in a modification of the learning set. Bagging compensates for a lack of data by applying the learning algorithm on several bootstrap replicates of the original learning set and averaging the predictions of the resulting models. A bootstrap replicate D' of size N' is obtained from an original data set D of N samples by uniformly and independently drawing N' natural numbers

$r_i \in [1, N]$, and by compiling $\mathbf{D}'$ by

$$\mathbf{x}_{D'_i} = \mathbf{x}_{D_{r_i}} \qquad \forall i \in [1, N'] \ , \tag{2.116}$$

where $\mathbf{x}_{D_j}$ (resp. $\mathbf{x}_{D'_k}$) refers to jth (resp. kth) observation of D (resp. D'), and where typically $N = N'$. This process is illustrated in Figure 2.15 for $N = 5$.

When $N = N'$, the expected proportion of samples from D present once or more in D' asymptotically converges to 63.2% as N increases.

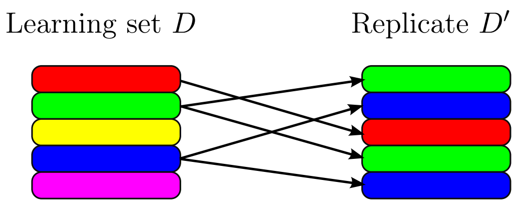


Figure 2.15: To construct a bootstrap replicate, observations are randomly selected from the original (with replacement). In this illustration, the sequence of observations selected is 2,4,1,2,4.

Number of components: In bagging, the bootstrap replicate used for learning each term of the models is independent from other replicates conditionally on the original learning set. Therefore each term of the mixture has the same expected model and therefore the same estimation and model bias. These biases may be larger than those of a model learned on the original learning set. However, averaging these terms does not modify this bias.

Learning a model on a bootstrap replicate rather than on the original learning set increases the variance. However, aggregating several of these terms decreases the variance.

Actually, the more terms m in the mixture the greater the variance reduction effect of the method ($\mathcal{O}(1/m)$). An infinite number of terms would minimize the variance of the aggregated model. However, the aggregated model usually converges when the number of terms is high enough (i.e. a few hundreds, a problem dependent number). The generation of more terms can also be stopped because of a practical constraint, such as memory usage or learning/exploitation time of the model.

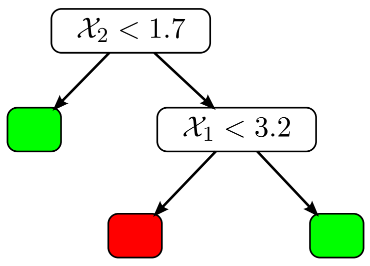
2.6.3 Mixture Models in Automatic Learning

This section contains illustrations of the previous discussions. A selection of mixture models used in machine learning are presented, and the links with the different techniques for building mixtures are established.

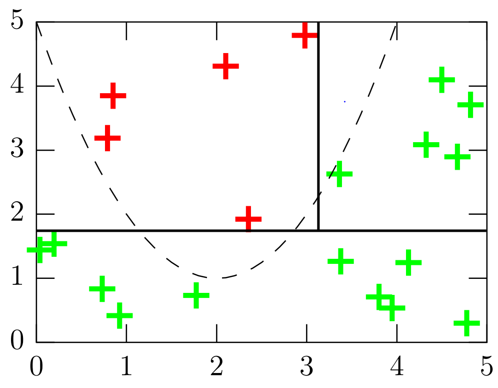
Random Forests

A random forest [Bre01] is a mixture of decision trees used for classification or regression. These trees are constructed independently from each other by a randomized version of the algorithm for learning a decision tree minimizing the classification error of the learning set. A decision tree, illustrated in Figure 2.16, is a model used for classification. It is a directed tree graph (although not a Bayesian network), where each leaf (node without outgoing edge) is labeled by a class, and each other node has two children and is labeled by a condition on a variable, e.g. *"Is* $\mathcal{X} > 0.3$*?"*.

The class of an observation is predicted as follows. Start at the root of the tree (the node without any incoming edge). For every non-leaf node , go down to the left child node if the condition of the current node is true, and to the right child otherwise. When a leaf is reached, the class associated to this leaf is attributed to the observation.



(a) Decision tree model



(b) The decision tree induces a partition of the input variable space.

Figure 2.16: A decision tree perfectly classifies the learning set of the binary classification problem of Figure 2.3.

The standard algorithm for learning such a model starts by a single node, and recursively splits nodes until the resulting tree perfectly classifies the learning set. A node is split by testing every possible condition on the elements of the learning set reaching this node, and selecting the one maximizing the entropy reduction in the classification variable associated to this split.

The randomization is introduced at the construction of each node, by considering only a subset of K possible conditions. The combination of several randomized decision trees usually leads to much better results than the optimal tree.

Clustering

Mixtures of both types have been used in clustering.

Mixtures used to reduce the bias, such as those learned by the EM algorithm, can be viewed as a combination of different mechanisms of the problem. Moreover, Section 2.6.1 explained how the EM algorithm constructs a partition[1] of the observations and how each term of the model is learned based on a different subset of observations resulting from this partition. Observations associated to the same term of the mixture are more similar among themselves than to observations associated to another term of the mixture. Therefore the observations can be clustered by creating one cluster corresponding to each term.

Mixtures are also used as a mean to reduce the variance of a clustering algorithm. Several different clusterings are created, and then aggregated to produce a more robust clustering [SG03, TJP04].

2.7 Outlook

This thesis focuses on learning mixtures of probabilistic graphical models for density estimation in high-dimensional settings. The present chapter covered machine learning and mixtures of models, while probabilistic graphical models will be presented in a dedicated chapter (Chapter 3). The following relevant elements were in particular presented in the current chapter.

[1]Although the partition is soft, each observation can be associated to the most likely term for this observation, randomly breaking ties.

Section 2.3 has established how the quality of a model and of an algorithm can be quantified. This information will be used to establish a validation protocol for our algorithms, presented in Section 2.7.1. Moreover, the analysis of an error of a model led to a decomposition in the sum of two positive terms, the bias and the variance, in Section 2.4.

Section 2.5.2 established that the variance, one component of the error, typically increases with the number of variables of the problem, for a constant number of observations and model type. This thesis will therefore focus on mixtures for reducing the variance.

Another challenge in high-dimensional problems is algorithmic complexity. A particular attention will therefore be given to the complexity of the algorithms considered, both in single models and in mixtures.

This motivated the topic of this thesis: to explore the possibility of mixtures of simple probabilistic graphical models, in particular for the reduction of variance. What constitutes a simple (in terms of algorithmic complexity) model will be discussed in the next chapter, in particular in Sections 3.3.2 and 3.4.1. These discussions will lead in Section 4.1 to the selection of the models used to construct mixtures in this thesis. Alternative works with other models will nevertheless be briefly presented in Chapter 4, where mixtures and PGM are combined.

2.7.1 Experimental Validation

Section 2.3 outlined how machine learning algorithms can be compared. The algorithms developed in this thesis will be tested on different target densities and learning sets.

- Different target densities are used, both synthetic and inspired by real problems.
  - Synthetic densities have the advantage to be easily controllable. Control over the generation process makes it possible to modify a characteristic of the problems, such as e.g. the number of variables, while leaving other characteristics of the problem more or less unchanged. This is important, because the behavior of algorithms may change based on these parameters. As an example, variance may take over bias as the main components of the error when the number of variables increases. These changes may favor different learning algorithms.

  Another advantage of synthetic densities is that for any given problem setting, several densities can be generated, reducing the variance of the experiments.

  - Realistic densities derived from real problems are also considered. The conclusions drawn from experiments carried on on such densities are more likely to apply on real problems.

- Different number of observations are considered. Again, this parameter of the problem may influence the relative accuracy of the learning algorithms with respect to one another. Testing the sensitivity of the results to a modification of the number of observations is thus necessary. However, the number of observations considered here will most of the time be small with respect to the number of variables, because high dimensional problems usually have few observations.

The different learning sets used in this thesis to evaluate the different algorithms are described in Appendix A. In particular, the origin and/or the generation mechanism used to construct them are provided.

Chapter 3

Probabilistic Graphical Models

This chapter provides background information about probabilistic graphical models and covers more in depth a few notions related to this thesis. In particular, a special attention is given to Markov trees, since they are the main model on which this thesis is based. On the other hand, mixtures of graphical models will be discussed in Chapter 4. Four excellent references covering the exciting field of probabilistic graphical models are [Pea88, NWL$^+$07, Dar09, KF09], on which most of the information provided here is based. The first three are slightly more accessible while the latter is a more exhaustive reference covering many very technical points. A reader already familiar with probabilistic graphical models might want to skip directly to Section 3.5 where I precise the context of this thesis with respect to the framework presented in this chapter.

The present chapter is organized as follows. Probabilistic graphical models and several classes of such models are first introduced intuitively and different classes of probabilistic graphical models are formally defined in Section 3.1. The usefulness of those models is then illustrated in Section 3.2 where some applications of those models are discussed. Sections 3.3 and 3.4 take a deeper look into respectively learning those models from a data set and performing inference on them. The complexity of those operations is first discussed, since they motivate my focus on simple graphical models. A few well-known algorithms are also presented to give a better understanding of those two processes.

3.1 Definition and General Presentation

Probabilistic graphical models combine graph theory with probability theory to efficiently encode a joint probability distribution over a set $\boldsymbol{\mathcal{X}}$ of p variables $\{\mathcal{X}_1, \mathcal{X}_2, ..., \mathcal{X}_p\}$. Their development was motivated by the need for a compact and interpretable representation of a joint probability distribution exploiting the problem structure.

Consider as an example the discrete random variable $\mathcal{X}_1$, defined as the outcome of the toss of a coin. $\mathcal{X}_1$ can take either of the values heads or tails (its cardinality $|Val(\mathcal{X}_1)|$ is 2) and its probability distribution is completely defined by two numbers, $\mathbb{P}(\mathcal{X}_1 = \text{heads})$ and $\mathbb{P}(\mathcal{X}_1 = \text{tails})$. Since these numbers must sum to one, the probability distribution can actually be completely defined by $|Val(\mathcal{X}_1)| - 1 = 1$ parameter ρ:

$$\mathbb{P}(\mathcal{X}_1 = \text{heads}) = \rho \tag{3.1}$$
$$\mathbb{P}(\mathcal{X}_1 = \text{tails}) = 1 - \rho \ . \tag{3.2}$$

If k coins are tossed into the air rather than one, the result of the experiment can be modelled as a set $\boldsymbol{\mathcal{X}}$ of k random variables, one for each coin. There are now $|Val(\boldsymbol{\mathcal{X}})| = |Val(\mathcal{X}_i)|^k = 2^k$ possible outcome to the experiments, so $2^k - 1$ probabilities must be specified, a tedious work if k is large.

The independence relationships between the variables can however be exploited to reduce that number. In the coin tossing problem, a reasonable assumption is that the result of one coin does not influence the other coins. Under such a hypothesis, only k parameters must be specified, one ρ_i for each $\mathcal{X}_i$, defined similarly to the previous example. The probability of any possible result can then be computed by

$$\mathbb{P}(\mathcal{X}_1 = x_1, \ldots, \mathcal{X}_p = x_p) = \prod_{i=1}^{p} \mathbb{P}(\mathcal{X}_i = x_i) \ . \tag{3.3}$$

Exploiting the independence relationships in the latter example is easy by hand or on a computer due to the symmetry of the problem. When the number of variables increases, when those variables are diverse and are connected asymmetrically, a human or a computer working on the joint probability distribution over those variables must have a representation of these relationships to manipulate them. This is precisely what the field of probabilistic graphical models is about.

This section contains a formal definition of probabilistic graphical models and of several classes of such models. Each class will be described in terms of the independence relationships it can encode, and of the actual encoding of the distribution.

3.1.1 Probabilistic Graphical Models

A **probabilistic graphical model** is composed of a set of parameters θ and a graphical structure $\mathcal{G} = (V, E)$, with V the nodes or vertices of the graph and $E \subset V \times V$ the edges. $\mathcal{G}$ encodes a set of independence relationships between variables. Typically, there is a bijection between the nodes of the networks and $\boldsymbol{\mathcal{X}}$, and the edges between nodes encode these relationships, which can usually be visually inferred from the graph. The parameters θ quantify the probability distribution. Note that $\boldsymbol{\mathcal{X}}$ can be composed of discrete or continuous variables, or a mixture of both types. Unless stated otherwise, I will however consider discrete variables only.

Both the exact encoding of the independence relationships and the parametrization of the distribution depend on the class of probabilistic graphical models considered. A probabilistic graphical model usually encodes a probability distribution by a product of k functions, where each function depends on a subset of variables only. These functions are called factors and typically:

$$\phi_i : Val(\boldsymbol{\mathcal{S}}_i) \to \mathbb{R}^+ \qquad \forall i \ . \tag{3.4}$$

Using $\phi_i(\boldsymbol{\mathcal{S}}_i)$ to denote these factors, with s_i the number of variables in $\boldsymbol{\mathcal{S}}_i$, the argument of factor ϕ_i, the probability distribution is:

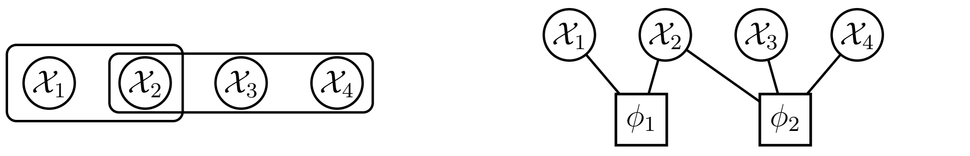
$$\mathbb{P}(\boldsymbol{\mathcal{X}}) = \frac{1}{Z} \prod_{i=1}^{k} \phi_i(\boldsymbol{\mathcal{S}}_i) \tag{3.5}$$

$$\boldsymbol{\mathcal{S}}_i = \{\mathcal{X}_1^i, \dots, \mathcal{X}_{s_i}^i\} \tag{3.6}$$

$$Z = \sum_{Val(\boldsymbol{\mathcal{X}})} \prod_{i=1}^{k} \phi_i(\boldsymbol{\mathcal{S}}_i) \ . \tag{3.7}$$

Z is a normalizing constant, also called the partition function. Note that $\boldsymbol{\mathcal{S}}_i \cap \boldsymbol{\mathcal{S}}_j$ doesn't have to be empty for $i \neq j$.

In the example of the multiple coin toss, $Z = 1$ and there are p factors $\phi_i(\boldsymbol{\mathcal{S}}_i) = \mathbb{P}(\mathcal{X}_i) \ \forall i = 1, \dots, p$.



(a) Representation as a hypergraph (b) Representation as a bipartite graph

Figure 3.1: The factor graph for $\mathbb{P}(\boldsymbol{\mathcal{X}}) = \phi_1(\mathcal{X}_1, \mathcal{X}_2)\phi_2(\mathcal{X}_2, \mathcal{X}_3, \mathcal{X}_4)/Z$ can be represented by two different methods.

A generic encoding of a PGM is a **factor graph**. A factor graph is a bipartite graph encoding the structure of a set of factors. There is a bijection between the first set of vertices of this graph and the set of factors Φ, and another bijection between the second set of vertices and the set $\boldsymbol{\mathcal{X}}$ of variables. An edge links a variable $\mathcal{X}_q$ to a factor ϕ_i if and only if $\mathcal{X}_q \in \boldsymbol{\mathcal{S}}_i$. Note that this bipartite graph is equivalent to a hypergraph whose nodes are the variables and with k hyperedges such that hyperedge i links variables $\boldsymbol{\mathcal{S}}_i$. The parameters θ encode the values of the different factors for any configuration of their input variables, either explicitely or by parameterized functions used to compute these values. An illustration of both graphical representations of a factor graph is given in Figure 3.1.

Different probabilistic graphical models encode different sets of independence relationships. Different classes of PGMs can be defined based on the relationships they can graphically represent. The relationships between all the classes mentioned in this section are graphically represented in the Venn diagram of Figure 3.2. All of them will be defined below. Of particular interest for this thesis are Bayesian networks, because they will be used as the target distribution, and Markov trees, because aggregating such models is the topic of this thesis. Other models are mentioned for the interested reader.

Among these classes, Markov random fields (MRF), Bayesian networks (BN) and chordal graphs can encode any probability distribution by using a factorization that encodes no independence relationship. Both Bayesian networks and MRF can encode independence relationships that the other is not able to represent.

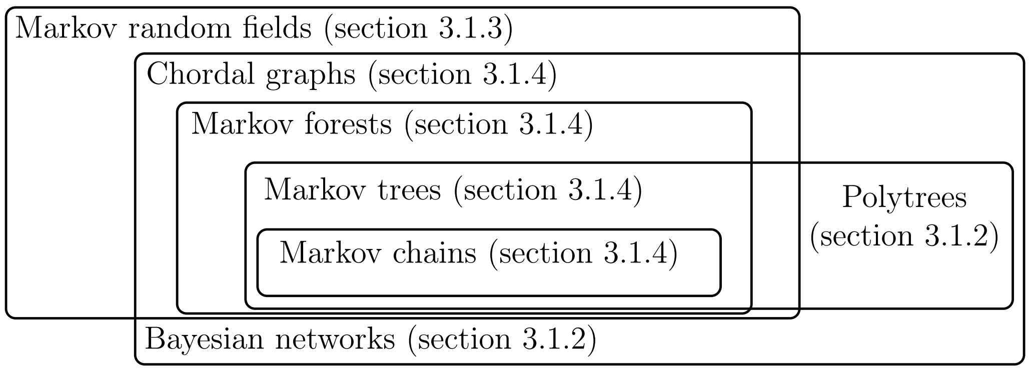

Markov random fields (section 3.1.3)
Chordal graphs (section 3.1.4)
Markov forests (section 3.1.4)
Markov trees (section 3.1.4)
Markov chains (section 3.1.4)
Polytrees (section 3.1.2)
Bayesian networks (section 3.1.2)

Figure 3.2: Different PGMs can encode different sets of conditional independence relationships. This figure shows how different classes of PGMs (described in this chapter) relate to each other with respect to these relationships.
Chordal graphs lie at the intersection of Bayesian networks and Markov random fields. Moreover, Markov chains $\subset$ Markov trees $\subset$ Markov forests $\subset$ chordal graphs, and Markov trees $=$ (Markov forests $\cap$ polytrees).

3.1.2 Bayesian Networks

A **Bayesian network** uses as graphical structure a Directed Acyclic Graph, whose nodes are in a bijection with $\boldsymbol{\mathcal{X}}$. It encodes a joint probability distribution as the product of the marginal distribution of each variable $\mathcal{X}_i$, conditionally to its parents $\boldsymbol{\mathcal{P}}a_{\mathcal{G}}^{\mathcal{X}_i}$ in the graph:

$$\mathbb{P}_{\mathcal{G}}(\boldsymbol{\mathcal{X}}) = \prod_{i=1}^{p} \mathbb{P}_{\mathcal{G}}(\mathcal{X}_i | \boldsymbol{\mathcal{P}}a_{\mathcal{G}}^{\mathcal{X}_i}, \theta) \ . \tag{3.8}$$

The network therefore encodes a factorization of the joint probability distribution $\mathbb{P}(\boldsymbol{\mathcal{X}})$. Note that a probability distribution can always be factorized as

$$\mathbb{P}(\boldsymbol{\mathcal{X}}) = \prod_{i=1}^{p} \mathbb{P}(\mathcal{X}_i | \mathcal{X}_1, \ldots, \mathcal{X}_{i-1}) \ . \tag{3.9}$$

A Bayesian network exploits the relationships between the variables to reduce the number of variables on which the marginal probability distributions are conditioned. A fully connected BN corresponds to the factorization of

Equation 3.9. It does not encode any independence relationship, and can therefore encode any probability distribution.

The parameter set θ of a BN is the union of the parameters of the marginal probability distributions, and I will use the following notation to refer to individual parameters:

$$\theta_{i,x|\mathbf{a}} = \mathbb{P}_{\mathcal{G}}(\mathcal{X}_i = x|\boldsymbol{\mathcal{P}}a_{\mathcal{G}}^{\mathcal{X}_i} = \mathbf{a}, \theta) \ . \tag{3.10}$$

Each factor is normalized, because it is a (marginal) probability distribution. The partition function is therefore always equal to one. In the future, θ may be omitted from Equation 3.8 when its value is easily deducible from the context.

In the remaining part of this thesis, I will concentrate on BN both to manipulate distributions and to construct Markov trees, because the absence of partition function makes these tasks easier.

Reading Independence Relationships in a Bayesian Network

In a Bayesian network, two sets of variables $\boldsymbol{\mathcal{A}}$, $\boldsymbol{\mathcal{B}}$ are independent conditionally to $\boldsymbol{\mathcal{C}}$, $\boldsymbol{\mathcal{A}} \perp \boldsymbol{\mathcal{B}}|\boldsymbol{\mathcal{C}}$, if $\boldsymbol{\mathcal{C}}$ is said to *d-seperate* $\boldsymbol{\mathcal{A}}$ and $\boldsymbol{\mathcal{B}}$. This property is true if and only if every path[1] between any $\mathcal{A} \in \boldsymbol{\mathcal{A}}$ and $\mathcal{B} \in \boldsymbol{\mathcal{B}}$ contains at least one of the following patterns:

- $\mathcal{X}_i \rightarrow \mathcal{C} \rightarrow \mathcal{X}_j$ with $\mathcal{C} \in \boldsymbol{\mathcal{C}}$,
- $\mathcal{X}_i \leftarrow \mathcal{C} \rightarrow \mathcal{X}_j$ with $\mathcal{C} \in \boldsymbol{\mathcal{C}}$,
- $\mathcal{X}_i \leftarrow \mathcal{C} \leftarrow \mathcal{X}_j$ with $\mathcal{C} \in \boldsymbol{\mathcal{C}}$,
- $\mathcal{X}_i \rightarrow \mathcal{X}_k \leftarrow \mathcal{X}_j$ with $\mathcal{X}_k$ (and none of its descendants) $\notin \boldsymbol{\mathcal{C}}$

The pattern $\mathcal{X}_i \rightarrow \mathcal{X}_k \leftarrow \mathcal{X}_j$ is called a v-structure.

On a side note, an arrow in the network does not necessarily indicate a cause $\rightarrow$ effect relationship. Indeed, the three networks represented in Figure 3.1 encode the same independence relationships and can be used to encode the same set of distributions, though their parameters have to be adapted. As a matter of fact, those networks are said to be **equivalent**. When the arrows in a Bayesian network do represent causal relationships, the model is called a **causal Bayesian network**.

[1]A path is a sequence of vertices of the graph, such that an edge of the graph links each variable of the sequence to the next variable in the sequence.

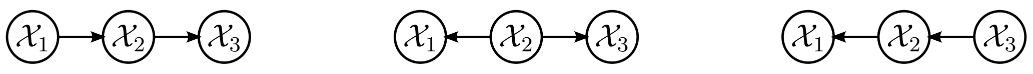
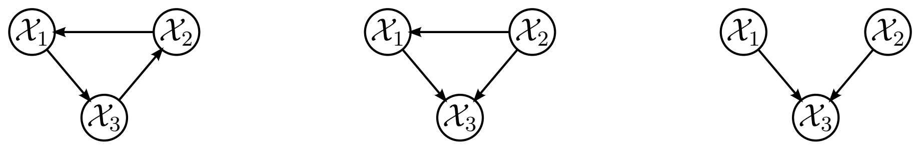


Figure 3.3: These 3 Bayesian networks encode the same set of independence relationships. Only the graphical structure is represented.



(a) This graph cannot define a Bayesian network: it contains a directed cycle.

(b) This graph can define a BN but not a polytree: it contains a cycle (not directed).

(c) This graph can define a polytree: it does not contain any cycle.

Figure 3.4: A Bayesian network does not contain any directed cycle, a polytree does not contain any cycle at all.

Polytrees

Polytrees are a subclass of Bayesian networks. The difference between polytrees and other BN is illustrated in Figure 3.4. Polytrees are characterized by a graph without any cycle, directed or not. By contrast, the class of BNs only prohibits directed cycles.

In addition, polytrees are connected structures. Because of these restrictions, polytrees defined over p variables can only have $p-1$ edges. Therefore, and unlike BNs, polytrees cannot encode any possible distribution.

The class of polytrees contain the class of Markov trees, defined in Section 3.1.4. A Markov tree is a polytree where each variable (except one, the root) has only one parent.

3.1.3 Markov Random Fields

A Markov random field (MRF) or undirected graphical model is a probabilistic graphical model whose graphical structure is an undirected graph. There is a bijection between the nodes of this graph and $\boldsymbol{\mathcal{X}}$. The parametrization

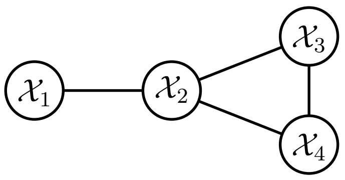


Figure 3.5: This graph can be used as the graphical component of a MRF encoding the distribution $\mathbb{P}(\boldsymbol{\mathcal{X}}) = \frac{1}{Z}\phi_1(\mathcal{X}_1, \mathcal{X}_2)\phi_2(\mathcal{X}_2, \mathcal{X}_3, \mathcal{X}_4)$.

of a MRF is similar to the one of a factor graph, i.e.

$$\mathbb{P}(\boldsymbol{\mathcal{X}}) = \frac{1}{Z}\prod_{i=1}^{k}\phi_i(\boldsymbol{\mathcal{S}}_i) \tag{3.11}$$

$$\boldsymbol{\mathcal{S}}_i = \{\mathcal{X}_1^i, \ldots, \mathcal{X}_{s_i}^i\} \ . \tag{3.12}$$

However the factors are not defined over arbitrary sets of variables, but over maximal cliques (completely connected components) of the undirected graph. There is an edge in the graph between each pair of variables $(\mathcal{X}_q, \mathcal{X}_l) \in \boldsymbol{\mathcal{S}}_i$ for any i.

Therefore, MRFs cannot exploit as many independence relationships as factor graphs. They can however still encode any probability distribution, but may have to use more parameters than a factor graph to do so.

A particular variant of those probabilistic graphical models is the class of **conditional random fields**. They directly encode a conditional probability distribution on a subset of (output) variables $\boldsymbol{\mathcal{C}}$ given another (input) subset $\boldsymbol{\mathcal{X}}$, where $\boldsymbol{\mathcal{C}} \cap \boldsymbol{\mathcal{X}} = \emptyset$. These models are described more precisely in Section 8.1. They are useful when the observed variables $\boldsymbol{\mathcal{X}}$ on which inference will be conditioned is known beforehand and never changes. Indeed, the distribution of those observed variables and their relationships are irrelevant for inference. Conditional random fields therefore do not model nor encode them, saving memory and time.

Reading Independence Relationships in a Markov Random Field

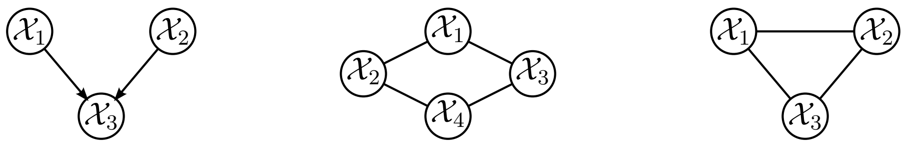
In a MRF, two sets of variables $\boldsymbol{\mathcal{A}}$, $\boldsymbol{\mathcal{B}}$ are independent conditionally to $\boldsymbol{\mathcal{C}}$ ($\boldsymbol{\mathcal{A}} \perp \boldsymbol{\mathcal{B}} | \boldsymbol{\mathcal{C}}$) if every path between any $\mathcal{A} \in \boldsymbol{\mathcal{A}}$ and any $\mathcal{B} \in \boldsymbol{\mathcal{B}}$ contains at least one variable $\mathcal{C} \in \boldsymbol{\mathcal{C}}$.

A MRF can therefore not graphically encode sets of independence relationships where one independence disappears when conditioning on an additional variable, such as e.g. $\mathcal{A} \perp \mathcal{B}$ and $\mathcal{A} \not\perp \mathcal{B}|\mathcal{C}$. Differences between BNs and MRFs are described in the next section.

3.1.4 Chordal Graphs

A Chordal graph is a MRF where each each cycle of $\mathcal{G}$ with a length higher than 3 has a chord. This class constitutes the intersection of Bayesian networks and Markov random fields. The sets of independence relationships a chordal graph can encode in its graphical structure can be represented by both BNs or MRFs. Apart from the structural constraint, chordal graphs also differ from BNs because any BN corresponding to a chordal graph has no v-structures (Equation 3.1.2). To obtain a BN structure corresponding to a chordal graph, the edges of this graph can be oriented randomly, but without introducing any v-structure or any directed cycle.

Figure 3.6 displays three graphs illustrating the difference between BNs, MRFs and chordal graphs in terms of independence relationships. The first contains a v-structure, and can therefore only be encoded as a BN. The second contains no v-structure but a 4 variables loop without a chord, and can therefore only be encoded as a MRF. The last one is a chordal graph, containing no v-structure and no chordless loop of more than 3 variables.



(a) $\mathcal{X}_1 \perp \mathcal{X}_2$ and $\mathcal{X}_1 \not\perp \mathcal{X}_2|\mathcal{X}_3$ can only be encoded by a BN, not a MRF.

(b) $\mathcal{X}_1 \perp \mathcal{X}_4|\mathcal{X}_2, \mathcal{X}_3$ and $\mathcal{X}_2 \perp \mathcal{X}_3|\mathcal{X}_1, \mathcal{X}_4$ can only be encoded by a MRF, not a BN.

(c) Chordal graphs are at the intersection of BNs and MRFs.

Figure 3.6: Chordal graphs, BNs and MRFs cannot encode any set of independence relationships.

Independence relationships can be read visually in chordal graphs as in MRFs. This is also equivalent to reading them as on BNs, because there is no v-structure in the corresponding BNs.

The encoding of the distribution of a chordal graph can be formatted as in Bayesian networks or as in MRFs. The encoding can also be transformed from one of these two formats to the other. A chordal graph can encode any distribution, because a complete graph is also chordal.

Markov Forests

A Markov forest is a chordal graphs whose graphical structure contains no cycle. The graph of a Markov forest can be disconnected, but can contain at most $p - 1$ edges. Alternatively, a Markov forest is a Bayesian network where each variable has at most one parent.

Therefore the associated factors take at most two variables (associated to an edge) as input. Due to these restrictions, a forest cannot model all possible distributions.

Markov Trees

A Markov tree is a Markov forest where the graphical structure is a connected tree (i.e. a graph without cycle) spanning all variables. It has therefore exactly $p - 1$ edges. Markov trees constitute the intersection of Markov forests and polytrees.

The graph of a Markov tree encoded as a MRF can be transformed into the graph of a corresponding Bayesian network by arbitrarily choosing a node as root of the tree and orienting all edges so that they do not point towards the root.

Markov Chains

A Markov chain is a subclass of Markov trees where each variable is connected to at most two variables. The resulting structure is therefore a chain.

3.2 Uses

Such graphical models can be used for several applications, and in this section a few are illustrated. Consider a probabilistic graphical network defining a probability distribution over a human body functioning properly or suffering from different diseases. A few possible nodes, focusing on liver

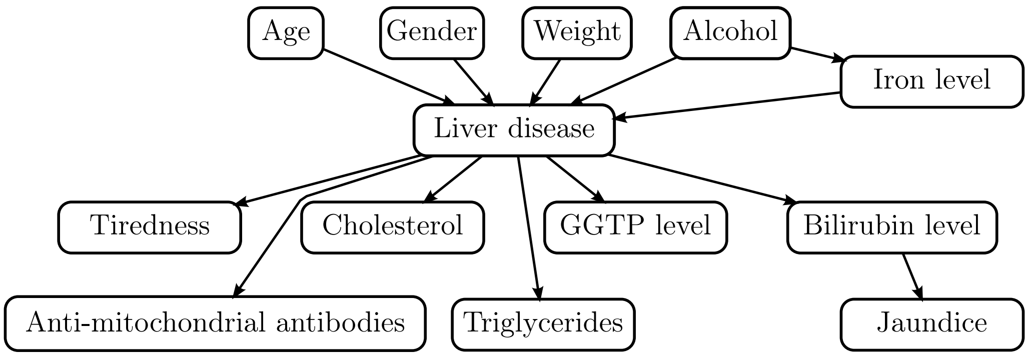


Figure 3.7: This Bayesian network models diseases of the liver (in particular *primary biliary cirrhosis* and *hepatic steatosis*), and several related variables. It may be used to diagnose a disease based on a set of observations, or to select the best observation to perform in order to obtain information about the disease. This example is inspired by [ODW99], and consists in a subgraph of the network developed in this latter article.

diseases, are represented in Figure 3.7. The central variable is the variable associated to the diseases. Variables above it are factors influencing the probability of the disease (e.g. women are 10 times more likely to contract a particular disease than men), while variables below it are consequences of a disease (e.g. anti-mitochondrial antibodies are present in the blood of most people suffering from a certain disease).

3.2.1 Probabilistic Inference

The probability distribution $\mathbb{P}$ encoded by a probabilistic graphical model can be used to answer queries about the underlying problem, a process called inference. Possible queries include the following examples:

- compute the probability of a problem instance, $\mathbb{P}(\boldsymbol{\mathcal{X}} = \mathbf{x})$;
- compute the conditional probability density of a set of variables $\boldsymbol{\mathcal{C}}$ given the observed values $\mathbf{x}'$ of another (possibly empty) set of variables $\boldsymbol{\mathcal{X}}'$, $\mathbb{P}(\boldsymbol{\mathcal{C}}|\boldsymbol{\mathcal{X}}' = \mathbf{x}')$;

- provide the maximum-a-posteriori configuration (MAP) for a set of variables $\boldsymbol{\mathcal{C}}$, or the set of the K most probable configurations, given observed values for another subset of variables $\boldsymbol{\mathcal{X}}'$, a process that can be used for classification;
- provide the most probable explanation configuration (MPE), a MAP inference where $\boldsymbol{\mathcal{C}} = \boldsymbol{\mathcal{X}} \backslash \boldsymbol{\mathcal{X}}'$.

On the medical PGM described above, such queries could be used to estimate the number of sick people in a population, the probability that a given patient requires hospitalization in the future, or to compute the most likely disease given available observations (diagnosis).

3.2.2 Feature Selection

The graphical structure of a model encodes the independence relationships between variables. This structure can therefore be exploited to select the best subset of variables that can be observed to predict another set of variables $\mathcal{C}$, a process called feature selection. More formally, the goal of feature selection is to determine a minimal subset of variable $\boldsymbol{\mathcal{X}}'$ such that

$$\boldsymbol{\mathcal{C}} \perp \boldsymbol{\mathcal{X}} \backslash \{\boldsymbol{\mathcal{X}}', \boldsymbol{\mathcal{C}}\} | \boldsymbol{\mathcal{X}}' \ . \tag{3.13}$$

In the example of Figure 3.7, a feature selection could be performed e.g. to select the observations necessary for a diagnosis.

3.3 Learning

A probabilistic graphical network can be constructed by an expert who specifies the whole network, learned automatically from a set of observations of the problem, or a combination of both, e.g. an expert-defined structure where the parameters are optimized on a learning set. In this thesis I will only consider automatic learning of the whole model, and this section gives a brief introduction to the subject. Common assumptions and techniques for learning Bayesian networks are presented. In addition, the complexity of learning a Bayesian network is discussed.

A bit more formally, I will consider learning a Bayesian network from a set of observations as the following optimization problem. Given a learning set $D = \{\mathbf{x}_{D_1}, \ldots, \mathbf{x}_{D_N}\}$ containing N realizations from a probability distribution $\mathbb{P}$ over a set of p variables $\boldsymbol{\mathcal{X}}$, which is, among a given class of models, the one ($\mathcal{G}$,θ) that minimizes (or maximizes) a given score.

If the goal of the learning procedure is to learn a joint probability distribution over all variables, i.e. generative learning, a usual score to minimize is the Kullback-Leibler divergence of the model learned to the probability distribution defined by the learning set, $\mathbb{P}_D$, or equivalently the log-likelihood of the learning set. Other cost functions may be considered depending on the application, e.g. the expected conditional likelihood for a prediction task or the number of edges recovered for knowledge discovery, if the distribution is encoded by a probabilistic graphical model and if the graph is known.

I will first discuss common assumptions, then the complexity of learning and finally learning techniques. The problem of learning a Bayesian network is usually divided in learning the structure alone and learning the parameters for a given structure. I will thus describe those two steps separately, starting by the parameters, followed by Markov trees structure, with general BN structures coming last.

3.3.1 Hypotheses

Algorithms for learning Bayesian networks usually make various assumptions. They are presented here, and later discussed in the context of this thesis, in Section 3.5.1.

The observations of the learning set are considered to be **independent and identically distributed** (iid). Each observation can be considered as a different random variable, and this hypothesis means the realizations of those different variables are independent from each other, and drawn from the same probability distribution $\mathbb{P}$.

Observability relates to the fact that the values of some variables in $\boldsymbol{\mathcal{X}}$ might be missing in the learning set. Some variables, usually qualified as "latent", might not be measured at all or some values, said to be "missing" might not have been acquired in some observations. Full observability means none of this happens.

Faithfulness signifies that there exists at least one model in the class of candidate models that encodes exactly the conditional independence relationships existing in the data generating probability distribution $\mathbb{P}$.

This last assumption is mostly relevant for learning a Bayesian network structure based on the independence relationships inferred from the learning set. It means that at least one considered model encodes all the independence relationships in $\mathbb{P}$ and no additional relationship. Asymptotically in the number of sample, a true structure can be recovered. On the other hand, in the context of density estimation, any probability distribution can be encoded by a complete Bayesian network, even if this BN is does not exactly encode the independence relationships verified by the distribution.

3.3.2 Complexity of Learning

The number of possible DAGs grows superexponentially with the number of variables p [Rob77]. Learning the structure of a Bayesian network is therefore a complex task, that is usually exponential in the number of variables, even when imposing strong restrictions on the structure. Detailed results regarding the complexity of learning probabilistic graphical models in general and Bayesian networks in particular are provided in Section C.2.

Because of these hardness results, learning algorithms usually rely on heuristics and/or limitations of the space of candidate structures to learn a model, by restricting the resolution of the search space [AW08] or by limiting its range (e.g. by constraining the number of candidate parents or the global structures searched [EG08]). In practice, learning is considered not to be possible over a few thousand variables [Auv02, EG08].

However, Markov trees are one class of models for which structure learning is easy. Indeed, it can be performed by the algorithm proposed by Chow and Liu [CL68], described in Section 3.3.4 and of complexity $\mathcal{O}(p^2 \log p)$.

3.3.3 Estimation of the Parameters of a Bayesian Network

Once a structure $\mathcal{G}$ has been selected based on a learning procedure and/or by an expert, the parameters θ associated to that structure can also be learned automatically from a set of observations D. As a reminder, these parameters correspond to the values of the conditional probabilities defined by the edges of the graph

$$\theta_{i,x_i|\mathbf{a}} = \mathbb{P}_{\mathcal{G}}(\mathcal{X}_i = x_i|\boldsymbol{\mathcal{P}}a_{\mathcal{G}}^{\mathcal{X}_i} = \mathbf{a}) \ . \tag{3.14}$$

The two main methods for computing θ from a fully observable learning set are maximum likelihood and Bayesian parameter estimation.

Maximum Likelihood Estimation

Maximum likelihood estimation (MLE) of the parameter aims to select the value of the parameters maximizing the likelihood of the observed samples:

$$\theta_{ML} = \arg\max_{\theta} \mathbb{P}(D|\mathcal{G},\theta) \ . \tag{3.15}$$

The values of the parameters are obtained by

$$\theta_{i,x_i|\mathbf{a}} = \frac{N_D(\mathbf{a},x_i)}{\sum_{x_i \in Val(\mathcal{X}_i)} N_D(\mathbf{a},x_i)} \triangleq \mathbb{P}_D(x_i|\mathbf{a}) \ , \tag{3.16}$$

the frequency of occurrence of the event $x_i|\mathbf{a}$ in the learning set, where $N_D(\mathbf{a},x_i)$ denotes the number of samples of D where $\boldsymbol{\mathcal{P}}a_{\mathcal{G}}^{\mathcal{X}_i} = \mathbf{a}$ and $\mathcal{X}_i = x_i$.

This can be proven by decomposing the likelihood of the observations

$$\begin{aligned}
\mathbb{P}(D|\mathcal{G},\theta) &= \prod_{j=1}^{N} \mathbb{P}(\mathbf{x}_{D_j}|\mathcal{G},\theta) & (3.17)\\
&= \prod_{j=1}^{N}\prod_{i=1}^{p} \mathbb{P}_{\mathcal{G}}(\mathcal{X}_i = x_{iD_j}|\boldsymbol{\mathcal{P}}a_{\mathcal{G}}^{\mathcal{X}_i} = \mathbf{a}_{D_j},\theta) & (3.18)\\
&= \prod_{j=1}^{N}\prod_{i=1}^{p} \theta_{i,x_{iD_j}|\mathbf{a}_{D_j}} & (3.19)\\
&= \prod_{i=1}^{p} \prod_{\mathbf{a}\in Val(\boldsymbol{\mathcal{P}}a_{\mathcal{G}}^{\mathcal{X}_i})} \prod_{x_i \in Val(\mathcal{X}_i)} (\theta_{i,x_i|\mathbf{a}})^{N_D(\mathbf{a},x_i)} \ , & (3.20)
\end{aligned}$$

and maximizing the logarithm of this quantity. x_{iD_j} represents the value of variable $\mathcal{X}_i$ in the j^th observation of the set D.

Bayesian Parameter Estimation

Bayesian estimation consists in defining a prior distribution over the parameters of interest and computing a posterior distribution through Bayes rule. From that distribution is it common to select the maximum a posteriori value of the parameters, i.e.

$$\theta_{MP} = \arg\max_{\theta} \mathbb{P}(D|\mathcal{G},\theta)\mathbb{P}(\theta|\mathcal{G}) \ , \tag{3.21}$$

or the expected value of the parameters,

$$\theta_E = E\left[\mathbb{P}(D|\mathcal{G},\theta)\mathbb{P}(\theta|\mathcal{G})\right] \ . \tag{3.22}$$

If the prior distribution is a product of Dirichlet distributions, one for each variable and per configuration of the parent variables,

$$\mathbb{P}(\theta|\mathcal{G}) = \prod_{i=1}^{p} \prod_{\mathbf{a}\in Val(\boldsymbol{\mathcal{P}}a_{\mathcal{G}}^{\mathcal{X}_i})} Dirichlet(\alpha_{i,x_i^1|\mathbf{a}}, \ldots, \alpha_{i,x_i^{|\mathcal{X}_i|}|\mathbf{a}}) \tag{3.23}$$

$$\propto \prod_{i=1}^{p} \prod_{\mathbf{a}\in Val(\boldsymbol{\mathcal{P}}a_{\mathcal{G}}^{\mathcal{X}_i})} \prod_{x_i^j\in Val(\mathcal{X}_i)} \left(\theta_{i,x_i^j|\mathbf{a}}\right)^{\alpha_{i,x_i^j|\mathbf{a}}-1} \ , \tag{3.24}$$

it satisfies global

$$\mathbb{P}(\theta|\mathcal{G}) = \prod_{i=1}^{p} \mathbb{P}(\theta_{i,\mathcal{X}_i|\boldsymbol{\mathcal{P}}a_{\mathcal{G}}^{\mathcal{X}_i}}|\mathcal{G}) \ , \tag{3.25}$$

and local parameter independence

$$\mathbb{P}(\theta_{i,\mathcal{X}_i|\boldsymbol{\mathcal{P}}a_{\mathcal{G}}^{\mathcal{X}_i}}|\mathcal{G}) = \prod_{\mathbf{a}\in Val(\boldsymbol{\mathcal{P}}a_{\mathcal{G}}^{\mathcal{X}_i})} \mathbb{P}(\theta_{i,\mathcal{X}_i|\mathbf{a}}|\mathcal{G}) \ . \tag{3.26}$$

In that case the MAP assignment of the parameters is given by

$$\theta_{i,x_i|\mathbf{a}} = \frac{\alpha_{i,x_i|\mathbf{a}} - 1 + N_D(\mathbf{a}, x_i)}{\sum_{x_i^j\in Val(\mathcal{X}_i)} \left[\alpha_{i,x_i^j|\mathbf{a}} - 1 + N_D(\mathbf{a}, x_i^j)\right]} \ , \tag{3.27}$$

and their expected value by

$$\theta_{i,x_i|\mathbf{a}} = \frac{\alpha_{i,x_i|\mathbf{a}} + N_D(\mathbf{a}, x_i)}{\sum_{x_i^j\in Val(\mathcal{X}_i)} \left[\alpha_{i,x_i^j|\mathbf{a}} + N_D(\mathbf{a}, x_i^j)\right]} \ . \tag{3.28}$$

This can be directly verified by incorporating Equations 3.20 and 3.24 into Bayes rule:

$$\mathbb{P}(D|\mathcal{G},\theta)\mathbb{P}(\theta|\mathcal{G}) \propto \prod_{i=1}^{p} \prod_{\mathbf{a}\in Val(\boldsymbol{\mathcal{P}}a_{\mathcal{G}}^{\mathcal{X}_i})} \prod_{x_i^j\in Val(\mathcal{X}_i)} \left(\theta_{i,x_i^j|\mathbf{a}}\right)^{\alpha_{i,x_i^j|\mathbf{a}}-1+N_D(\mathbf{a},x_i^j)} \ , \tag{3.29}$$

which is similar to Equation 3.20.

The hyperparameters α of the prior can be viewed in Equation 3.28 as supplementary observations, that stir the estimation of the parameters towards the prior. The more observations available, the lesser the influence of the prior on the estimated values of the parameters.

The Dirichlet prior is uniform when $\alpha_{i,x_i|\mathbf{a}} = 1\ \forall i, x_i, \mathbf{a}$. The expectation of the parameters computed based on such a prior is sometimes called the "Laplace approximation":

$$\theta_{i,x_i|\mathbf{a}} = \frac{1 + N_D(\mathbf{a}, x_i)}{|Val(\mathcal{X}_i)| + \sum_{x_i^j \in Val(\mathcal{X}_i)} N_D(\mathbf{a}, x_i^j)} \ . \tag{3.30}$$

3.3.4 Learning a Tree Structure: the Chow-Liu Algorithm

Mixtures of Markov trees (formally defined in Section 4.2) considered in this thesis are an extension of Markov trees. The problem of finding the best Markov tree structure given a set of observations, solved by the Chow-Liu algorithm, plays a central role in this thesis, since it is used as a building block in most algorithms building mixtures. Therefore this algorithm is presented in this dedicated section rather than as a subcase of Bayesian network learning. I will relate many novel ideas presented in this manuscript to this algorithm, and this section will serve as reference regarding its structure learning part.

The algorithm for learning a Markov tree structure $\mathcal{T}_{CL}(D)$ maximizing the likelihood of a training set D was introduced by Chow and Liu [CL68]. Maximizing the log-likelihood is equivalent to minimizing the Kullback-Leibler divergence of the model to the empirically observed distribution $\mathbb{P}_D(\mathbf{x}) \triangleq \frac{N_D(\mathbf{x})}{N}$. This is easily shown using a decomposition similar to Equation 2.17, with $M_D = \mathbb{P}_\mathcal{T}$ and $\mathbb{P} = \mathbb{P}_D$:

$$D_{KL}(\mathbb{P}_D, \mathbb{P}_\mathcal{T}) = \sum_{\mathbf{x}} \mathbb{P}_D(\mathbf{x}) \log \frac{\mathbb{P}_D(\mathbf{x})}{\mathbb{P}_\mathcal{T}(\mathbf{x})} \tag{3.31}$$

$$= \underbrace{\sum_{\mathbf{x}} \mathbb{P}_D(\mathbf{x}) \log \mathbb{P}_D(\mathbf{x})}_{\text{no dependence on } \mathcal{T}} - \underbrace{\sum_{\mathbf{x}} \mathbb{P}_D(\mathbf{x}) \log \mathbb{P}_\mathcal{T}(\mathbf{x})}_{\text{logll}_D(\mathcal{T})} \ . \tag{3.32}$$

The structure of interest is the structure of the Markov tree that can maximize the likelihood of the observations using the best parametrization, i.e.

$$\begin{aligned}\mathcal{T}_{CL}(D) &= \arg\max_{\mathcal{T}} \max_{\theta} \log \mathbb{P}(D|\mathcal{T}, \theta) && (3.33)\\ &= \arg\max_{\mathcal{T}} \log \mathbb{P}(D|\mathcal{T}, \theta_{ML}(\mathcal{T}, D)) \ , && (3.34)\end{aligned}$$

since the parameters $\theta_{ML}(\mathcal{T}, D)$ are the maximum likelihood parameters as defined in Equation 3.15,

$$= \arg\max_{\mathcal{T}} \sum_{\mathbf{x}} \mathbb{P}_D(\mathbf{x}) \log \mathbb{P}_{\mathcal{T}}(\mathbf{x}|\theta_{ML}(\mathcal{T}, D)) \ , \quad (3.35)$$

using the formulation of Equation 3.32. The distribution encoded by a tree structure and the corresponding maximum likelihood parameters can be decomposed into

$$\begin{aligned}\mathbb{P}_{\mathcal{T}}(\mathbf{x}|\theta_{ML}(\mathcal{T}, D)) &= \mathbb{P}_{\mathcal{T}}(x_r|\theta_{ML}(\mathcal{T}, D)) \\ &\qquad \prod_{(\mathcal{X}_j, \mathcal{X}_i) \in E(\mathcal{T})} \mathbb{P}_{\mathcal{T}}(x_i|x_j, \theta_{ML}(\mathcal{T}, D)) && (3.36)\\ &= \mathbb{P}_D(x_r) \prod_{(\mathcal{X}_j, \mathcal{X}_i) \in E(\mathcal{T})} \mathbb{P}_D(x_i|x_j) && (3.37)\end{aligned}$$

where $\mathcal{X}_r$ is the root of the tree. The last equation is a consequence of the fact that the maximum likelihood parameters of a given structure are equal to the observed frequency in the learning set (see Equation 3.16).

Therefore, the optimal Markov tree structure is therefore the solution to the following optimization problem:

$$\mathcal{T}_{CL}(D) = \arg\max_{\mathcal{T}} \sum_{(\mathcal{X}_j, \mathcal{X}_i) \in E(\mathcal{T})} I_D(\mathcal{X}_i; \mathcal{X}_j) \ , \quad (3.38)$$

where $E(\mathcal{T})$ is the set of edges in $\mathcal{T}$, constrained to be a tree, and where $I_D(\mathcal{X}_i; \mathcal{X}_j)$ (resp. $H_D(\mathcal{X}_i)$, used in the proof) is the maximum likelihood estimate of the mutual information among variables $\mathcal{X}_i$ and $\mathcal{X}_j$ (resp. entropy of $\mathcal{X}_i$) computed from the dataset D.

Proof.

$$\mathcal{T}_{CL}(D) = \arg\max_{\mathcal{T}} \mathbb{P}(D|\mathcal{T}, \theta_{ML}(\mathcal{T}, D)) \tag{3.39}$$

$$= \arg\max_{\mathcal{T}} \sum_{\mathbf{x}} \mathbb{P}_D(\mathbf{x}) \log \left[\mathbb{P}_D(x_r) \prod_{(\mathcal{X}_j,\mathcal{X}_i)\in E(\mathcal{T})} \mathbb{P}_D(x_i|x_j) \right] \tag{3.40}$$

$$= \arg\max_{\mathcal{T}} \sum_{\mathbf{x}} \mathbb{P}_D(\mathbf{x}) \log \left[\mathbb{P}_D(x_r) \prod_{(\mathcal{X}_j,\mathcal{X}_i)\in E(\mathcal{T})} \frac{\mathbb{P}_D(x_i, x_j)}{\mathbb{P}_D(x_j)} \frac{\mathbb{P}_D(x_i)}{\mathbb{P}_D(x_i)} \right] \tag{3.41}$$

$$= \arg\max_{\mathcal{T}} \sum_{\mathbf{x}} \mathbb{P}_D(\mathbf{x}) \log \left[\prod_{i=1}^{p} \mathbb{P}_D(x_i) \prod_{(\mathcal{X}_j,\mathcal{X}_i)\in E(\mathcal{T})} \frac{\mathbb{P}_D(x_i, x_j)}{\mathbb{P}_D(x_j)\mathbb{P}_D(x_i)} \right] \tag{3.42}$$

$$\begin{aligned} &= \arg\max_{\mathcal{T}} \left[\sum_{\mathbf{x}} \mathbb{P}_D(\mathbf{x}) \sum_{i=1}^{p} \log \mathbb{P}_D(x_i) \right. \\ &\qquad \left. + \sum_{\mathbf{x}} \mathbb{P}_D(\mathbf{x}) \sum_{(\mathcal{X}_j,\mathcal{X}_i)\in E(\mathcal{T})} \log \frac{\mathbb{P}_D(x_i, x_j)}{\mathbb{P}_D(x_j)\mathbb{P}_D(x_i)} \right] \end{aligned} \tag{3.43}$$

$$\begin{aligned} &= \arg\max_{\mathcal{T}} \left[\sum_{i=1}^{p} \sum_{\mathbf{x}} \mathbb{P}_D(\mathbf{x}) \log \mathbb{P}_D(x_i) \right. \\ &\qquad \left. + \sum_{(\mathcal{X}_j,\mathcal{X}_i)\in E(\mathcal{T})} \sum_{\mathbf{x}} \mathbb{P}_D(\mathbf{x}) \log \frac{\mathbb{P}_D(x_i, x_j)}{\mathbb{P}_D(x_j)\mathbb{P}_D(x_i)} \right] \end{aligned} \tag{3.44}$$

$$\begin{aligned} &= \arg\max_{\mathcal{T}} \left[\sum_{i=1}^{p} \sum_{x_i \in Val(\mathcal{X}_i)} \mathbb{P}_D(x_i) \log \mathbb{P}_D(x_i) \right. \\ &\qquad \left. + \sum_{(\mathcal{X}_j,\mathcal{X}_i)\in E(\mathcal{T})} \sum_{\substack{x_i \in Val(\mathcal{X}_i) \\ x_j \in Val(\mathcal{X}_j)}} \mathbb{P}_D(x_i, x_j) \log \frac{\mathbb{P}_D(x_i, x_j)}{\mathbb{P}_D(x_j)\mathbb{P}_D(x_i)} \right] \end{aligned} \tag{3.45}$$

$$= \arg\max_{\mathcal{T}} \left[-\sum_{i=1}^{p} H_D(\mathcal{X}_i) + \sum_{(\mathcal{X}_j,\mathcal{X}_i)\in E(\mathcal{T})} I_D(\mathcal{X}_i; \mathcal{X}_j) \right] \qquad \square$$

Finding one solution to Equation 3.38 decomposes into two steps:

1. the computation of the $p \times p$ (symmetric) matrix of pairwise mutual informations (MI) between variables, which requires $\mathcal{O}(p^2N)$ computations;

2. the use of the MI matrix as an edge-weight matrix to build a maximum weight spanning tree (MWST). Possible algorithms include [Kru56], used here and of $\mathcal{O}(p^2 \log p)$ algorithmic complexity, or the faster [Cha00] whose execution times are nearly linear[2] in the number of edges of non zero weight, i.e. in our context $\mathcal{O}(p^2\alpha(p^2, p))$.

The Chow-Liu algorithm thus has essentially a time and space complexity of $\mathcal{O}(p^2 \log p)$ in terms of the number p of variables.

Algorithm 3.1 Chow-Liu (CL) tree [CL68]

```
**Input:** $\boldsymbol{\mathcal{X}}$; $D$
$MI = [0]_{p \times p}$
**for** $i_1 = 1 \to p-1$ **and** $i_2 = i_1 + 1 \to p$ **do**
  $MI[i_1, i_2] = MI[i_2, i_1] = I_D(\mathcal{X}_{i_1}; \mathcal{X}_{i_2})$
**end for**
$\mathcal{T}_{CL} = \text{MWST}(MI)$ {MWST computation, done here by Algorithm 3.2}
**return** $\mathcal{T}_{CL}$.
```

[2] $\alpha(m, n)$ is the inverse Ackermann function and grows very slowly. It is defined for any $m, n > 0$ as

$$\alpha(m, n) = \min\{i \geq 1 : A(i, 4\lceil m/n \rceil) > \log n\} \quad ,$$

where $A(i, j)$ is an Ackermann's function defined for any $i, j \in N^+$ as

$$\begin{cases} A(0, j) = 2j, & \forall j \geq 0 \\ A(i, 0) = 0, \qquad \text{and} \qquad A(i, 1) = 2 & \forall i \geq 1 \\ A(i, j) = A(i-1, A(i, j-1)), & \forall i \geq 1, j \geq 2 \ . \end{cases}$$

Kruskal Algorithm

A popular algorithm to solve the maximum weight spanning tree problem in the Chow-Liu algorithm is the Kruskal algorithm [Kru56]. This procedure is described in Algorithm 3.2 and illustrated in Figure 3.8 for 6 nodes.

Algorithm 3.2 Kruskal MWST algorithm [Kru56]

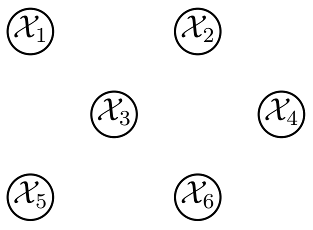
```
Input: symmetric matrix $M$ of positive weights
$\mathcal{T}$ = empty graph of size row($M$)
repeat
  $i, j$ = indices of max (non-diagonal) element in $M$
  if nodes $i$ and $j$ are not connected in $\mathcal{T}$ then
    $\mathcal{T} = \mathcal{T} \cup (i, j)$
  end if
  $M[i, j] = 0$
until all (non-diagonal) elements of $M$ are 0
return $\mathcal{T}$.
```

The Kruskal algorithm receives as input a matrix of the weights of candidate edges. It first constructs an empty graph (Figure 3.8a). Then it iteratively considers the addition of edges to this graph, one edge at a time. These edges are considered by decreasing weight (Figures 3.8b,3.8c,3.8d). However an edge is included in the graph only if it does not create a cycle with edges already present in the graph (Figure 3.8d). When the resulting model is a connected tree ($p - 1$ edges for p nodes), or when all candidate edges have been considered, the procedure stops (Figure 3.8e).

3.3.5 Learning General Bayesian Network Structures

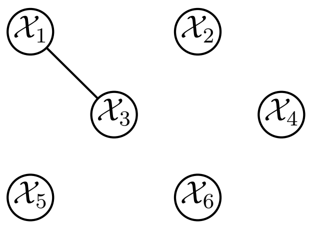
Learning the good structure(s) among the class of candidate models can be done using three main approaches.

- The *constraint-based approach* consists in extracting from the observational data a set of conditional independence relationships (statements $\mathcal{S}_i \perp \mathcal{S}_j | \mathcal{S}_k$, where $\mathcal{S}_i, \mathcal{S}_j, \mathcal{S}_k$ are disjoint subsets of variables), and searching for a structure that best matches those constraints.



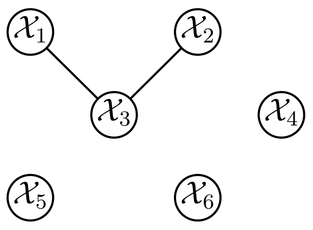
	$\mathcal{X}_1$	$\mathcal{X}_2$	$\mathcal{X}_3$	$\mathcal{X}_4$	$\mathcal{X}_5$	$\mathcal{X}_6$
$\mathcal{X}_1$		21	25	14	3	15
$\mathcal{X}_2$			23	17	5	8
$\mathcal{X}_3$				11	13	18
$\mathcal{X}_4$					7	19
$\mathcal{X}_5$						4

(a) Initial situation: empty graph and symmetric matrix of edge-weights.



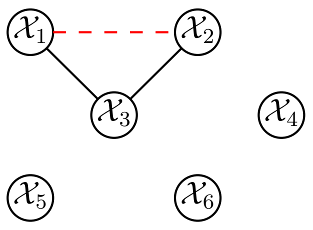
	$\mathcal{X}_1$	$\mathcal{X}_2$	$\mathcal{X}_3$	$\mathcal{X}_4$	$\mathcal{X}_5$	$\mathcal{X}_6$
$\mathcal{X}_1$		21	25	14	3	15
$\mathcal{X}_2$			23	17	5	8
$\mathcal{X}_3$				11	13	18
$\mathcal{X}_4$					7	19
$\mathcal{X}_5$						4

(b) The edge associated to the highest value in the matrix is added.



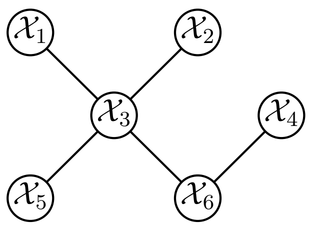
	$\mathcal{X}_1$	$\mathcal{X}_2$	$\mathcal{X}_3$	$\mathcal{X}_4$	$\mathcal{X}_5$	$\mathcal{X}_6$
$\mathcal{X}_1$		21	25	14	3	15
$\mathcal{X}_2$			23	17	5	8
$\mathcal{X}_3$				11	13	18
$\mathcal{X}_4$					7	19
$\mathcal{X}_5$						4

(c) Edges are considered by decreasing order of the associated values.



	$\mathcal{X}_1$	$\mathcal{X}_2$	$\mathcal{X}_3$	$\mathcal{X}_4$	$\mathcal{X}_5$	$\mathcal{X}_6$
$\mathcal{X}_1$		21	25	14	3	15
$\mathcal{X}_2$			23	17	5	8
$\mathcal{X}_3$				11	13	18
$\mathcal{X}_4$					7	19
$\mathcal{X}_5$						4

(d) However, edges that would create cycles are discarded.



	$\mathcal{X}_1$	$\mathcal{X}_2$	$\mathcal{X}_3$	$\mathcal{X}_4$	$\mathcal{X}_5$	$\mathcal{X}_6$
$\mathcal{X}_1$		21	25	14	3	15
$\mathcal{X}_2$			23	17	5	8
$\mathcal{X}_3$				11	13	18
$\mathcal{X}_4$					7	19
$\mathcal{X}_5$						4

(e) When the graph is connected, the procedure stops.

Figure 3.8: The Kruskal algorithm for computing a maximum-weight spanning tree.

- In the *score-based approach*, a numerical criterion (maximum likelihood, BIC, AIC...) is defined over the set of candidate structures, and learning amounts to selecting, among them, the one that maximizes this score with respect to the data set.
- The *model averaging approach* considers the set of all possible structures rather than identifying a single best one, and averages predictions from those structures in accordance with the goal of the learning procedure. Taking into account all possible structures is rarely possible, and approximations must thus be employed.

These three classes of methods are briefly reviewed here, although the latter will only be discussed here as a tool for working on a single graphical structure. Approaches where several structures are involved are the main topic of this thesis and will be further described in Chapter 4. Hybrid methods combining several approaches have also been developed, such as [TBA06]. These hybrid methods are not described here.

Constraint-based Approach

This approach consists in searching for the graph $\mathcal{G}$ that best matches the conditional independence relationships inferred from the learning set, called the constraints. The algorithms of this class must answer two problems: inferring those relationships from the learning set, and selecting a network based on them.

Evaluating conditional independence relationships based on observational data D has been largely studied in statistics through hypothesis testing. As a brief reminder of on that topic, H_0, the null hypothesis, here independence, is evaluated based on a chosen risk of false rejection ρ and $\text{stat}(D)$, a statistics chosen in accordance with the hypothesis. H_0 is rejected if

$$\text{stat}(D) > t_\rho \tag{3.46}$$

$$t_\rho \triangleq \arg_t \mathbb{P}(\text{stat}(D) > t | H_0 \text{ is true}) = \rho \ . \tag{3.47}$$

t_ρ is a threshold function of ρ and $\text{stat}(D)$. ρ is the probability to incorrectly reject H_0 if it is true. A typical value is 0.05.

For evaluating an independence relationship $H_0 = \boldsymbol{\mathcal{A}} \perp \boldsymbol{\mathcal{B}} | \boldsymbol{\mathcal{C}}$ over discrete variables, the χ^2 statistics or the maximum likelihood estimation of the mutual information are usual choices for stat(D). These two statistics are respectively computed as follows [NWL$^+$07]:

$$\chi^2 = \sum_{\mathbf{a},\mathbf{b},\mathbf{c}} \frac{\left[N_D(\mathbf{a},\mathbf{b},\mathbf{c}) - \dfrac{N_D(\mathbf{a},\mathbf{c})N_D(\mathbf{b},\mathbf{c})}{N_D(\mathbf{c})}\right]^2}{\dfrac{N_D(\mathbf{a},\mathbf{c})N_D(\mathbf{b},\mathbf{c})}{N_D(\mathbf{c})}} \tag{3.48}$$

$$I_D(\boldsymbol{\mathcal{A}};\boldsymbol{\mathcal{B}}|\boldsymbol{\mathcal{C}}) = \frac{1}{N}\sum_{\mathbf{a},\mathbf{b},\mathbf{c}} N_D(\mathbf{a},\mathbf{b},\mathbf{c}) \log_2 \frac{N_D(\mathbf{c})N_D(\mathbf{a},\mathbf{b},\mathbf{c})}{N_D(\mathbf{a},\mathbf{c})N_D(\mathbf{b},\mathbf{c})} \ . \tag{3.49}$$

For the maximum likelihood estimate of the mutual information, the quantity $2N(\ln 2)I_D(\boldsymbol{\mathcal{A}};\boldsymbol{\mathcal{B}}|\boldsymbol{\mathcal{C}})$ is asymptotically χ-square distributed under independence, with a degree of freedom equal to $(|Val(\boldsymbol{\mathcal{A}})|-1)(|Val(\boldsymbol{\mathcal{B}})|-1)|Val(\boldsymbol{\mathcal{C}})|$.

The SGS algorithm [SGS93], described in Algorithm 3.3 is one of the early algorithms exploiting the results of those hypothesis tests to construct a Bayesian network structure from a set of observations, and its description will serve as an illustration of this class of methods.

Like most constraint-based algorithms, it assumes the existence of an oracle capable of assessing any conditional independence relationships on the set of input variables $\boldsymbol{\mathcal{X}}$. It first constructs an undirected graph, the skeleton, based on the independence relationships, then detects the v-structures of the network and finally propagates the orientation of the edges. The algorithm outputs a graph with both directed and undirected edges, but the edges that remain undirected can be oriented arbitrarily, provided they do not create v-structures or a directed cycle.

This algorithm requires the evaluation of a large number of independence relationships between variables. More recent algorithms try to limit the assessment of independence relationships, by using clever search strategies or by limiting the cardinality of the subsets of variables inspected and the in-degree of each node in the candidate structure [KF09].

Score-based Approaches

In the *score-based approach*, a numerical criterion is defined over the set of candidate structures, and learning can be defined as selecting, among all DAGs, the one that maximizes this score with respect to the learning

Algorithm 3.3 SGS algorithm [SGS93]

```
**Input:** $\boldsymbol{\mathcal{X}}$; an oracle for conditional independence relationships
$S = \emptyset$ {The skeleton}
**for** $\mathcal{X}_i, \mathcal{X}_j \in \boldsymbol{\mathcal{X}}, j > i$ **do**
  **if** $\nexists\ \boldsymbol{\mathcal{Z}} \in \boldsymbol{\mathcal{X}} \backslash \{\mathcal{X}_i, \mathcal{X}_j\} : \mathcal{X}_i \perp \mathcal{X}_j | \boldsymbol{\mathcal{Z}}$ **then**
    $S = S \cup \{(\mathcal{X}_i, \mathcal{X}_j), (\mathcal{X}_j, \mathcal{X}_i)\}$ {Add undirected edge $\mathcal{X}_i - \mathcal{X}_j$}
  **end if**
**end for**
**for** $\mathcal{X}_i, \mathcal{X}_j, \mathcal{X}_k \in \boldsymbol{\mathcal{X}} : (\mathcal{X}_i - \mathcal{X}_j - \mathcal{X}_k$ **and** $\mathcal{X}_i \nleftrightarrow \mathcal{X}_j)$ in $S$ **do**
  **if** $\nexists\ \boldsymbol{\mathcal{Z}} \in \boldsymbol{\mathcal{X}} \backslash \{\mathcal{X}_i, \mathcal{X}_j, \mathcal{X}_k\} : \mathcal{X}_i \not\perp \mathcal{X}_k | \mathcal{X}_j \cup \boldsymbol{\mathcal{Z}}$ **then**
    $S = S \backslash \{(\mathcal{X}_j, \mathcal{X}_i), (\mathcal{X}_j, \mathcal{X}_k)\}$ {Create v-structure $\mathcal{X}_i \to \mathcal{X}_j \leftarrow \mathcal{X}_k$}
  **end if**
**end for**
**repeat** {Propagate edge orientation}
  **if** $S : \mathcal{X}_i \to \mathcal{X}_j - \mathcal{X}_k$ **and** $S : \mathcal{X}_i \nleftrightarrow \mathcal{X}_k$ **then**
    $S = S \backslash (\mathcal{X}_k, \mathcal{X}_j)$ {orient $\mathcal{X}_k \leftarrow \mathcal{X}_j$}
  **end if**
  **if** $S : (\mathcal{X}_i - \mathcal{X}_j$ **and** there is a directed path from $\mathcal{X}_i$ to $\mathcal{X}_j)$ **then**
    $S = S \backslash (\mathcal{X}_j, \mathcal{X}_i)$ {orient $\mathcal{X}_j \leftarrow \mathcal{X}_i$}
  **end if**
**until** No more edges can be oriented
**return** $\mathcal{G} = (\boldsymbol{\mathcal{X}}, S)$
```

set. Since there are a superexponential number of possible structures, considering them all explicitely is not feasible. A good search strategy must also be chosen in addition to the score. A few general remarks about those heuristics and scores will be given, and two scores (out of many) will be presented.

Such heuristics are usually standard optimization procedures such as greedy hill-climbing, simulated annealing or tabu search. In this context, they typically modify the current structure iteratively by applying modification operators such as edge addition, reversal or removal to create structures in a neighborhood of the original. These new structures are then compared to the current one through their scores, and a new one is selected according to the optimization procedure used.

Such a procedure requires evaluating the scores of numerous structures at each iteration. To limit the complexity of computing them, *decomposable* *scores* are usually used. These are scores that can be written as

$$\text{score}(\mathcal{G}, D) = \sum_{i=1}^{p} f(\mathcal{X}_i, \boldsymbol{\mathcal{P}}a_{\mathcal{G}}^{\mathcal{X}_i}) \ . \tag{3.50}$$

In a decomposable score, a modification to $\mathcal{G}$ only changes the values of the terms of the sum that are related to the affected variables. When only one edge is affected (as with the operators listed above), at most two terms are modified, and all others are left unchanged. This property can be exploited by the heuristic to speed-up the computation of the scores of the structures close to the current structure. Moreover, once a new structure is selected and another iteration of the optimization algorithm begins, the score gain provided by any operator applied to variables unconcerned by the accepted change need not be computed again. In other words, only the terms related to the change must be recomputed. Both points lead to significant gains in computational time.

The **likelihood score** of the set of observations is such a decomposable score. The likelihood is computed for a given structure based on the values of the parameters maximizing this likelihood. Using a decomposition similar to the one leading to the Chow-Liu algorithm (Section 3.3.4, replace x_j by $\boldsymbol{\mathcal{P}}a_{\mathcal{G}}^{\mathcal{X}_i}$), the log-likelihood score can be shown to equal

$$\text{logll}_D(\mathcal{G}) = \max_{\theta} \sum_{j=1}^{N} \log \mathbb{P}(\mathbf{x}_{D_j} | \mathcal{G}, \theta) \tag{3.51}$$

$$= N \sum_{i=1}^{p} \left[I_D(\mathcal{X}_i; \boldsymbol{\mathcal{P}}a_{\mathcal{G}}^{\mathcal{X}_i}) - H_D(\mathcal{X}_i) \right] \ . \tag{3.52}$$

This score will always favor the addition of edges to the model structure, and will cause overfitting. Therefore the likelihood score is often modified to include a penalty on the size (number of edges or number of parameters) of the model. This principle is called regularization, and its application to Markov trees is discussed in Section 5.1.1.

Another possibility to limit overfitting is the **Bayesian Dirichlet (BD) score** [CH92]. As opposed to the likelihood score, the BD score averages over all possible values of the parameters for the structure considered. The Bayesian score requires the definition of a prior distribution $\mathbb{P}(\mathcal{G})$ on the structures and $\mathbb{P}(\theta|\mathcal{G})$ of the parameters given the structure. Given these priors, the Bayes rule states that the posterior probability of a structure conditionally on a given learning set is

$$\mathbb{P}(\mathcal{G}|D) = \frac{\mathbb{P}(D|\mathcal{G})\mathbb{P}(\mathcal{G})}{\mathbb{P}(D)}. \tag{3.53}$$

The constant denominator is discarded and the BDscore is the logarithm of the remaining quantity:

$$BD(\mathcal{G}, D) = \log \mathbb{P}(D|\mathcal{G}) + \mathbb{P}(\mathcal{G}) \ . \tag{3.54}$$

$\mathbb{P}(D|\mathcal{G})$ is computed by averaging over all possible parameters:

$$\mathbb{P}(D|\mathcal{G}) = \int_\theta \mathbb{P}(D|\mathcal{G}, \theta)\mathbb{P}(\theta|\mathcal{G})d\theta \ . \tag{3.55}$$

For categorical variables, the Dirichlet distribution is the conjugate prior. Using $\left\{\alpha_{i,x_i|\mathbf{a}}\right\}_{x_i \in Val(\mathcal{X}_i)}$ to denote the hyperparameter of the prior associated to $\mathcal{X} = x|\boldsymbol{\mathcal{P}}a(\mathcal{X}) = \mathbf{a}$, the probability of the learning set given a structure can be computed by

$$\mathbb{P}(D|\mathcal{G}) = \prod_{i=1}^{p} \prod_{\mathbf{a} \in Val(\boldsymbol{\mathcal{P}}a_{\mathcal{G}}^{\mathcal{X}_i})} \frac{\Gamma(\alpha_{\mathcal{X}_i|\mathbf{a}})}{\Gamma(\alpha_{\mathcal{X}_i|\mathbf{a}} + N_D(\mathbf{a}))} \prod_{x_i \in Val(\mathcal{X}_i)} \frac{\Gamma(\alpha_{i,x_i|\mathbf{a}} + N_D(\mathbf{a}, x_i))}{\Gamma(\alpha_{i,x_i|\mathbf{a}})} \tag{3.56}$$

$$\alpha_{\mathcal{X}_i|\mathbf{a}} = \sum_{x_i \in Val(\mathcal{X}_i)} \alpha_{i,x_i|\mathbf{a}} \ , \tag{3.57}$$

on the condition that the prior distribution satisfies global and local parameter independence. $\Gamma(x)$ is the gamma function $\int_0^\infty t^{x-1}e^{-t}dt$, for $x \in \mathbb{R}^+$.

Model Averaging Approaches

The two classes of mixtures presented in Section 2.6 have been exploited to build mixtures of probabilistic graphical models, but the discussion of those models (averages of probability densities) is postponed to Section 4.1. This section only covers methods that combine several models to construct a single output structure.

Rather than identifying a single best structure, mixtures are used to identify the probability that any graphical feature (such as e.g. an edge

$\mathcal{X} \rightarrow \mathcal{Y}$) is present in the real structure. This allows uncertainty in the model selection procedure. The probability of a feature f can be established through a Bayesian approach:

$$\mathbb{P}(f) = \sum_{\mathcal{G}} \delta(f \in \mathcal{G}) \mathbb{P}(\mathcal{G}|D) \ , \tag{3.58}$$

where $\delta(f \in \mathcal{G})$ is equal to 1 if the feature f is present in the structure $\mathcal{G}$ and 0 otherwise. This can be done for all possible structures with a complexity exponential in p [KS04]. More efficient algorithms exist when the topological ordering[3] of the variables is known [Bun91].

When taking into account all possible structures is not possible, approximations can be employed.

One of these strategies is the bootstrap aggregation approach (section 2.6.2), and has been used for structure learning of graphical models, e.g. by considering the frequency of occurrence of interesting graphical features among the structures derived from bootstrap replicas [FGW99], and also to improve score-based structure learning by incorporating the bootstrap procedure in the computation of the score [Eli11].

Another approach consists in constructing a Markov Chain Monte Carlo over the structure search space, then to use this chain to sample good structures. The sequence $\boldsymbol{\mathcal{S}}$ of structures generated by the chain is then used as an approximation of Equation 3.58.

$$\mathbb{P}(f) = \sum_{\mathcal{G} \in \boldsymbol{\mathcal{S}}} \delta(f \in \mathcal{G}) \ . \tag{3.59}$$

[MYA95] was the first to develop such an approach. One refinement, proposed in [FK03], is to perform a MCMC chain only over the topological ordering of the variables, and to use the result of [Bun91] to average over all networks for a given ordering.

Alternative stochastic methods have also been considered, such as in [MR94].

[3]A topological ordering of the variables is an ordering such that, for every pair of variables $\mathcal{X}_i$,$\mathcal{X}_j$ ($i \neq j$), an edge $\mathcal{X}_i \rightarrow \mathcal{X}_j$ can be present in $\mathcal{G}$ if and only if $\mathcal{X}_i$ precedes $\mathcal{X}_j$ in the ordering. It therefore reduces the number of possible DAG structures, and facilitates the development of an efficient search strategy.

3.4 Inference

Once a Bayesian network has been specified or constructed, it can be exploited to answer queries about the distribution, a process generically called inference. While the contributions contained in this thesis are mostly related to learning PGMs that can be used for inference, but not on performing inference, a basic knowledge of inference may in my opinion help to understand the motivation of this work and appreciate its interest. Moreover, inference is often necessary in subroutines of the learning procedure. The purpose of this section is to provide the required background about these algorithms.

The complexity of inference is first discussed in Section 3.4.1. Remember the complexity of inference is one of the motivations of using simple graphical models. Belief propagation in Markov trees is then explained in Section 3.4.2. This algorithm computes inferences of the form $\mathbb{P}_{\mathcal{G}}(\boldsymbol{\mathcal{O}}|\boldsymbol{\mathcal{I}} = \mathbf{i})$, the probability distribution of a subset of variables $\boldsymbol{\mathcal{O}}$ conditionally on observed values $\mathbf{i}$ of a distinct subset of variables $\boldsymbol{\mathcal{I}}$. Finally, key ideas behind a few methods for inference on any Bayesian network are presented in Section 3.4.3.

3.4.1 Complexity of Inference

This section gives a brief summary about the complexity of inference. The different inference operations defined in Section 3.2.1 are in most cases a hard problem, and scale badly with the size of the problem. Without going into the details, answering an inference query is generally a problem requiring a worse than polynomial time in the number of variables. Note that more detailed theoretical results about the complexity of inference in probabilistic graphical models are reported in Appendix C.3.

As regards inference complexity in probabilistic graphical models, an important quantity is the maximum tree-width of the structure. The **maximum tree-width** of a graph is defined as the number of nodes in the largest clique of the moralized[4] and triangularized[5] graph, minus 1. A clique is a completely connected subgraph. Algorithms such as the junction tree algorithm scales as $p \exp(k)$ for inference on Bayesian networks of bounded

[4]The parents of any node are married, i.e. linked by an edge.

[5]Edges are added to make the graph chordal.

tree-width[6] k, see e.g. [LS98], and inference in Markov trees (tree-width=1) is of linear complexity in the number of variables [KP83].

Today, inference in PGMs remains an active research topic. Contests are regularly held to evaluate the best algorithms available. To the best of my knowledge, the last contest of this kind is the Probabilistic Inference Challenge (PIC2011)[7]. Many learning methods also specifically target low tree-width structures in order to limit inference complexity, see e.g. [BJ01, EG08, FNP99, SCG09].

3.4.2 Inference in Polytrees

Belief propagation is the algorithm used to perform probabilistic inference on Markov trees and on polytrees. It was proposed in [KP83] and computes all marginals $\mathbb{P}(\mathcal{X}_i|\mathbf{i})$ for an evidence $\mathbf{i}$. The description of this algorithm proposed here is inspired by [NWL$^+$07]. As pointed out in the previous section, the complexity of this inference operation is linear in p for Markov trees. Although I will mostly consider Markov trees in this thesis, the algorithm is described for the more general class of polytrees. It easily simplifies to the case of Markov trees.

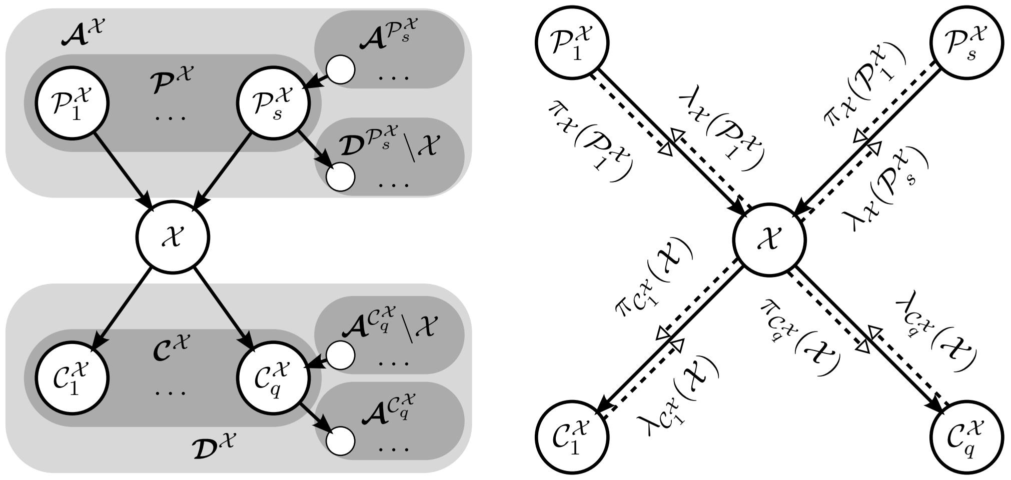
The algorithm relies on the transmission of information between nodes of the graph and along the edges. This information is encoded in messages sent from one node to another. Inference is complete when two messages have been transmitted (and processed) along every edge of the network: one upwards and one downwards.

I will first discuss the case of an unobserved variable $\mathcal{X}$ that is neither a leaf nor a root of the polytree. The variables can be partitioned into three subsets with respect to $\mathcal{X}$: $\mathcal{X}$, $\boldsymbol{\mathcal{A}}^{\mathcal{X}}$ (the ancestors of $\mathcal{X}$) and $\boldsymbol{\mathcal{D}}^{\mathcal{X}}$ (the descendants of $\mathcal{X}$). Figure 3.9 represents all quantities mentioned in this section on a polytree structure. The evidence (here denoted as $\mathbf{i}$) can be divided in correspondance with the partition of the variables:

$$\mathbf{i} = \mathbf{i}_{\boldsymbol{\mathcal{A}}^{\mathcal{X}}} \cup \mathbf{i}_{\boldsymbol{\mathcal{D}}^{\mathcal{X}}} \quad . \tag{3.60}$$

[6]This tree-width is actually the maximal tree-width of the moralized and chordalized graph of the Bayesian network.

[7]`http://www.cs.huji.ac.il/project/PASCAL/`


$\mathcal{A}^{\mathcal{X}}$
$\mathcal{P}^{\mathcal{X}}$
$\mathcal{P}_1^{\mathcal{X}}$
$\mathcal{P}_s^{\mathcal{X}}$
$\mathcal{A}^{\mathcal{P}_s^{\mathcal{X}}}$
$\mathcal{D}^{\mathcal{P}_s^{\mathcal{X}}}\backslash\mathcal{X}$
$\mathcal{X}$
$\mathcal{C}^{\mathcal{X}}$
$\mathcal{C}_1^{\mathcal{X}}$
$\mathcal{C}_q^{\mathcal{X}}$
$\mathcal{A}^{\mathcal{C}_q^{\mathcal{X}}}\backslash\mathcal{X}$
$\mathcal{A}^{\mathcal{C}_q^{\mathcal{X}}}$
$\mathcal{D}^{\mathcal{X}}$
$\pi_{\mathcal{X}}(\mathcal{P}_1^{\mathcal{X}})$
$\lambda_{\mathcal{X}}(\mathcal{P}_1^{\mathcal{X}})$
$\pi_{\mathcal{X}}(\mathcal{P}_1^{\mathcal{X}})$
$\lambda_{\mathcal{X}}(\mathcal{P}_s^{\mathcal{X}})$
$\pi_{\mathcal{C}_1^{\mathcal{X}}}(\mathcal{X})$
$\lambda_{\mathcal{C}_1^{\mathcal{X}}}(\mathcal{X})$
$\pi_{\mathcal{C}_q^{\mathcal{X}}}(\mathcal{X})$
$\lambda_{\mathcal{C}_q^{\mathcal{X}}}(\mathcal{X})$

(a) Variables $\boldsymbol{\mathcal{X}}\backslash\mathcal{X}$ can be decomposed with respect to $\mathcal{X}$ into its ancestors $\boldsymbol{\mathcal{A}}^{\mathcal{X}}$ and its descendants $\boldsymbol{\mathcal{D}}^{\mathcal{X}}$. In each of these two sets, the variables directly linked to $\mathcal{X}$ are respectively called its parents $\boldsymbol{\mathcal{P}}^{\mathcal{X}}$ and children $\boldsymbol{\mathcal{C}}^{\mathcal{X}}$.

(b) Each variable $\mathcal{X}$ receives $|\boldsymbol{\mathcal{P}}^{\mathcal{X}}| + |\boldsymbol{\mathcal{C}}^{\mathcal{X}}|$ messages, one from each neighbor (parent or child). It also sends a message to each of these variables. Before a message can be computed or sent, the messages from the other neighbors must be received.

Figure 3.9: Variables and messages involved in belief propagation in a polytree.

$\mathbb{P}(\mathcal{X}|\mathbf{i})$ can be factorized according to Bayes rule as follows, where the normalization constant is omitted (normalization is usually carried out at the end of all computations):

$$\mathbb{P}(\mathcal{X}|\mathbf{i}) \propto \mathbb{P}(\mathcal{X}|\mathbf{i}_{\boldsymbol{\mathcal{A}}^{\mathcal{X}}})\mathbb{P}(\mathbf{i}_{\boldsymbol{\mathcal{D}}^{\mathcal{X}}}|\mathcal{X}, \mathbf{i}_{\boldsymbol{\mathcal{A}}^{\mathcal{X}}}) \tag{3.61}$$

$$= \underbrace{\mathbb{P}(\mathcal{X}|\mathbf{i}_{\boldsymbol{\mathcal{A}}^{\mathcal{X}}})}_{\pi(\mathcal{X})}\underbrace{\mathbb{P}(\mathbf{i}_{\boldsymbol{\mathcal{D}}^{\mathcal{X}}}|\mathcal{X})}_{\lambda(\mathcal{X})} \tag{3.62}$$

since $\mathbf{i}_{\boldsymbol{\mathcal{D}}^{\mathcal{X}}} \perp \mathbf{i}_{\boldsymbol{\mathcal{A}}^{\mathcal{X}}}|\mathcal{X}$ due to the tree structure: any path between $\mathbf{i}_{\boldsymbol{\mathcal{D}}^{\mathcal{X}}}$ and $\mathbf{i}_{\boldsymbol{\mathcal{A}}^{\mathcal{X}}}$ must contain $\mathcal{X}$. In this latter equation, $\mathbb{P}(\mathcal{X}|\mathbf{i}_{\boldsymbol{\mathcal{A}}^{\mathcal{X}}}) = \pi(\mathcal{X})$ plays the role of a prior distribution while $\mathbb{P}(\mathbf{i}_{\boldsymbol{\mathcal{D}}^{\mathcal{X}}}|\mathcal{X}) = \lambda(\mathcal{X})$ is a likelihood. I now focus on those two terms, one at a time, starting with $\lambda(\mathcal{X})$.

$\mathbf{i}_{\boldsymbol{\mathcal{D}}^{\mathcal{X}}}$ can be written as

$$\mathbf{i}_{\boldsymbol{\mathcal{D}}^{\mathcal{X}}} = \bigcup_{\mathcal{C} \in \boldsymbol{\mathcal{C}}^{\mathcal{X}}} \mathbf{i}_{\mathcal{C},\boldsymbol{\mathcal{D}}^{\mathcal{X}}} \ , \tag{3.63}$$

where $\boldsymbol{\mathcal{C}}^{\mathcal{X}}$ denotes the children of $\mathcal{X}$ and $\mathbf{i}_{\mathcal{C},\boldsymbol{\mathcal{D}}^{\mathcal{X}}}$ the observations related to variables from which the path to $\mathcal{X}$ contains $\mathcal{C}$. Therefore, $\lambda(\mathcal{X})$ decomposes into a product of terms, where each term is related to a different child of $\mathcal{X}$:

$$\begin{aligned}
\lambda(\mathcal{X}) &= \mathbb{P}(\mathbf{i}_{\boldsymbol{\mathcal{D}}^{\mathcal{X}}}|\mathcal{X}) && (3.64)\\
&= \mathbb{P}(\bigcup_{\mathcal{C} \in \boldsymbol{\mathcal{C}}^{\mathcal{X}}} \mathbf{i}_{\mathcal{C},\boldsymbol{\mathcal{D}}^{\mathcal{X}}}|\mathcal{X}) && (3.65)\\
&= \prod_{q=1}^{|\boldsymbol{\mathcal{C}}^{\mathcal{X}}|} \mathbb{P}(\mathbf{i}_{\mathcal{C}_q,\boldsymbol{\mathcal{D}}^{\mathcal{X}}}|\mathbf{i}_{\mathcal{C}_1,\boldsymbol{\mathcal{D}}^{\mathcal{X}}}, \dots \mathbf{i}_{\mathcal{C}_{q-1},\boldsymbol{\mathcal{D}}^{\mathcal{X}}}, \mathcal{X}) && (3.66)\\
&= \prod_{q=1}^{|\boldsymbol{\mathcal{C}}^{\mathcal{X}}|} \mathbb{P}(\mathbf{i}_{\mathcal{C}_q,\boldsymbol{\mathcal{D}}^{\mathcal{X}}}|\mathcal{X}) \ , && (3.67)
\end{aligned}$$

because $\mathbf{i}_{\mathcal{C}_q,\boldsymbol{\mathcal{D}}^{\mathcal{X}}} \perp \mathbf{i}_{\mathcal{C}_r,\boldsymbol{\mathcal{D}}^{\mathcal{X}}}|\mathcal{X} \ \forall q \neq r$. $\mathbb{P}(\mathbf{i}_{\mathcal{C},\boldsymbol{\mathcal{D}}^{\mathcal{X}}}|\mathcal{X})$ will be denoted by $\lambda_{\mathcal{C}}(\mathcal{X})$ from now on. $\lambda_{\mathcal{C}}(\mathcal{X})$ can be seen as a message going up from $\mathcal{C}$ to $\mathcal{X}$.

$\mathbf{i}_{\mathcal{C},\boldsymbol{\mathcal{D}}^{\mathcal{X}}}$ can be decomposed with respect to $\mathcal{C}$ as $\mathbf{i}$ was with respect to $\mathcal{X}$ in Equation 3.60:

$$\mathbf{i}_{\mathcal{C},\boldsymbol{\mathcal{D}}^{\mathcal{X}}} = \mathbf{i}_{\boldsymbol{\mathcal{A}}^{\mathcal{C}} \backslash \mathcal{X}} \cup \mathbf{i}_{\boldsymbol{\mathcal{D}}^{\mathcal{C}}} \ , \tag{3.68}$$

where the notation $\mathbf{i}_{\boldsymbol{\mathcal{A}}^{\mathcal{C}} \backslash \mathcal{X}}$ stresses that evidences related to variables in $\boldsymbol{\mathcal{A}}^{\mathcal{C}}$ but whose path to $\mathcal{C}$ contains $\mathcal{X}$ are not considered. Based on this decomposition,

$$\begin{aligned}
\lambda_{\mathcal{C}}(\mathcal{X}) &= \mathbb{P}(\mathbf{i}_{\mathcal{C},\boldsymbol{\mathcal{D}}^{\mathcal{X}}}|\mathcal{X}) && (3.69)\\
&= \mathbb{P}(\mathbf{i}_{\boldsymbol{\mathcal{A}}^{\mathcal{C}} \backslash \mathcal{X}}, \mathbf{i}_{\boldsymbol{\mathcal{D}}^{\mathcal{C}}}|\mathcal{X}) && (3.70)\\
&= \sum_{\substack{c \in Val(\mathcal{C}) \\ \mathbf{e} \in Val(\boldsymbol{\mathcal{P}}^{\mathcal{C}} \backslash \mathcal{X})}} \mathbb{P}(\mathbf{i}_{\boldsymbol{\mathcal{A}}^{\mathcal{C}} \backslash \mathcal{X}}, \mathbf{i}_{\boldsymbol{\mathcal{D}}^{\mathcal{C}}}|c, \mathbf{e}, \mathcal{X}) \mathbb{P}(c, \mathbf{e}|\mathcal{X}) \ . && (3.71)
\end{aligned}$$

This equation can be simplified by exploiting the independence relationships encoded by the tree structure:

$$\lambda_{\mathcal{C}}(\mathcal{X}) = \sum_{c,\mathbf{e}} \mathbb{P}(\mathbf{i}_{\boldsymbol{\mathcal{D}}^{\mathcal{C}}}|c) \mathbb{P}(\mathbf{i}_{\boldsymbol{\mathcal{A}}^{\mathcal{C}} \backslash \mathcal{X}}|\mathbf{e}) \mathbb{P}(c, \mathbf{e}|\mathcal{X}) \ , \tag{3.72}$$

and by the application of Bayes rule:

$$= \sum_{c,\mathbf{e}} \mathbb{P}(\mathbf{i}_{\boldsymbol{\mathcal{D}}^{\mathcal{C}}}|c) \frac{\mathbb{P}(\mathbf{e}|\mathbf{i}_{\boldsymbol{\mathcal{A}}^{\mathcal{C}}\setminus\mathcal{X}})\mathbb{P}(\mathbf{i}_{\boldsymbol{\mathcal{A}}^{\mathcal{C}}\setminus\mathcal{X}})}{\mathbb{P}(\mathbf{e})} \mathbb{P}(c|\mathcal{X},\mathbf{e})\mathbb{P}(\mathbf{e}|\mathcal{X}) \ . \tag{3.73}$$

Moreover, $\mathbb{P}(\mathbf{e}|\mathcal{X}) = \mathbb{P}(\mathbf{e})$ (independence) and $\mathbb{P}(\mathbf{i}_{\boldsymbol{\mathcal{A}}^{\mathcal{C}}\setminus\mathcal{X}})$ is a constant. Consequently,

$$\lambda_{\mathcal{C}}(\mathcal{X}) \propto \sum_{c} \mathbb{P}(\mathbf{i}_{\boldsymbol{\mathcal{D}}^{\mathcal{C}}}|c) \sum_{\mathbf{e}} \mathbb{P}(\mathbf{e}|\mathbf{i}_{\boldsymbol{\mathcal{A}}^{\mathcal{C}}\setminus\mathcal{X}})\mathbb{P}(c|\mathcal{X},\mathbf{e}) \tag{3.74}$$

$$= \sum_{c} \underbrace{\mathbb{P}(\mathbf{i}_{\boldsymbol{\mathcal{D}}^{\mathcal{C}}}|c)}_{\lambda(c)} \sum_{\mathbf{e}} \mathbb{P}(c|\mathcal{X},\mathbf{e}) \prod_{r=1,\ldots,|\boldsymbol{\mathcal{P}}^{\mathcal{C}}\setminus\mathcal{X}|} \mathbb{P}(e_r|\mathbf{i}_{\mathcal{P}_r^{\mathcal{C}},\boldsymbol{\mathcal{A}}^{\mathcal{C}}}) \ , \tag{3.75}$$

where $\mathbf{i}_{\mathcal{P}_r^{\mathcal{C}},\boldsymbol{\mathcal{A}}^{\mathcal{C}}}$ are the evidences related to variables of $\boldsymbol{\mathcal{A}}^{\mathcal{C}}$ whose path to $\mathcal{C}$ contains $\mathcal{P}_r^{\mathcal{C}}$.

$\lambda_{\mathcal{C}}(\mathcal{X})$ is therefore a function of the local parameters of the model, $\mathbb{P}(c|\mathcal{X},\mathbf{e})$, of $\mathbb{P}(\mathbf{i}_{\boldsymbol{\mathcal{D}}^{\mathcal{C}}}|c) = \lambda(c)$, and of $\mathbb{P}(\mathbf{e}|\mathbf{i}_{\boldsymbol{\mathcal{A}}^{\mathcal{C}}\setminus\mathcal{X}})$, which is close to $\pi(\mathcal{C})$ (see Equation 3.62), and actually follows a decomposition similar to $\pi(\mathcal{C})$.

The corresponding decomposition of $\pi(\mathcal{X})$ can be obtained by a reasoning similar to the one used for $\lambda(\mathcal{X})$. Going faster than for $\lambda(\mathcal{X})$, this decomposition yields:

$$\pi(\mathcal{X}) = \mathbb{P}(\mathcal{X}|\mathbf{i}_{\boldsymbol{\mathcal{A}}^{\mathcal{X}}}) \tag{3.76}$$

$$= \sum_{\mathbf{p}\in Val(\boldsymbol{\mathcal{P}}^{\mathcal{X}})} \mathbb{P}(\mathcal{X}|\mathbf{p})\mathbb{P}(\mathbf{p}| \bigcup_{\mathcal{P}\in\boldsymbol{\mathcal{P}}^{\mathcal{X}}} \mathbf{i}_{\mathcal{P},\boldsymbol{\mathcal{A}}^{\mathcal{X}}}) \tag{3.77}$$

$$= \sum_{\mathbf{p}\in Val(\boldsymbol{\mathcal{P}}^{\mathcal{X}})} \mathbb{P}(\mathcal{X}|\mathbf{p}) \prod_{s=1,\ldots,|\boldsymbol{\mathcal{P}}^{\mathcal{X}}|} \mathbb{P}(p_s|\mathbf{i}_{\mathcal{P}_s,\boldsymbol{\mathcal{A}}^{\mathcal{X}}}) \ . \tag{3.78}$$

$\mathbb{P}(p|\mathbf{i}_{\mathcal{P},\boldsymbol{\mathcal{A}}^{\mathcal{X}}})$ can be viewed as a message, as information about the evidences $\mathbf{i}$ reaching $\mathcal{X}$ through its parent $\mathcal{P}$, and will be denoted by $\pi_{\mathcal{X}}(p)$.

$$\pi_{\mathcal{X}}(p) = \mathbb{P}(p|\mathbf{i}_{\mathcal{P},\boldsymbol{\mathcal{A}}^{\mathcal{X}}}) \tag{3.79}$$

$$= \mathbb{P}(p|\mathbf{i}_{\boldsymbol{\mathcal{A}}^{\mathcal{P}}} \cup \mathbf{i}_{\boldsymbol{\mathcal{D}}^{\mathcal{P}}\setminus\mathcal{X}}) \tag{3.80}$$

$$\propto \mathbb{P}(\mathbf{i}_{\boldsymbol{\mathcal{D}}^{\mathcal{P}}\setminus\mathcal{X}}|p)\mathbb{P}(p|\mathbf{i}_{\boldsymbol{\mathcal{A}}^{\mathcal{P}}}) \tag{3.81}$$

$$\mathbb{P}(\mathbf{i}_{\boldsymbol{\mathcal{D}}^{\mathcal{P}}\setminus\mathcal{X}}|p) = \prod_{\mathcal{F}\in\boldsymbol{\mathcal{C}}^{\mathcal{P}}\setminus\mathcal{X}} \mathbb{P}(\mathbf{i}_{\mathcal{F},\boldsymbol{\mathcal{D}}^{\mathcal{P}}}|p) \ . \tag{3.82}$$

Regrouping all equations developed in this section results in the following summary:

$$\mathbb{P}(\mathcal{X}|\mathbf{i}) \propto \mathbb{P}(\mathcal{X}|\mathbf{i}_{\boldsymbol{\mathcal{A}}^\mathcal{X}})\mathbb{P}(\mathbf{i}_{\boldsymbol{\mathcal{D}}^\mathcal{X}}|\mathcal{X}) \tag{3.83}$$

$$= \pi(\mathcal{X})\lambda(\mathcal{X}) \tag{3.84}$$

$$\lambda(\mathcal{X}) \propto \prod_{\mathcal{C}\in\boldsymbol{\mathcal{C}}^\mathcal{X}} \lambda_\mathcal{C}(\mathcal{X}) \tag{3.85}$$

$$\pi(\mathcal{X}) \propto \sum_{\mathbf{p}\in Val(\boldsymbol{\mathcal{P}}^\mathcal{X})} \mathbb{P}(\mathcal{X}|\mathbf{p}) \prod_{s=1,\ldots,|\boldsymbol{\mathcal{P}}^\mathcal{X}|} \pi_\mathcal{X}(p_s) \tag{3.86}$$

$$\lambda_\mathcal{C}(\mathcal{X}) \propto \sum_{c\in Val(\mathcal{C})} \lambda(c) \sum_{\mathbf{e}\in Val(\boldsymbol{\mathcal{P}}^\mathcal{C}\backslash X)} \mathbb{P}(c|\mathcal{X},\mathbf{e}) \prod_{r=1,\ldots,|\boldsymbol{\mathcal{P}}^\mathcal{C}\backslash\mathcal{X}|} \pi_\mathcal{C}(e_r) \tag{3.87}$$

$$\pi_\mathcal{X}(p) \propto \pi(\mathcal{P}) \prod_{\mathcal{F}\in\boldsymbol{\mathcal{C}}^\mathcal{P}\backslash\mathcal{X}} \lambda_\mathcal{F}(p) \qquad \forall p \in Val(\mathcal{P}) \ . \tag{3.88}$$

A few observations can be made about these equations.

- Each $\lambda_\mathcal{C}(\mathcal{X})$ and $\{\pi_\mathcal{X}(p)\}_{p\in Val(\mathcal{P})}$ is a message to $\mathcal{X}$ from one of its neighbor. The latter are following the orientation of an edge, the former are going in the opposite direction. For $\mathcal{X}$, incomming and outgoing messages are represented in Figure 3.9b.

- To compute the message to one neighbour $\mathcal{Y}$, $\mathcal{X}$ needs the messages from all its neighbours, except $\mathcal{Y}$. Therefore, it is always possible to propagate these messages in a polytree until convergence (shown by recurrence: a polytree has at least one node with only one neighbour).

- A variable $\mathcal{X}$ must receive the message of all its neighbours in order to compute $\mathbb{P}(\mathcal{X})$.

The propagation of all the messages can be done by only considering each variable at most twice: the first time when all its neighbours but one have sent their messages to it, to compute the message for the last neighbour; the second time when this neighbour sends his message, to compute and send the messages for all the other neighbours.

The last point to discuss is the case of observed variables. An easy way to include them in the model is the addition of a fictive child to any observed variable $\mathcal{Y}$, such that the λ message from that new variable equals $\delta(y; i_\mathcal{Y})$, i.e. 1 for $\mathcal{Y} = i_\mathcal{Y}$, the evidence value for the variable $\mathcal{Y}$ and 0 otherwise.

3.4.3 Inference on Bayesian Networks

Inference over unconstrained Bayesian networks is unfortunately not as efficient as on a polytree structure. However, since polytree models are often too restrictive to properly model a distribution, many algorithms have been developed for exact or approximate inference on Bayesian networks. Presenting them all and even specifying some of them is outside the scope of this chapter, but the key ideas behind some of them will be described in a few lines to give a glimpse on this field.

Exact Inference

Variable elimination and the junction tree algorithm are two main algorithms for performing probabilistic inference exactly on Bayesian networks.

Variable elimination iteratively removes variables from the network until there remain only variables of interest to the inference query. When a variable is eliminated, parameters of other variables are modified so that the probability distribution does not change. In fact, eliminating a variable implies a marginalization of the joint probability distribution. Consider as an illustration the probability distribution defined on 3 binary variables and encoded by the Markov chain

$$\mathbb{P}(\boldsymbol{\mathcal{X}}) = \mathbb{P}(\mathcal{A})\mathbb{P}(\mathcal{B}|\mathcal{A})\mathbb{P}(\mathcal{C}|\mathcal{B}) \ . \tag{3.89}$$

Computing $\mathbb{P}(\mathcal{C})$ means marginalizing out $\mathcal{B}$ and $\mathcal{C}$:

$$\begin{aligned}
\mathbb{P}(\mathcal{C}) &= \sum_{\substack{a \in Val(\mathcal{A}) \\ b \in Val(\mathcal{B})}} \mathbb{P}(a)\mathbb{P}(b|a)\mathbb{P}(\mathcal{C}|b) &\quad (3.90)\\
&= \sum_{b \in Val(\mathcal{B})} \mathbb{P}(\mathcal{C}|b) \underbrace{\sum_{a \in Val(\mathcal{A})} \mathbb{P}(a)\mathbb{P}(b|a) \ .}_{\mathbb{P}(b)} &\quad (3.91)
\end{aligned}$$

Marginalizing both variables at the same time (first equation) necessitates 3 summations and 8 multiplications, while eliminating one variable at a time to define a new network (second equation) requires only 3 summations and 6 multiplications. The improvement is more significant when there are more than 3 variables. Selecting a good order of elimination can however be challenging, and the complexity of this algorithm heavily depends on a good elimination order.

The **junction tree** algorithm is a two-step process. In the first, the Bayesian network is transformed into a junction or clique tree, and in the second step inference is carried out on this tree structure, using an algorithm similar to the one developed for inference in polytrees (Section 3.4.2). A clique tree decomposition (see also Figure 3.10) of a Bayesian network is a tree structured graphical model where

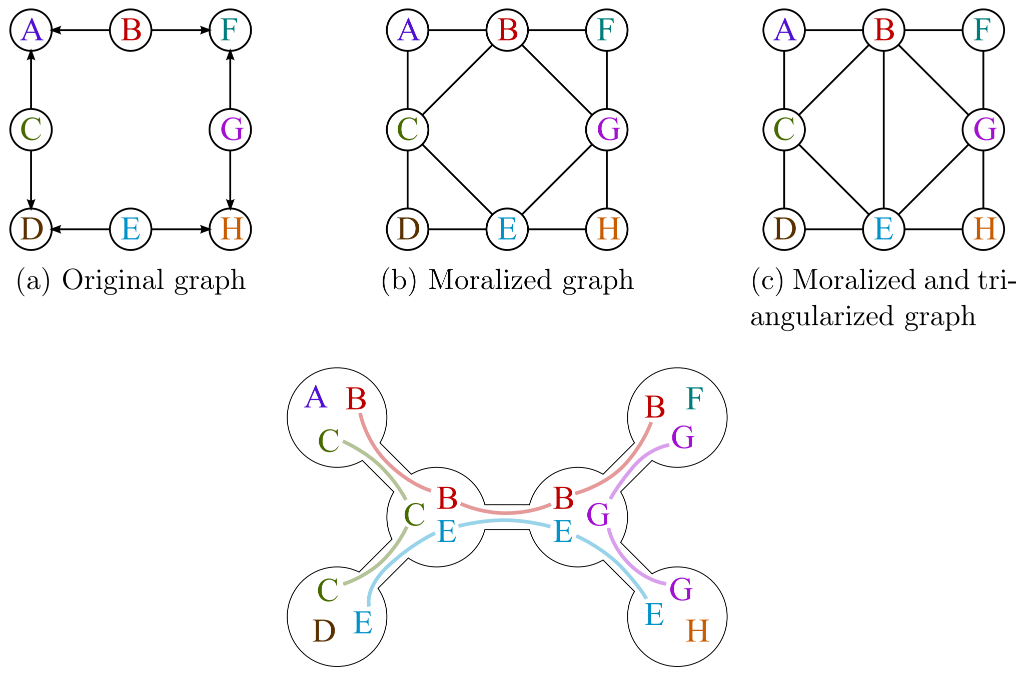
- each node is associated to a subset $\boldsymbol{\mathcal{S}}_i$ of $\mathcal{X}$,
- $\bigcup_i \boldsymbol{\mathcal{S}}_i = \mathcal{X}$,
- for every edge $\mathcal{X}, \mathcal{Y}$ in the moralised (the parents of a node are "married", i.e. joined by an edge) and chordalized (edges are added so that each cycle of at least 4 nodes has a chord), there is at least one $\boldsymbol{\mathcal{S}}_i \ni \mathcal{X}, \mathcal{Y}$,
- if $\mathcal{X} \in \boldsymbol{\mathcal{S}}_i, \boldsymbol{\mathcal{S}}_j$, then $\mathcal{X}$ also belongs to every $\boldsymbol{\mathcal{S}}_k$ on the (unique) path between $\boldsymbol{\mathcal{S}}_i$ and $\boldsymbol{\mathcal{S}}_j$ (running intersection property).

Approximate Inference

Many different approximate algorithms for inference have been developed over the years. I will present two important classes of such algorithms: stochastic sampling and loopy belief propagation.

Stochastic sampling algorithms perform inference by generating samples (or instances) from the network, and estimating the quantity of interest by its frequency of occurrence in these observations. Many different generating mechanisms are possible. Complete instances over all variables can be considered, but instances restricted to subsets of variables are also possible. Many generation mechanisms exist: the observations can be generated randomly, sampled from the probability encoded by the network, constructed deterministically etc [Dar09].

Loopy belief propagation is, in a nutshell, the application of the belief propagation algorithm (described in Section 3.4.2) to a general Bayesian network. Similar algorithms were found to produce good approximations on several networks. This led to the development of many methods inspired by this algorithm. Since the structure is no longer a polytree, convergence is not guaranteed. Indeed, the variables in a loop cannot receive the message



(a) Original graph (b) Moralized graph (c) Moralized and triangularized graph

(d) Clique tree decomposition

Figure 3.10: A Bayesian network is transformed into a clique tree by moralizing and triangularizing its graph. The colored lines highlight the running intersection property (original image from wikipedia by David Eppstein).

of all their neighboring variables but one. Therefore, messages are computed iteratively by initializing them all and propagated until convergence. Many different strategies have been developed around the idea of propagating messages, such as approximated messages or propagation on approximated structures (such as e.g. a tree).

3.5 Outlook

This chapter so far has provided the background about the probabilistic graphical models framework necessary to understand the models manipulated in this thesis and its main contributions. Different models have first been defined and their utility illustrated. Learning and inference with those models have then been introduced, with a particular emphasis on algorithmic complexity and on Markov trees. The latter was discussed in depth because it is the main model used in this thesis, and the former because it is the main motivation of our focus on simple PGMs and on Markov trees in particular.

The main topic of this thesis is the construction of mixtures of PGMs (described in detail in Chapter 4) for learning and inference on many variables. This principle relies on the generation of several models, and the aggregation of the answer provided by each model to a given query. In order for this process to be tractable, both learning and answering a query must also be tractable on a term of the mixture. The complexity of an operation on a mixture of m models is the sum of the complexity of this operation on each term of the model, plus an eventual overhead. Since learning and inference is mostly driven by the treewidth of the model and is intractable (NP-hard) even for polytrees (though not for all inference operations) but respectively of logquadratic and linear complexity for Markov trees, these models seem to be the most sensible choice to use in a mixture. This subject is further discussed in Section 4.1, once mixtures have been introduced.

Apart from this important point, this section discusses common PGM learning hypothesis with respect to this research and specifies which method is used in the algorithm developed for learning the parameters of the models.

3.5.1 Learning Hypotheses

This thesis focuses on learning mixtures of Markov trees for reducing the variance, a source of error defined in Section 2.4. Therefore the hypothesies underlying the algorithms developed in this manuscript differ slightly from the standard ones presented in Section 3.3.1.

First, the hypothesis that the observations are iid and fully observable are considered valid.

More importantly, I do not assume that the target distribution can be exactly encoded by the class of candidate models. As mentioned in Section 3.1, Markov trees are rather restrictive models and cannot encode all distributions exactly, they simply do not have enough parameters. However this assumption is mostly used to establish asymptotic results when the number of available observations tends to infinity. So, it is not particularly relevant for situations with many variables and few samples, the focus of this thesis. Moreover, it is particularly interesting to evaluate the methods developed here on distributions encoded by a Bayesian network whose structure is not a tree, since those distributions are closer to real situations.

Constraining the model is also a way to limit overfitting (see Section 2.4 for an introduction to the bias-variance trade-off in machine learning), and one of the reasons to select simple models, the other being tractable algorithmic complexity.

3.5.2 Parameter Learning

Section 3.3.3 presented two methods for learning the parameters of a Bayesian network for a given structure. In this thesis and unless otherwise stated the parameters of the models are estimated by the Bayesian method, and in particular the Laplace approximation. As a short reminder, it is based on a uniform Dirichlet prior, which is tantamount to adding one sample for each possible configuration of the subset of variables related to those parameters:

$$\theta_{i,x|\mathbf{a}} = \frac{1 + N_D(\mathbf{a}, x)}{|Val(\mathcal{X}_i)| + \sum_{x \in Val(\mathcal{X}_i)} N_D(\mathbf{a}, x)} \quad . \tag{3.92}$$

Using such a prior ensures that probabilities encoded by the resulting Bayesian network are all non-zero. Therefore the two measures of accuracy used to evaluate this model, the Kullback-Leibler divergence and the log-likelihood (presented in Section 2.3.2) are always finite.

Chapter 4

Mixtures of Markov Trees

The goal of this chapter is to give a first introduction to the learning algorithms that will be developed in the second part of this thesis.

I build on the material presented in the two previous chapters, namely: (i) mixture models, and in particular the perturb and combine framework (Chapter 2), and (ii) probabilistic graphical models, and in particular Markov trees, a class of Bayesian networks (Chapter 3).

The learning algorithms developed will apply the perturb and combine principle in order to construct mixtures of Markov trees, with the goal of reducing the variance with respect to a single Markov tree constructed by the Chow-Liu algorithm. Even though a Markov tree is a simple model with respect to the class of Bayesian networks, it may still suffer from a large variance in a high-dimensional setting, as will be shown in later experiments, and as noted e.g. in [LXHG+11, TAW11].

Markov trees obtained by a randomized algorithm will be aggregated using an arithmetic mean (a choice discussed in Section 4.3.1). The mixtures considered in this thesis are therefore of the form

$$\mathbb{P}_{\boldsymbol{\mathcal{T}}}(\boldsymbol{\mathcal{X}}) = \sum_{j=1}^{m} \lambda_j \mathbb{P}_{\mathcal{T}_j}(\boldsymbol{\mathcal{X}}) \tag{4.1}$$

$$\sum_{j=1}^{m} \lambda_j = 1 \tag{4.2}$$

$$\lambda_j \in [0,1] \qquad \forall j \ , \tag{4.3}$$

where m is the number of Markov trees in the mixture, $\mathcal{T}_j$ denotes a Markov tree of the mixture, and λ_j is the weight associated to this tree. The probability density encoded by the model is thus a convex combination of the densities encoded by the individual Markov trees (the weights are positive and sum to 1).

I have chosen to work with Markov trees, because inference can be performed efficiently in this class of models. Indeed, to exploit a mixture model, an inference operation must be performed on every tree of the mixture. The high number of variables p makes algorithmic complexity a key design element, both for the learning and the inference part. In addition, I target algorithms that behave well on small learning sets, because those are the norm for high-dimensional problems. On the other hand, in this work I put far less emphasis on good asymptotic (large sample) properties.

The rest of this chapter is organized as follows. Section 4.1 provides a brief overview of the literature on mixture models based on different subclasses of probabilistic graphical models, while Section 4.2 provides a survey of already published work in the context of mixtures of Markov trees. Section 4.3 aims at presenting my approach: I first explain why geometric averaging was not considered as an alternative or in replacement of arithmetic averaging, and then describe the general meta-algorithm used in the second part of this thesis to construct mixtures of Markov trees. Finally, I provide an overview of the subsequent chapters of the thesis.

4.1 An Overview of Mixtures of PGMs

It is possible to use other models than Markov trees to construct mixtures, although inference might not be tractable. This section discusses those alternative classes of models.

In a general setting, the models used should be able to represent any distribution while allowing learning and inference to be algorithmically tractable. In high-dimensional learning however, simple models are likely to be more useful than complex ones:

- the number of parameters (edges and local distributions) should be limited, to keep models simple and avoid overfitting;
- the tree-width of the models considered should be small, or other constrains enforced, to allow learning and inference algorithms to be tractable (see Section 3.4.2).

The first requirement might not be necessary in a perturb and combine setting. Although learning an overly complex model may translate into a high variance, this variance is likely to be reduced by model averaging.

In the second point, the complexity of inference may be more important than the complexity of learning. Indeed, the perturb and combine framework introduces randomization in learning algorithms, and this randomization can be selected so as to lower the complexity of the algorithm. Exact inference, on the other hand, depends on the structural constraint. However there exist approximate inference algorithms, and they could be applied for inference on a mixture of unconstrained terms.

The complexity of learning and inference for a mixture is $\mathcal{O}(mf(p))$ if each term is learned only once and independently from the others, where $f(p)$ is the complexity of learning or performing inference on one term of the mixture.

Different classes of models that could be used to construct mixtures will be discussed by increasing inference complexity: models of tree-width 0, acyclic models, and other models.

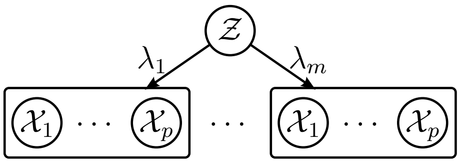
Tree-width 0: Naive Bayes

A Bayesian network of tree-width zero contains no edge and defines the following factorization of the joint probability distribution:

$$\mathbb{P}(\boldsymbol{\mathcal{X}}) = \prod_{i=1}^{p} \mathbb{P}(\mathcal{X}_i) \ . \tag{4.4}$$

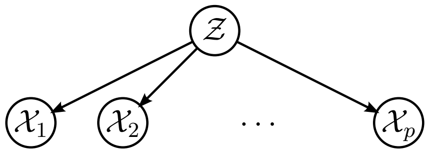
Learning the structure of such a distribution is of course immediate and learning its parameters is trivial and of $\mathcal{O}(Np)$ complexity. Inference in such a model is very efficient too and only depends on the queried variables, not on the variables observed.

A mixture of models of tree-width zero can be represented in two different ways. As any other mixture model presented so far, the weights of the mixture can be viewed as the parameters of a categorical hidden variable $\mathcal{Z}$, and each term of the model encodes a joint probability distribution (here of tree-width 0) defined over the full set of variables, as represented in Figure 4.1a. The second representation is possible because the graphical structure is identical for all terms. Therefore the mixture model can be represented as that unique graphical structure (in the case of the naive Bayes, an edgeless model) where the weight variable conditions every other variable. This structure, represented in Figure 4.1b, is a more classical depiction of a naive Bayes model.



$\mathcal{Z}$
λ_1
λ_m
$\mathcal{X}_1$ $\cdots$ $\mathcal{X}_p$
$\cdots$
$\mathcal{X}_1$ $\cdots$ $\mathcal{X}_p$

(a) The first representation emphasises the mixture decomposition and makes explicit the number of terms m.



(b) This more classical representation of the naive Bayes model stresses the independence of the variables conditionally on $\mathcal{Z}$.

Figure 4.1: A mixture of 0 tree-width Bayesian networks has two possible graphical representations.

The naive Bayes model is mostly exploited in classification and clustering [CS96, DP97]. In that case the hidden variable $\mathcal{Z}$ is the class (or the cluster) variable, the probability $\mathbb{P}(\mathcal{Z}|\mathbf{x})$ encodes the probability that a given sample $\mathbf{x}$ belongs to the different classes or clusters.

Nevertheless, a few algorithms have been proposed to build naive Bayes model for density estimation.

[KMT96] constructs a mixture of independent variables as a generative model for classification. This model is a mixture model as considered in this thesis and not a naive Bayes classifier: the root of the tree is not one of the observed variables. The number of terms is selected empirically in [KMT96].

[LD05] uses a variant of the EM algorithm. It is adapted to address the problem of selecting the number of terms and the problem of local maxima. Starting from an initial number m of terms, those characterised by a low weight are periodically removed from the mixture. In addition, once a mixture has converged, new random terms (based on randomly assigned samples) are generated and added to the mixture. The iterative EM update is then restarted on the modified mixture. New terms are added according to this procedure until the resulting mixture does not improve (in terms of data fit) over the previous one.

When a naive Bayes model is built for density estimation over the variables $\boldsymbol{\mathcal{X}}$, the complexity of inference is linear in the number of variables considered, both as queried and observed variables.

Acyclic Models of Tree-width 1: Markov Forests

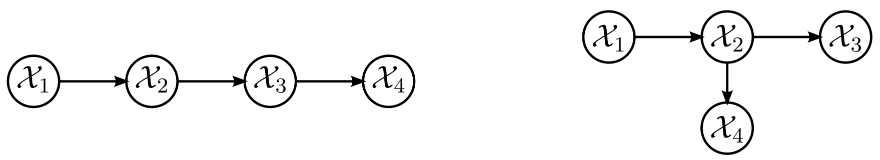
These models are characterized by a graph without cycle where each variable has at most one parent. This class of models contain Markov trees.

For Markov forests, inference of a marginal conditional probability distribution is of linear complexity in the number of variables (see Section 3.4.1 for details). Fully connected models of this type can be partitioned into two categories, based on the shape of their graphical structure, as illustrated in Figure 4.2:

- a **chain** is a structure where each variable has at most one parent and one child;

- a **tree** is a structure where each variable has at most one parent.

Due to the absence of any v-structure, chains and trees can also be considered as undirected models. Considering unconnected structures does not increase the upper bound on the complexity of inference: inference is performed separately on each connected component, because the variables of each component are independent from the variables of any other component. However, learning unconnected models may make learning more difficult and usually involves regularization. Regularization of Markov trees is discussed in more details in Section 5.1.1.



(a) A chain (b) A tree

Figure 4.2: There are 2 categories of graphical models whose undirected graph is acyclic and connected.

Constructing a chain or a tree $\mathcal{G}$ in order to encode a distribution $\mathbb{P}_{\mathcal{G}}$ minimizing the Kullback-Leibler divergence with respect to a target distribution $\mathbb{P}$ amounts to maximizing (see Section 3.3.4 for details):

$$\mathcal{G} = \arg\max_{\mathcal{G}'} \sum_{(\mathcal{X},\mathcal{Y})\in E(\mathcal{G}')} I_{\mathbb{P}}(\mathcal{X};\mathcal{Y}) \ , \tag{4.5}$$

while $\mathcal{G}'$ is constrained to be either a chain or a tree. In the case of chains, this problem reduces to the traveling salesman problem. It is therefore NP-hard [Mee01]. For trees, this problem is a maximum-weight spanning tree problem in a complete graph, solvable by $\mathcal{O}(p^2 \log p)$ or faster algorithms, and is explained in details in Section 3.3.4.

Since chains cannot be learned optimally, they can not easily be used to build a bias reducing mixture by using the EM algorithm introduced in Section 2.6.1.

Both can be used in a variance reduction scheme. However, my preliminary experiments (not reported in this thesis) to construct mixtures of chains by using the heuristic proposed in [DBIV96] led to results clearly worse than with mixtures of trees. Learning and inference algorithmic complexity are similar (because of the heuristic), but the accuracy was much worse for mixtures of chains. I conjecture this is due to a higher bias of chains with respect to trees. This area of research was not pursued further.

In addition, working in the class of tree structures one can easily construct an optimal model of the class, by using the Chow-Liu algorithm. The absence of an algorithm to obtain such an optimal model is not problematic in the perturb and combine framework, since it will be randomized. However the evaluation of the algorithms greatly benefits of the reference score provided by this optimal solution inside the class.

Acyclic Models of Tree-width >1: Polytrees

These models are characterized by a graph without any cycle. As opposed to Markov forests, there is no restriction on the number of parents. In that case, inference of a marginal conditional probability distribution is more complex, and depends on the exact tree-width of the model, i.e. here the maximum number of parents of any variable minus 1 (see Section 3.4.1 for details).

Constructing a polytree $\mathcal{G}$ to minimize the Kullback-Leibler divergence with respect to a target distribution $\mathbb{P}$ is NP-hard [Das99]. However, if the target distribution $\mathbb{P}$ is known to be a polytree, it can be recovered asymptotically in the number of samples [RP87, OOM04]. [Das99] also shows that the result of the Chow-Liu tree is a good approximation of the best polytree, in the sense that:

$$\begin{aligned} &\text{nlogll}_D(\mathcal{T}_{\text{chow-liu}}) \leq \\ &\qquad \text{nlogll}_D(\mathcal{G}_{\text{best polytree}}) \left(1 + \mathcal{O}\left(\log \frac{\max_i H_D(\mathcal{X}_i)}{\min_i H_D(\mathcal{X}_i)}\right)\right) \quad , \end{aligned} \tag{4.6}$$

where $\text{nlogll}_D(.)$ and $H_D(.)$ are respectively the negative log-likelihood and the entropy functions, according to the empirical distribution observed in the learning set D. In addition, an experimental evaluation of the interest of learning a polytree from data has been carried out by [AdC95] when the original distribution is not a polytree. Moreover, [GKL$^+$11] recently proposed an algorithm to learn an optimal k-branching, i.e. a polytree that can be transformed into a tree (or more precisely in a forest) by removing at most k edges from its structure. The complexity of this algorithm is $\mathcal{O}(n^{3k+4})$.

Polytrees therefore suffer from the same disadvantage than chains, namely a NP-hard complexity for learning them optimally, a drawback for using them in both types of mixtures. Unlike chains however, inference on polytrees is not of linear complexity in p.

In addition, a comparison between mixtures of uniformly sampled tree structures and mixtures of uniformly sampled polytree structures showed no advantage of the latter over the former [ALDW08].

Other Models of Tree-width 2 or Higher

The complexity of inference increases with the tree-width of the model in the absence of any other constraint, as explained in Section 3.4.

Many learning methods for PGMs specifically target low tree-width structures [BJ01, CG07, EG08, FNP99, SCG09] in order to ensure tractable inference complexity. Those methods are all heuristic, because learning a graphical structure of tree-width higher than 1 is NP-complete [KS01, Sre03].

To the best of my knowledge, no mixture of bounded tree-width Bayesian networks has ever been considered, although inference could still be tractable for a reasonable number of variables. Both types of mixtures could be interesting for those models. In the maximum-likelihood setting, the tree-width bound on the models considered in the mixture is an additional parameter that can be tuned to achieve the desired trade-off between algorithmic complexity and accuracy. In a variance reduction framework, making a mixture would logically be even more beneficial for bounded tree-width models than for Markov trees, since the formers have more parameters then the latters and are therefore more prone to overfitting.

Mixtures of unbounded Bayesian networks have already been considered, for small numbers of variables. In the context of high-dimensional problems, they are not really attractive, because of the associated algorithmic complexity.

[TMCH98] optimizes such a model by a 3-step iterative process[1]:

1. an EM algorithm to optimize the weights and parameters of the mixture while keeping the structure fixed for each term;
2. an optimization of the structure of each term, through the maximization of a modified (to make it decomposable) Cheeseman-Stutz score [CS96, CH97], an approximation for the maximum likelihood score of a data set with missing variables, here constructed around the parameters and weights estimated in the first step;
3. one M-step of an EM algorithm to optimize the weights and parameters of the new mixture structure.

The key point of this approach is that the EM algorithm is not involved in the structure optimization of each term.

[CH92] suggests performing inference on all possible graphical model structures, and weighting the results by the probability of the structure given the data set and the observations. While [CH92] notes that this is intractable for more than a few variables, they point out alternatives such as searching for a set of good models and combining their predictions, or estimating the average using standard stochastic simulation techniques. [MR94] and several other works developed alternative techniques to sample a set of good structures. These approaches can also be used to predict the probability of a structural feature in the real network. See Section 3.3.5 for details and additional methods.

[1]The algorithm is developed to deal with a learning set with missing values.

[RS02] constructs a mixture of Bayesian networks by considering them as weak learners in a boosting algorithm [FS95]. Boosting is a model averaging meta-algorithm that iteratively expands a mixture by applying a given learning algorithm on a reweighted data set. The weights are modified for learning each new model so that the observations that are not modelled accurately by the previous models have a larger weight.

4.2 Literature on Mixtures of Markov Trees

Several learning algorithms for mixtures of Markov trees have already been proposed. While most algorithms could be described as methods learning tree structures on reweighted data sets, the different reweighting schemes used lead to a partition of those methods into three categories, based on the bias-variance trade-off.

This section is meant as a state of the art review of existing algorithms for learning such mixtures, and therefore each method is only briefly described. However, some algorithms mentioned here will be further discussed in the core of this thesis, when they inspire and/or are compared to the algorithms developed. Relevant information about all existing methods are summarized in Table 4.1.

target	references	complexity	sections
bias	[MP99, MJ01]	$mp^2 \log p$	2.6.1,7.1.1
only	[KK09]	$m^2 + mp^2 \log p$	
variance	[ALW08, ALDW08, Amm10]	p	5.4.1,6.1
only	[ALDW09b, ALDW09a, Amm10]	$mp^2 \log p$	2.6.2,6.1
	[ALW10b, ALW10a, Amm10]	$mK \log K$	6.2.1
both	[KK06, KS07]	$m_t p^3$ (iteration t)	

Table 4.1: Existing algorithms for mixtures of Markov trees can be divided into 3 categories. K is a parameter, and corresponds to a number of edges. Therefore, $0 \leq K \leq p(p-1)/2$.

4.2.1 Bias Reduction

In the first two approaches, the targeted component of the error is the bias. The mixture of trees is primarily used as a mean to exploit the good algorithmic properties of trees while improving their modeling capabilities.

The first method uses the EM algorithm to simultaneously partition the learning set between a given number of terms [MJ01, MP99], which minimizes the negative log-likelihood of the learning set.

The second method [KK09] builds a mixture of increasing size by using for each new tree a clever reweighting scheme on the whole data set based on fractional covering. This identifies the highest mode of the density not covered yet and minimizes the ∞-divergence:

$$D_\infty(\mathbb{P}||\mathbb{P}_{\mathcal{T}}) = \max_{\mathbf{x}\in Val(\boldsymbol{\mathcal{X}})} \log \frac{\mathbb{P}(\mathbf{x})}{\mathbb{P}_{\mathcal{T}}(\mathbf{x})} \ . \tag{4.7}$$

The first method has e.g. been used in computer vision for human tracking [IF01a] and object recognition [IF01b], or in optimization through estimation of distribution algorithms [SORS01].

4.2.2 Variance Reduction

More recently, mixtures of Markov trees have been constructed to reduce the variance of a Markov tree. In these methods, a set of tree models are generated using a more or less strongly randomized procedure, that can range from completely random structures based on Prüfer lists to bagged Mixtures of trees. The weights associated to these trees can be either uniform or proportional to the score of the structure based on the data set. A good selection of theses approaches can be found in [ALSW10]. The present work builds on this strategy and these methods are further described in Chapter 6.

In addition, it is possible to perform true Bayesian inference over Markov trees, by considering all structures and weighting them by their posterior probability according to the learning set. Both the prior and the posterior distributions of tree structures are decomposable: they can both be expressed as a product of a set of terms, where one term is associated to each edge of a tree structure. The prior and posterior distributions over the parameters are also decomposable. It is therefore possible to compute the posterior distribution of any tree structure in closed-form for the class of

Markov trees [MJ06] by an application of Kirchhoff's matrix-tree theorem. However, the resulting model suffers from two drawbacks. Since it requires the inversion of a matrix of size $p \times p$, the complexity of this approach is cubic in the number p of variables and therefore not appealing for high-dimensional problems. Moreover, inference is intractable on the resulting model: marginalizing is not possible, only the probability of a value $\mathbf{x} = \boldsymbol{\mathcal{X}}$ of the full set of variables can be computed.

4.2.3 Bias and Variance Reduction

Finally, the third category is a combination of the first two categories. Taking a Bayesian approach to mixtures of Markov trees, [KK06] has developed a Markov chain Monte-Carlo procedure over the space of possible partitions of a learning set into k (a fixed, user specified number) distinct subsets of observations. On each subset of the partition, they construct a Bayesian posterior distribution over Markov trees (see below). [KS07] has proposed an alternative MCMC exploration scheme of the space of mixtures of Markov trees. This scheme is defined using a Dirichlet process and a suitable prior on tree structures [MJ06]. By using the Dirichlet process, they allow the number of trees in the mixture to vary during the MCMC iterations (k must not be specified and is modified by the algorithm).

4.3 Perturb and Combine of Markov Trees

This section describes the general meta-algorithm for building mixtures of Markov trees. Those trees are constructed iteratively by applying the perturb and combine framework, described in Section 2.6.2, to the Chow-Liu algorithm, presented in Section 3.3.4.

As a short reminder, the perturb and combine framework consists in randomizing an "optimal" algorithm (here the Chow-Liu algorithm), and in combining, or averaging, several suboptimal models produced by this perturbed learning algorithm. The first step makes the algorithm stochastic. This reduces the dependence of the model on the training data, and may help to improve the algorithmic complexity of learning. The second step leads to a reduction in overfitting, thanks to a reduction of the variance of the model learned. This method may therefore lead to an increase in performance.

4.3.1 Why Geometric Averaging is not Used

Both an arithmetic and a geometric average of the distributions can be considered to build a mixture to reduce the variance, each based on a specific bias-variance decomposition of the learning error, as discussed in Section 2.4.2. A mixture using either aggregating scheme on a set of m PGMs can be viewed as an attempt to approximate the mean model underlying the decomposition:

$$\begin{aligned}\bar{M}^G(\boldsymbol{\mathcal{X}}) &= \frac{1}{Z}\exp(\mathbb{E}_D \log M_D(\boldsymbol{\mathcal{X}})) &(4.8)\\ &\approx \frac{1}{Z'}\exp(\sum_{j=1}^{m}\log \mathbb{P}_{\mathcal{T}_j}(\boldsymbol{\mathcal{X}})) &(4.9)\end{aligned}$$

for the geometric mean and

$$\begin{aligned}\bar{M} &= \mathbb{E}_D M_D(\boldsymbol{\mathcal{X}})) &(4.10)\\ &\approx \frac{1}{m}\sum_{j=1}^{m}\mathbb{P}_{\mathcal{T}_j}(\boldsymbol{\mathcal{X}}) &(4.11)\end{aligned}$$

for the arithmetic one. In both cases $\mathbb{P}_{\mathcal{T}_j}$ denotes the probability distribution defined by the the jth Markov tree of the mixture, $\mathcal{T}_j$.

Of those two mixtures, the arithmetic mean is the most interesting one.

From an algorithmic point of view, the geometric mean contains a normalization constant Z'. Computing it would mean integrating over all possible configurations of variables. This is infeasible for a large number p of variables, because the number of configurations is exponential in p. On the other hand, there is no such problem with the arithmetic mean. By using a set of weights defining a convex combination, the probability density encoded by the model is automatically normalized (provided each term encodes a normalized density).

From a theoretical point of view, it was shown in Section 2.4.2 that the variance of the decomposition of the error based on the geometric mean may be smaller than the variance of the decomposition based on the arithmetic mean. If the variance associated to the geometric mean is smaller, the error of a geometric aggregation scheme is greater than the error of the arithmetic scheme. Indeed, the error of the geometric or arithmetic averaged model (if it is computed exactly, i.e. by using an infinite number of leaning sets) equal the bias of the corresponding bias-variance decomposition.

In addition to being more tractable, an arithmetic averaging scheme may therefore also be more accurate than a geometric averaging scheme.

4.3.2 Meta-algorithm for Learning a Mixture of Markov Trees

The meta-algorithm for learning a mixture of Markov trees for statistical estimation and inference is stated in Algorithm 4.1. It takes as input a set of variables, a learning set of observations and an integer m (the number of terms). It applies m times the three subroutines that

i generate a tree structure labeled by the variables (SampleTreeStructure),

ii learn a set of parameters (LearnParameters),

iii and determine a weight for this structure (LearnWeight).

When the m trees have been learned, the weights are normalized (assuming they were not already summing to 1). Note that for clarity, potential configuration arguments of SampleTreeStructure or modifications on the learning set supplied to each method are not displayed and are left implicit.

Algorithm 4.1 Meta-algorithm for learning a mixture [ALDW08, Amm10]

```
Input: set of variables $\boldsymbol{\mathcal{X}}$; learning set $D$; number of trees $m$
$\boldsymbol{\mathcal{T}} = \emptyset$; $\boldsymbol{\lambda} = \emptyset$; $\boldsymbol{\theta} = \emptyset$
for $j = 1 \to m$ do
  $\boldsymbol{\mathcal{T}}[j]$ =SampleTreeStructure($\boldsymbol{\mathcal{X}}$,$D$ [,$\boldsymbol{\mathcal{T}}$])
  $\boldsymbol{\theta}[j]$=LearnParameters($\boldsymbol{\mathcal{T}}[j]$,$D$)
  $\boldsymbol{\lambda}[j]$ =LearnWeight($\boldsymbol{\mathcal{T}}[j]$,$D$)
end for
Normalize($\boldsymbol{\lambda}$)
return ($\boldsymbol{\mathcal{T}}$,$\boldsymbol{\theta}$,$\boldsymbol{\lambda}$)
```

This meta-algorithm is later instantiated into many different algorithms for learning mixtures of Markov trees, based on the different variants of its three subroutines. These different variants will be described later in the thesis (and more specifically in Chapter 6), when the algorithms based on this meta-algorithm are instantiated and tested. Nevertheless, a few comments based solely on the structure of the meta-algorithm can already be made here.

LearnParameters and LearnWeight each take a single tree structure as input, so the parameters and weight of any tree are learned independently from the other tree structures. Other strategies may be possible, e.g. forcing the parameters of a tree to be different from the others, so as to increase the diversity of the models generated. These strategies are however not considered in this thesis.

SampleTreeStructure is the subroutine that will receive the most attention in this thesis. It has a large influence on the resulting mixture: both LearnParameters and LearnWeight takes as argument the tree structure generated by SampleTreeStructure.

I will distinguish between two classes of SampleTreeStructure subroutines, depending whether the tree structures generated by SampleTreeStructure subroutine are independent from each other, or not. This is highlighted by the optional argument [,$\boldsymbol{\mathcal{T}}$] of the subroutine. When this argument is absent, each tree structure is independent from the others (conditionally on the learning set). It will therefore be denoted as the *independent Markov trees* category. In the second category, this argument is provided to the subroutine, and each tree structure is influenced by all or some of the previous structures computed.

4.4 Organization of Part II

Some of the main contributions of this thesis are the development and the evaluation of new algorithms for learning mixtures of Markov trees, based on this meta-algorithm and using existing or new variants of the three subroutines. These contributions will be presented in the following chapters, organized as follow.

- Chapter 5 focuses on the construction of a single Markov tree based on a learning set. This analysis is important, because SampleTreeStructure is the subroutine for which the most variants will be constructed in this thesis. Chapter 5 will discuss existing approaches to construct a Markov tree structure, and propose two new randomization schemes of the Chow-Liu algorithm. These schemes are evaluated against the Chow-Liu algorithm in the context of the construction of a single tree structure. In addition, that chapter also discusses the regularization of the Chow-Liu algorithm. Regularization is another approach to reduce the variance of an algorithm. Based on this discussion, a gold standard method is constructed for regularization. It will be used to assess the accuracy of mixtures of Markov trees (built later) against regularization.

- Chapter 6 revolves around the specialization of meta-algorithm 4.1. For the three subroutines, different variants are discussed (in addition to the variants of SampleTreeStructure developed in Chapter 5). These subroutines are then combined to generate mixtures of Markov trees. There are two main parts in this chapter: one part per category of SampleTreeStructure subroutines. The first part deals with mixtures of independent Markov trees, the second part with mixtures of sequential Markov trees. In each part, new algorithms are developed and evaluated.
- Chapter 7 combines bias and variance reducing mixtures. More specifically, this chapter evaluates the interest of replacing each term of a bias reducing mixture, originally a single Markov tree, by a variance reducing mixture such as those developed in Chapter 6.

Part II

New algorithms for learning mixtures of Markov trees

Chapter 5

Learning a Single Markov Tree

This chapter takes a closer look at the construction of one Markov tree structure based on a set of observations. The algorithm for learning the maximum likelihood structure is first recalled. Then different methods for regularizing this algorithm are discussed. Finally, several algorithms for accelerating the Chow-Liu algorithm are presented. In particular, two approximate methods trading accuracy for computational speed are proposed, and evaluated empirically.

The Chow-Liu (CL) algorithm [CL68] optimizes the likelihood of the learning set over the set of connected tree structures. It was already developed in Section 3.3.4, and it is first briefly recalled here, in Section 5.1. However when the number of learning samples is small, it might me necessary to further constrain the structure so that it generalizes better. Section 5.1.1 discusses the introduction of a regularization penalty inside the Chow-Liu algorithm, and Section 5.1.2 presents a "gold standard" method used to measure the accuracy achievable by regularizing the Chow-Liu algorithm. This "gold standard" will be compared to the mixtures of Markov trees developed in this thesis, in Chapter 6.

Good algorithmic scaling, in particular for inference, has made Markov trees a model of choice to tackle large problems. Despite its quadratic complexity, the Chow-Liu algorithm may take a long time to run on very large problems, and a few specialized versions of this algorithm have been developed for specific situations. This area of research is of particular importance to this thesis, because learning a tree structure must be performed many times to learn an ensemble model, and is a core subroutine of the algorithms developed in the present work.

In the context of high-dimensional learning, the critical component of the complexity is the dependency in p. Lowering the influence of p on the run time of the CL algorithm is the main goal of this chapter. Unfortunately, existing specialized versions of the CL algorithm were not oriented towards that objective: one develops a compression scheme for sparse learning sets (Section 5.2), another is an approximation motivated by large number of samples N (Section 5.3). However, there exists an efficient algorithm to generate tree structures uniformly (Section 5.4).

Therefore I have developed two randomization of the Chow-Liu algorithm based on edge subsampling, which effectively lowers the complexity in p. These approximations are described in Sections 5.4, and empirically evaluated in Section 5.5.

Note that constructing mixtures based on these algorithms is studied in Chapter 6. Moreover, the present chapter will only discuss learning an undirected Markov tree structure. Constructing a Markov tree from a given undirected tree structure is easy, and described in Algorithm 5.1. The tree must first de directed, by randomly choosing a root and orienting the tree downwards from this root. For forest structures, one root must be chosen in each connected subtree of the forest. The model can then be parametrized, using any parameter estimation method (see Section 3.5.2 for details).

Algorithm 5.1 Markov tree construction meta-algorithm

```
**Input:** learning set $D$; undirected tree $\mathcal{T}$ labeled by a variable set $\boldsymbol{\mathcal{X}}$.
$r = \text{RandInteger}(1, p)$
OrientDownwardsFromRoot($\mathcal{T}, r$)
$\theta = \text{LearnParameters}(\mathcal{T}, D)$
**return** $(\mathcal{T}, \theta)$.
```

5.1 The Chow-Liu Algorithm

The Chow Liu algorithm, introduced by Chow and Liu [CL68], computes a Markov tree maximizing the likelihood of a training set D. A detailed presentation of this algorithm is available in Section 3.3.4, but the structure learning part of the algorithm is recalled here for convenience before its specialized versions and approximations are discussed.

The Chow-Liu algorithm computes an optimal Markov tree structure[1]

$$\mathcal{T}_{CL}(D) = \arg\max_{\mathcal{T}} \sum_{(\mathcal{X}_i,\mathcal{X}_j)\in E(\mathcal{T})} I_D(\mathcal{X}_i;\mathcal{X}_j) \tag{5.1}$$

in two steps (see also Algorithm 5.2):

1. computation of $p \times (p-1)/2$ pairwise mutual information values between variables;
2. use of the these values as an edge-weight matrix to build a maximum weight spanning tree (MWST), e.g. using [Kru56].

Therefore it has essentially a time and space complexity of $\mathcal{O}(p^2N+p^2\log p)$, assuming the Kruskal algorithm is used in the second step (slightly faster algorithms exist [Cha00]).

Algorithm 5.2 Chow-Liu (CL) tree [CL68]

```
**Input:** $\boldsymbol{\mathcal{X}}$;$D$
$MI = [0]_{p\times p}$
**for** $i-1 = 1 \to p-1$ **and** $i_2 = i_1+1 \to p$ **do**
  $MI[i_1,i_2] = MI[i_2,i_1] = I_D(\mathcal{X}_{i_1};\mathcal{X}_{i_2})$
**end for**
$\mathcal{T}_{CL}$ = MWST($MI$) {MWST computation e.g. using [Kru56] as here}
**return** $\mathcal{T}_{CL}$.
```

5.1.1 Regularization

The CL algorithm always outputs a connected tree structure. However, a model associated to this structure may still suffer from overfitting, see e.g. [LXHG$^+$11, TAW11]. Regularization is one approach to countering this phenomenon, and it is the focus of this section.

Adding an edge to any Bayesian network never decreases the likelihood of a learning set. The Chow-Liu algorithm therefore always outputs the maximum number of edges, but this can lead to an overfit of the learning set. A Markov forest could lead to a better accuracy than a Markov tree. Such a situation can appear for at least the two following reasons.

[1]If there is a tie between several structures, one is chosen randomly.

- If the structure of the target distribution is a set of disconnected components, an edge between two variables belonging to different subcomponents is not necessary. Such an edge will not decrease the accuracy of the model if its parameters are estimated based on a perfect knowledge of the target distribution. However, estimating them from a set of observations is likely to introduce some noise in the model, especially since the Chow-Liu algorithm will link two components (assuming it first construct a forest spanning each component) by the edge where the noise is the strongest.

- If the number of samples is small, estimating the parameters by the maximum likelihood principle will model sampling noise in addition to significant information. The model may therefore not generalize well on unseen instances and may have poor performance in practice. It may therefore be better to limit the number of edges, even if the target distribution has a connected structure.

Constructing a forest rather than a tree is able to address these problems.

Algorithms for learning a Markov forest were recently studied in two papers published almost at the same time: [TAW11] and [LXHG+11]. Although they center their theoretical analysis around the recovery of a Markov forest, they also consider approximating a distribution whose underlying graph is connected (and not a tree). They are therefore rather interesting in the context of this section, and are briefly summarized here.

[TAW11] considers learning a forest over categorical variables as a thresholding problem. The Chow-Liu algorithm is regularized by modifying its optimization criterion so as to penalize model complexity in terms of its number of edges $|\mathcal{F}|$,

$$\mathcal{F}(D) = \arg\max_{\mathcal{F}} \sum_{(\mathcal{X},\mathcal{Y})\in E(\mathcal{F})} I_D(\mathcal{X};\mathcal{Y}) - \lambda_N|\mathcal{F}| \ , \tag{5.2}$$

where $\lambda_N \geq 0$ is the threshold. The optimal solution to this problem can be obtained by modifying the Chow-Liu algorithm, to return the maximum weight "forest model" spanning the graph containing the edges for which $I_D(\mathcal{X};\mathcal{Y})$ is greater than λ_N. Higher values of λ_N lead to sparser forests, in the limit to an empty graph.

[TAW11] makes this threshold dependent on N, and demonstrates how defining λ_N such that

$$\lim_{N\to\infty} \lambda_N = 0 \qquad\qquad \lim_{N\to\infty} \frac{N\lambda_N}{\log N} = \infty \ , \tag{5.3}$$

if a sufficient condition to ensure that the estimation is consistent for the structure and the density estimation when the true distribution is indeed a Markov forest, or consistent with respect to the best projection on a Markov forest when the true distribution is not such a model. The key idea is that λ_N must converge to 0 so that all mutual information values above 0 in the true distribution are eventually considered, but the convergence must be slower than the expected value of a mutual information equal to 0 estimated on a finite learning set.

They suggest using λ_N of the form $N^{-\beta}$, $\beta \in]0, 1[$, but offer no practical way to select β.

As for supervised decision tree growing [Weh93], penalizing in this way the tree complexity is tantamount to using a hypothesis test for checking independence of the next pair of variables to be included. Such a test can therefore be performed by comparing the quantity of interest to a threshold depending on a postulated p-value, say $\rho = 0.05$ or smaller (see Section 3.3.5). This allows the definition of a more interpretable parameter, but also naturally adapts the threshold to the cardinality of the variables, an approach I suspect to be more appropriate to reduce overfitting.

[LXHG$^+$11] focuses on continuous variables, and therefore devotes a lot of time to the problem of estimating mutual information using kernels. Other parts of the paper are nevertheless relevant for categorical variables too. Two problems are investigated: estimating an optimal forest based on a learning set and estimating a maximally connected forest, where each subtree can contain at most k (a parameter) variables. The latter problem is NP-hard, hence not interesting in the context of this thesis, and will not be discussed here. Two methods are proposed to solve the first problem, both based on a bipartition of the learning set, and their asymptotic consistency is established.

M1. The weights are computed on the first set of samples, and the Kruskal algorithm is used to generate a sequence of forests. The algorithm first outputs an empty structure, and then outputs the structure obtained after each edge addition. This sequence contains all the structures (provided there is no tie) that could be generated by a penalized Chow-Liu algorithm (Equation 5.2), where λ_N can take any value in $[0, \infty]$. The optimal forest is then chosen based on their score on the second set of samples. This amounts to evaluating all possible threshold on the first set of observations, and selecting the best one based on the second set.

M2. Mutual information estimates are based on probabilities estimated on the two partitions D_1 and D_2:

$$I_{D_1,D_2}(\mathcal{X};\mathcal{Y}) = \sum_{x,y} \mathbb{P}_{D_1}(x,y) \log \frac{\mathbb{P}_{D_2}(x,y)}{\mathbb{P}_{D_2}(x)\mathbb{P}_{D_2}(y)} \quad . \tag{5.4}$$

5.1.2 “Gold” Standard for Regularization

Most algorithms proposed within this thesis attempt to improve over a single Chow-Liu tree by reducing the variance. Because a regularization of the Chow-Liu algorithm has the same objective, it is of interest to provide a regularized method to be compared against the approaches developed here. This section describes this very method, that will be denoted by rCL.

The regularization methods described so far depend either on a partition of the learning set, a threshold on mutual information or a p-value for an independence test. The choice of any particular regularization method as a standard to compare against the algorithms developed here would be open to criticism. Therefore I decided on using an optimistic regularization method that would provide an upper bound on the accuracy (a lower bound on the KL divergence or the negative log-likelihood) of any regularized method based on a threshold. This optimistic method uses the test set to optimize the number K of edges in the structure rather than the learning set. Edges of the Chow-Liu trees are added one by one, by decreasing weight, in an empty graph, as in the Kruskal algorithm. The structure is recorded and evaluated after each edge addition, producing a sequence of decreasingly regularized models $\hat{\mathbb{P}}_K$, where K is the number of edges in the Markov forest. Each model is evaluated on the test set, and the score of the best model is considered the best possible result achievable through regularization. The score of the regularization method on a given learning set D is therefore:

$$\min_K \operatorname{score}(\mathbb{P}, \hat{\mathbb{P}}_K | D) \ , \tag{5.5}$$

assuming that the better the model, the lower the score.

Figure 5.1 offers an illustration of this process on the *Gene* problem and on a learning set of $N = 200$ samples. The negative log-likelihood of an independent test set tends to decrease for the first edges, until a minimum is reached. From that point and as the last edges are included in the model, the log-likelihood increases.

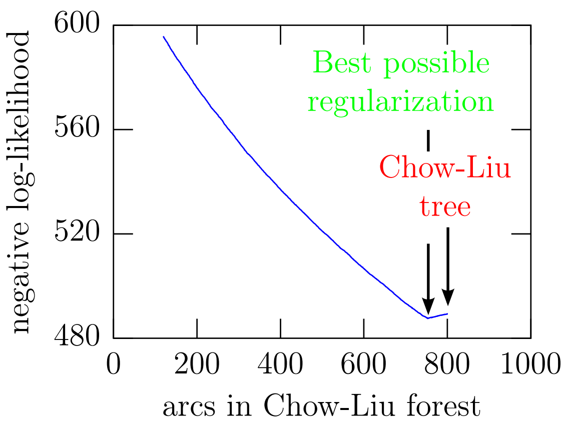


Figure 5.1: An illustration of the regularization of the Chow-Liu algorithm, on the Gene distribution and for 200 samples. The number of edges is progressively increased from 0 to $p-1$.

The optimal structure can still be a forest, even if the number of edges is optimized on the real distribution and this distribution is encoded by a connected structure. The parameters of the tree are still estimated based on the learning set. Adding an edge can decrease the accuracy of the resulting model if the parameters associated to this edge are badly estimated due to a lack of samples.

In practice, choosing the optimal model is slightly more complicated, because several learning sets D and several target distributions $\mathbb{P}$ are usually used to evaluate the algorithms in a given problem setting (number of variables p, number of samples N, maximum number of parents etc), in order to decrease the variance of the estimates (see also Section 2.3.1). The combination of the estimates obtained on these learning sets leads to a global score for the algorithm:

$$\sum_{\mathbb{P}} \sum_{D|\mathbb{P}} \text{score}(\mathbb{P}, \hat{\mathbb{P}}|D) \ . \tag{5.6}$$

Three different global scores could be generated, based on three different methods to select the best number of edges:

$$\text{score}_1^{reg} = \min_K \sum_{\mathbb{P}} \sum_{D|\mathbb{P}} \text{score}(\mathbb{P}, \hat{\mathbb{P}}_K|D) \tag{5.7}$$

$$\text{score}_2^{reg} = \sum_{\mathbb{P}} \min_{K|\mathbb{P}} \sum_{D|\mathbb{P}} \text{score}(\mathbb{P}, \hat{\mathbb{P}}_{K|\mathbb{P}}|D) \tag{5.8}$$

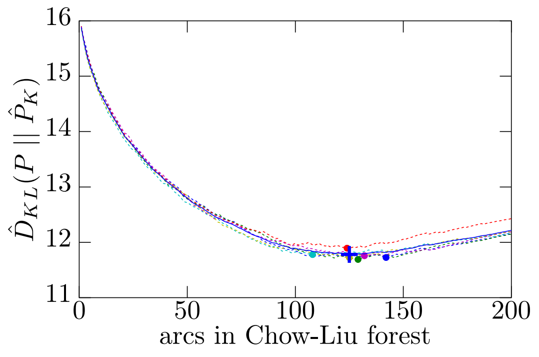
$$\text{score}_3^{reg} = \sum_{\mathbb{P}} \sum_{D|\mathbb{P}} \min_{K|D,\mathbb{P}} \text{score}(\mathbb{P}, \hat{\mathbb{P}}_{K|D,\mathbb{P}}|D) \ . \tag{5.9}$$

In the first score, K is optimized for a given problem configuration. In the second, $K|\mathbb{P}$ is optimized for every problem configuration and target distribution. In the third, $K|D, \mathbb{P}$ is different for every learning set. The finer K can be optimized for each target distribution and learning set, the better the score:

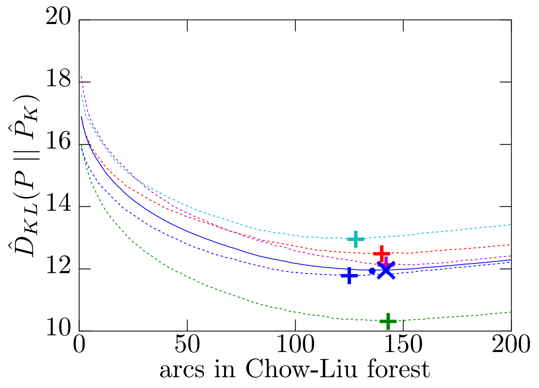
$$\text{score}_1^{reg} \geq \text{score}_2^{reg} \geq \text{score}_3^{reg} \ . \tag{5.10}$$

I have considered the second score as the golden score of a regularization method. This second score can be viewed as the score of a regularization method that knows the optimal number of edges for a given distribution and for a set of learning sets of a given size N. This seemed like a reasonable assumption: such an optimal number of edges could be provided by an expert with a good knowledge of the problem at hand. Knowing the behavior of a particular realization of the sampling process generating the learning set (the third score) is however less likely. Nevertheless, in the settings considered here, selecting the third score rather than the second did not seem to improve it much (see e.g. Figure 5.2)

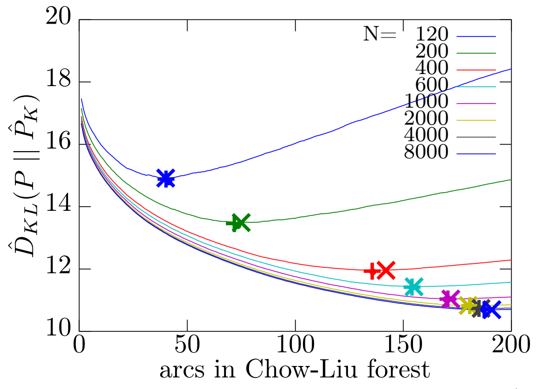
The optimal results achieved by this gold standard are summarized in Table 5.1 for both realistic and synthetic distributions. In addition, the three aggregated scores discussed are illustrated for one synthetic distribution in Figure 5.2. Notice that, in synthetic distributions, the score is an



(a) For a given distribution, $N = 400$ and 6 learning sets. Each dashed line corresponds to score$(\mathbb{P}, \hat{\mathbb{P}}_K|D)$ for one learning set, the full line to the average.



(b) For $N = 400$. Each dashed line corresponds to $\frac{1}{|D|\mathbb{P}|}\sum_{D|\mathbb{P}}$ score$(\mathbb{P}, \hat{\mathbb{P}}|D)$ for one distribution, the full line to the average.



(c) Each line is the mean score $\frac{1}{|\mathbb{P}|}\sum_{\mathbb{P}}\frac{1}{|D|\mathbb{P}|}\sum_{D|\mathbb{P}}$ score$(\mathbb{P}, \hat{\mathbb{P}}|D)$, for a given N.

Figure 5.2: The three possible scores for the optimally regularized method are illustrated on 5 DAG-200-5 distributions times 6 learning sets. x markers signal the first score (Equation 5.7), + markers the second (Equation 5.8), dot markers the third (Equation 5.9).

Distribution	N	Chow-Liu		Gold standard	
		score	edges	score	edges
DAG-200-5	200	14.9	199	13.5	70.8
	600	11.6	199	11.4	153
	1000	11.1	199	11.0	169.6
DAG-1000-5	200	79.7	999	67.2	302.8
	600	57.2	999	55.6	711.6
	1000	54	999	53.4	805
Alarm10	200	166.65	369	166.65	369
	500	136.37	369	136.28	336
	2500	129.99	269	129.96	349
Child10	200	135.29	199	135.08	173
	500	131.71	199	131.71	196
	2500	130.17	199	130.11	192
Gene	200	485.21	800	483.6	752
	500	477.48	800	476.75	756
	2500	472.85	800	472.69	757
Hailfinder10	200	550.85	559	547.64	420
	500	523.81	559	523.26	519
	2500	511.03	559	511.03	557
Insurance10	200	210.1	269	210.1	269
	500	198.87	269	198.87	269
	2500	180.49	269	178.96	198
Link	200	618.09	723	618.09	723
	500	535.75	723	535.75	723
	2500	534.47	723	543.77	472
LungCancer	200	435.72	799	435.46	766
	500	424.69	799	424.44	783
	2500	420.47	799	420.42	784
Munin	200	42.614	188	36.987	5
	500	37.66	188	35.414	4
	2500	35.925	188	33.225	21
Pigs	200	390.75	440	390.75	440
	500	385.59	440	385.59	440
	2500	382.36	440	382.36	440

Table 5.1: Gold standard for regularization

estimation of the Kullback-Leibler divergence computed by using 50 000 observations, in realistic distribution, it is the negative log-likelihood of 5 000 observations.

5.2 Acceleration for Sparse Observations

A sparse learning set is characterised by a large proportion of observed values equal to 0 (or another constant value), for each variable. The Chow-Liu algorithm has been specialised for sparse learning set, providing a significant reduction of the time and memory required [Mei99]. The motivation for this particular algorithm was text mining, where a sample corresponds to a document and each variable $\mathcal{X}$ corresponds to a word of vocabulary that is ($\mathcal{X}$=1) or is not present ($\mathcal{X}$=0) in the document. While both N and p may in such a setting be rather large, each document typically contains only a fraction of the words, hence the sparsity.

The main points of the improved algorithms consists in the following points, for binary variables.

- $I_D(\mathcal{X};\mathcal{Y})$ is a function of $N_D(\mathcal{X}=1,\mathcal{Y}=1)$, $N_D(\mathcal{Y}=1)$ and $N_D(\mathcal{X}=1)$.
- The learning set can hence be compressed by storing only the occurrence of those values, speeding up the computation of the observed statistics mentioned in the previous point.
- If $N_D(\mathcal{X}=1,\mathcal{Y}=1)=0$, $I_D(\mathcal{X};\mathcal{Y})$ increases monotonically with $N_D(\mathcal{Y}=1)$. Therefore, it is possible to order the set of all variables $\{\mathcal{Y}_i \in \boldsymbol{\mathcal{X}}\} : N_D(\mathcal{X}=1,\mathcal{Y}_i=1)=0$ by their pairwise mutual information to $\mathcal{X}$ without computing those values. This set is large since the learning set is sparse.

This is exploited in [Mei99] to build an efficient algorithm minimizing the number of mutual information values computed.

5.3 Approximation for Large N

[PM06] proposed an approximate Chow-Liu algorithm targeted at problems with a high number of samples. Mutual information weights are no longer estimated based on the whole learning set but only on a fraction of all available samples. Confidence intervals are maintained for these estimations, and when two weights have to be relatively ordered, the algorithm exploits more samples (if needed) until the two confidence intervals no longer overlap and the query can be answered.

In order to limit the number of comparisons, they develop a special MWST algorithm (MIST), based on the fact that a MWST cannot contain any loop. MIST is described in Algorithm 5.3. The MWST of a complete weighted graph is computed by selecting an initial candidate tree, and considering all the other edges one by one. For each edge considered, a loop can be formed by using this edge and the unique path of the tree linking the extremities of this edge. The considered edge is discarded if its weight is weaker than any edge in this path; and otherwise replace in the tree the weakest edge of this path.

Algorithm 5.3 MIST algorithm [PM06]

```
Input: graph $\mathcal{G}$ with edge weights $w_e$
$\mathcal{T}$ = random tree
$\mathbf{E} = E(\mathcal{G}) \backslash E(\mathcal{T})$
repeat {Discard an edge}
  $e$ = SelectEdge($\mathbf{E}$)
  $C$ = FindCycle($\mathcal{T} \cup e$)
  $e'$ = WeakestEdge($\mathbf{C} \backslash e$)
  if $w_e > w_{e'}$ then
    $\mathcal{T} = \mathcal{T} \cup e \backslash e'$
    $e = e$'
  end if
  $\mathbf{E} = \mathbf{E} \backslash e$
until $\mathbf{E} = \emptyset$
return $\mathcal{T}$.
```

The first tree structure is the output of the Chow-Liu algorithm computed on a restricted set of samples. This computation also initializes the confidence intervals for the mutual information values. Different heuristics are proposed to select the next edge to be compared to the current tree.

5.4 Approximation for Large p

In the context of high-dimensional learning (large p and low N) no heuristic described so far is very appealing, since the bottleneck is the number of variables and the quadratic complexity of the Chow-Liu algorithm. In this section I therefore consider randomized versions of the Chow-Liu algorithm, with the aim of reducing the computational complexity in p. This can only be achieved by reducing the number of edges considered for inclusion in the model in the first stage of the algorithm.

Three strategies are considered here, by increasing sophistication: purely random structure, random edge sampling and finally cluster-based edge sampling. The last two methods were developed during this thesis.

5.4.1 Random Structure

This algorithm generates a tree structure totally at random, which may be done in $\mathcal{O}(n)$ operations [Qui89] through the use of Prüfer sequences [Prü18]. While the output structure will therefore not depend on D, remember that its parameters will still be estimated based on D. This algorithm was used to generate mixtures of Markov trees for variance reduction in [ALDW09b], see Section 6.1.5.

A Prüfer sequence is a sequence of $p-2$ integers $\in [1,p]$. For any given p, the set of Prüfer sequences is in bijection with the set of undirected trees of p nodes labeled by the integers $\in [1,p]$. This property was used by Prüfer to build a proof of Cayley's formula: the number of labeled trees on p nodes is p^{p-2} [Cay89].

These sequences can therefore be used to generate a random undirected tree, a process detailed in Algorithm 5.4. From a Prüfer sequence **A** randomly generated, the tree is iteratively constructed from an empty graph. To do so, denote by **B** the set of all p integer labels. Until **A** is empty, add an edge between its first element and the smallest element b in **B** that is not present in **A**, before removing these elements from respectively **A** and **B**.

This algorithm is of $\mathcal{O}(p \log p)$ complexity in the number of variables p, where $\log p$ comes from the complexity of the operations associated to a simple priority queue. It may be decreased by using more efficient priority queue.

Algorithm 5.4 Random tree algorithm (RT) [Qui89]

```
  **Input:** variable set $\boldsymbol{\mathcal{X}}$ of size $p$
  $\mathcal{T} = \emptyset$; b=0
  **for** $i = 1 \to p-2$ **do** {Generate Prüfer sequence}
    $\mathbf{A}[i] \sim \text{RandInteger}(1, p)$
  **end for**
  $\mathbf{B} = [1, 2, \ldots, p]$
  **repeat** {Construct corresponding tree}
    $b = \min(\mathbf{B}\backslash\text{Unique}(\mathbf{A}))$
    $\mathcal{T} = \mathcal{T} \cup \{\mathcal{X}_b, \mathcal{X}_{\mathbf{A}[1]}\} \cup \{\mathcal{X}_{\mathbf{A}[1]}, \mathcal{X}_b\}$
    $\mathbf{B} = \mathbf{B}\backslash b$
    PopFront($\mathbf{A}$)
  **until** $\mathbf{A} = \emptyset$
  $\mathcal{T} = \mathcal{T} \cup \{\mathbf{B}[1], \mathbf{B}[2]\}$
  **return** $\mathcal{T}$.
```

5.4.2 Random Edge Sampling

The random edge sampling algorithm reduces the complexity of the Chow-Liu algorithm in a naive way: by only inspecting a subset of a priori fixed size K of randomly selected pairs of variables. The weights associated to these edges are used to partially fill the matrix MI used as input to the MWST algorithm. Algorithm 5.5 describes this procedure. This Chow-Liu-like algorithm therefore operates on an incomplete matrix of mutual information values.

Algorithm 5.5 Random Edge sampling (RES) [SLW10]

```
  **Input:** $\boldsymbol{\mathcal{X}}$; $D$; integer $K \leq p(p-1)/2$
  $MI = [0]_{p\times p}$
  **for** $k = 1 \to K$ **do**
    $(i_1, i_2)$ = drawNewRandomEdge
    $MI[i_1, i_2] = MI[i_2, i_1] = I_D(X_{i_1}; X_{i_2})$
  **end for**
  $\mathcal{F}$ = MWST($MI$) {MWST computation by using e.g. [Kru56] as here}
  **return** $\mathcal{F}$.
```

The complexity of Algorithm 5.5 is loglinear in the number K of edges drawn. The graphical structure that it infers may be disconnected, and its dependence on the learning set D is increasing with the value of K.

5.4.3 Cluster-based Approach

In this section, I explore a less naive idea so as to sample the potentially interesting (i.e. of large weight) edges. The resulting two-step algorithm first builds a local structure of the problem, and then focuses on pairs of variables located close to each other in that structure. It is detailed in Algorithm 5.6 and illustrated in Figure 5.4.

The first step consists in an approximate online clustering of the variables based on their mutual information. This algorithm was inspired both by leader clustering (a cluster C_p is represented by its leader L_p) and by the analogy of the MWST problem with questions such as the nearest neighbor query or the shortest path problem defined over metric spaces. In metric spaces, such queries can sometimes be solved by sub-quadratic algorithms [IM98] exploiting the triangular inequality satisfied by distance measures: when point A is close to point B, which is far away from C, A is also likely to be far from C.

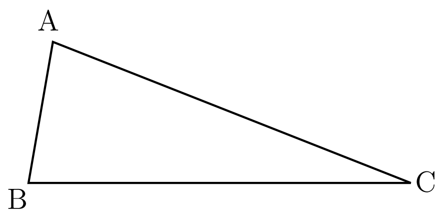


Figure 5.3: Triangle inequality

In our context, the mutual information is used to measure the “closeness” between variables, though it is not a distance measure in the mathematical sense. Still, if two pairs of variables $\{\mathcal{A}, \mathcal{B}\}$ and $\{\mathcal{A}, \mathcal{C}\}$ both have a high mutual information, then $\mathcal{B}$ and $\mathcal{C}$ may be expected to be close in this sense too. More formally, one may derive the following bound:

$$I(\mathcal{B}; \mathcal{C}) \geqslant I(\mathcal{A}; \mathcal{B}) + I(\mathcal{A}; \mathcal{C}) - H(\mathcal{A}) \ . \tag{5.11}$$

Proof.

$$\begin{aligned}
H(\mathcal{B},\mathcal{C},\mathcal{A}) &\geqslant H(\mathcal{B},\mathcal{C}) && (5.12)\\
H(\mathcal{A},\mathcal{B}) + H(\mathcal{C}|\mathcal{A}\mathcal{B}) &\geqslant H(\mathcal{B},\mathcal{C}) && (5.13)\\
H(\mathcal{A},\mathcal{B}) + H(\mathcal{C}|\mathcal{A}) &\geqslant H(\mathcal{B},\mathcal{C}) && (5.14)\\
H(\mathcal{A}) + H(\mathcal{B}) + H(\mathcal{C}) & \\
+H(\mathcal{A},\mathcal{B}) + H(\mathcal{C}|\mathcal{A}) &\geqslant H(\mathcal{B},\mathcal{C}) && (5.15)\\
&\quad + H(\mathcal{A}) + H(\mathcal{B}) + H(\mathcal{C}) \\
H(\mathcal{B}) + H(\mathcal{C}) & \\
+H(\mathcal{A},\mathcal{B}) + H(\mathcal{C},\mathcal{A}) &\geqslant H(\mathcal{B},\mathcal{C}) && (5.16)\\
&\quad + H(\mathcal{A}) + H(\mathcal{B}) + H(\mathcal{C}) \\
H(\mathcal{B}) + H(\mathcal{C}) - H(\mathcal{B},\mathcal{C}) &\geqslant H(\mathcal{B}) + H(\mathcal{A}) - H(\mathcal{A},\mathcal{B}) \\
&\quad + H(\mathcal{C}) + H(\mathcal{A}) - H(\mathcal{C},\mathcal{A}) && (5.17)\\
&\quad - H(\mathcal{A}) \\
I(\mathcal{B};\mathcal{C}) &\geqslant I(\mathcal{A};\mathcal{B}) + I(\mathcal{A};\mathcal{C}) - H(\mathcal{A}) && (5.18)
\end{aligned}$$

□

Mutual information is commonly used to cluster variables [KSAG05], although generally by computing the full mutual information matrix before any clustering is done. The goal of the algorithm considered here is however to avoid the computation of the complete set of mutual information values. To this end, it builds an online clustering of the variables based on the mutual information of only on a subset of all pairs of variables. iteratively and one at a time, until all variables belong to a cluster. This process is illustrated in Figures 5.4a to 5.4d, for 13 variables, represented by a set of points in the Euclidian plane. The distance between two points is supposed to be proportional to the mutual information of the associated variables.

A new cluster C_k is constructed around a leader (or center) variable $\mathcal{L}_k$, chosen first at random ($\mathcal{X}_5$ is chosen in Figure 5.4a), then among unclustered variables as

$$\mathcal{L}_k = \arg \min_{\mathcal{L} \in \boldsymbol{\mathcal{X}} \setminus \bigcup_{l=1}^{k-1} C_l} \sum_{l=1}^{k-1} I_D(\mathcal{L};\mathcal{L}_l) \quad \forall k > 1 \ . \tag{5.19}$$

In Figure 5.4c, the second cluster center is $\mathcal{X}_{13}$, because it is the variable the further away from (with the lowest mutual information to) $\mathcal{X}_5$.

The construction of a cluster (Algorithm 5.7) depends on two “thresholds” on the mutual information value: one cluster-threshold (I_C) and one neighborhood-threshold (I_N). The cluster is built by comparing the mutual information of each remaining unclustered variable ($\boldsymbol{\mathcal{X}} \setminus \bigcup_{l=1}^{k-1} C_l$) to $\mathcal{L}_k$. An unclustered variable $\mathcal{X}$ is categorized as:

1. member of C_k ($\mathcal{X} \in C_k$), if $I_D(\mathcal{X}, \mathcal{L}_k) > I_C$,
2. neighbor of C_k ($\mathcal{X} \in \text{neighbors}(C_k)$), if $I_C \geqslant I_D(\mathcal{X}, \mathcal{L}_k) > I_N$,
3. not related to C_k, otherwise.

The center $\mathcal{L}_k$ of a cluster always belongs to its associated cluster C_k.

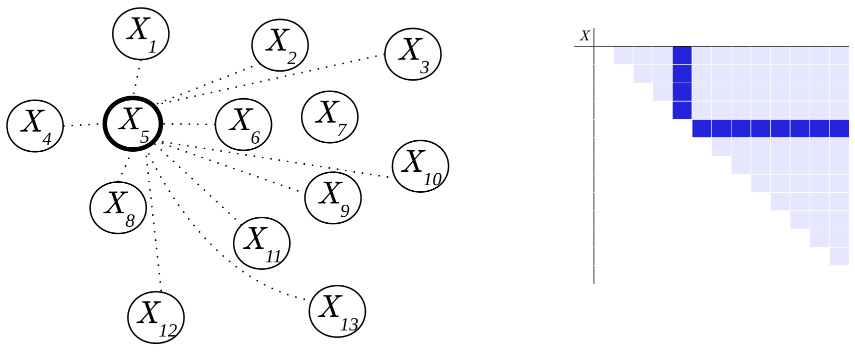
As for regularizing the Chow-Liu algorithm (Section 5.1.1), setting those thresholds over edges can be seen as excluding from the set of candidate edges pairs of potentially independent variables. Rather than a threshold, I therefore prefer to select the probability of a type 1 error associated to an hypothesis check to evaluate this independence. When independence is accepted, the mutual information is considered to be below the threshold. The hypothesis tests are based on a χ^2 distribution, and two user-specified p-values, one per threshold. These values will be respectively denoted by ρ_C (associated to I_C) and ρ_N (associated to I_N). Unless otherwise stated, these parameters are the percentile 0.005 (respectively 0.05) for I_C (I_N). These values were chosen because they are classical values used in hypothesis testing.

In the second step of the algorithm, the mutual information of all potentially interesting pairs $(\mathcal{X}_i, \mathcal{X}_j)$ are evaluated and used as edge-weights for the MWST algorithm. Interesting pairs

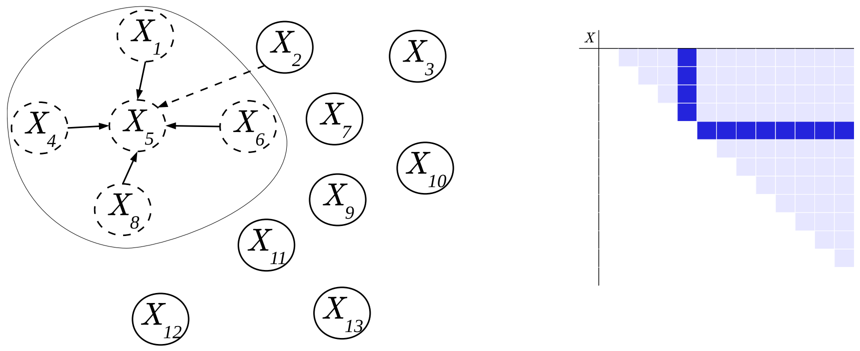
(a) are in the same cluster (Figure 5.4e),
(b) or span two neighboring clusters, i.e one variable of one cluster is a neighbor of the other cluster (Figure 5.4f).

In addition, all edges evaluated during the clustering process are used as candidate edges.

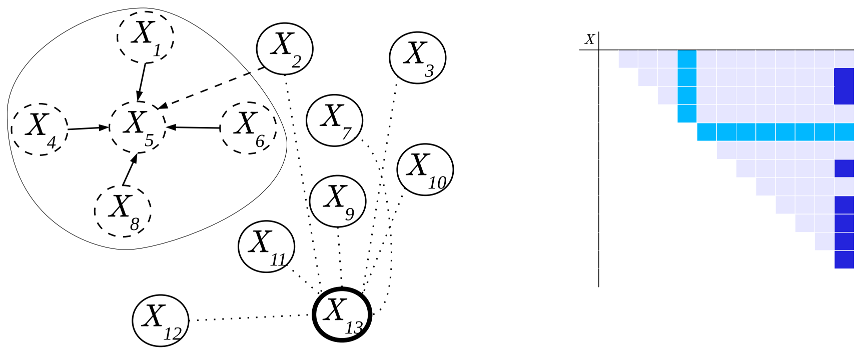
The complexity of this algorithm is between linear and quadratic in the number of variables, depending on the numerical values of ρ_C and ρ_N and the problem structure.



(a) The first cluster center ($\mathcal{X}_5$) is randomly chosen and compared to the other 12 variables.

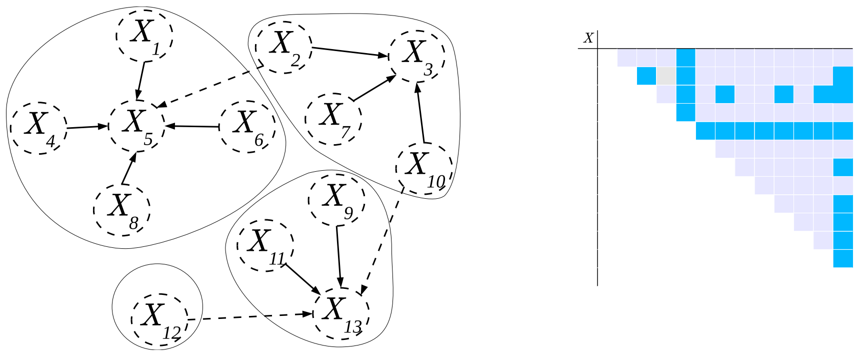


(b) C_1 is constructed: it contains 5 variables and has one neighbour.

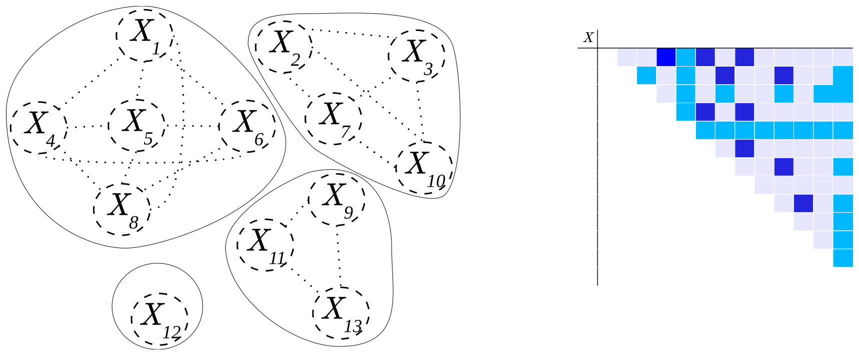


(c) C_2 is constructed around $\mathcal{X}_{13}$, the variable furthest from $\mathcal{X}_5$. The new center is compared to 7 variables only.

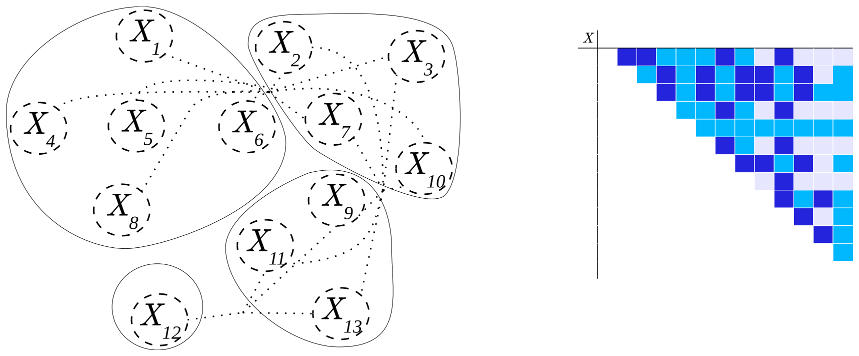
Figure 5.4: Illustration of the Cluster-based edge sampling algorithm. On the left is a set of vertices, represented as points in an Euclidian plane, on the right the matrix of mutual information. A dot-line (and a dark blue square) between two points signals the computation of a mutual information value, a full (respectively dashed) arrow that the source variable was assigned to (respectively is a neighbor of) the cluster whose center is the target of the arrow.



(d) After 4 iterations, all variables belong to a cluster.



(e) Computation of mutual information between variables of the same cluster.



(f) Computation of mutual information between variables of neighbouring clusters.

Figure 5.4: (continued) Illustration of the Cluster-based edge sampling algorithm. On the left is a set of vertices, represented as points in an Euclidian plane, on the right the matrix of mutual information.

Algorithm 5.6 Cluster-based edge sampling (CES) [SLW10]

```
**Input:** $\boldsymbol{\mathcal{X}}$; $D$; percentiles $\rho_C$,$\rho_N$
$\mathbf{C} = \emptyset$; $MI = [0]_{n\times n}$; $k = 1$
**repeat** {First step: create the clusters (Figures 5.4a to 5.4d)}
  $\mathcal{L}_k = \arg\min_{\mathcal{L}\in\boldsymbol{\mathcal{X}}\setminus\mathbf{C}} \sum_{l=1}^{k-1} I_D(\mathcal{L};\mathcal{L}_l)$
  $C_k = \text{MakeCluster}(\mathcal{L}_k, \boldsymbol{\mathcal{X}}\setminus\mathbf{C}, \rho_C, \rho_N)$ {Algorithm 5.7}
  $\mathbf{C} = \mathbf{C} \cup C_k$
  $k = k + 1$
**until** $\boldsymbol{\mathcal{X}} = \mathbf{C}$
**for** $k = 1 \to size(\mathbf{C})$ **do** {Second step: focus interesting edges}
  **for** $i_1, i_2 : \mathcal{X}_{i_1}, \mathcal{X}_{i_2} \in C_k$ **do** {Intra-cluster (Figure 5.4e)}
    $MI[i_1, i_2] = MI[i_2, i_1] = I_D(\mathcal{X}_{i_1}; \mathcal{X}_{i_2})$
  **end for**
  **for** $l = 1 \to k - 1$ **do** {Neighboring clusters (Figure 5.4f)}
    **if** $\exists\mathcal{X} \in C_k : \mathcal{X} \in Neighbors(C_l)$ **then**
      **for** $i_1, i_2 : \mathcal{X}_{i_1} \in C_k, \mathcal{X}_{i_2} \in C_l$ **do**
        $MI[i_1, i_2] = MI[i_2, i_1] = I_D(\mathcal{X}_{i_1}; \mathcal{X}_{i_2})$
      **end for**
    **end if**
  **end for**
**end for**
$\mathcal{T} = \text{MWST}(MI)$
**return** $\mathcal{T}$
```

Algorithm 5.7 MakeCluster

```
**Input:** $\mathcal{X}_{i_1} \equiv \mathcal{L}_k$, $\boldsymbol{\mathcal{V}}$, $\rho_C$, $\rho_N$
$C_k = \mathcal{L}_k$
**for** $\mathcal{X}_{i_2} \in \boldsymbol{\mathcal{V}}\setminus\mathcal{L}_k$ **do**
  $MI[i_1, i_2] = MI[i_2, i_1] = I_D(\mathcal{X}_{i_1}; \mathcal{X}_{i_2})$
  **if** $I_D(\mathcal{X}_{i_1}; \mathcal{X}_{i_2}) > CDF^{-1}_{\chi^2(|val(\mathcal{X}_{i_1})-1||val(\mathcal{X}_{i_2})-1|)}(1-\rho_C)/(2N\ln 2)$ **then**
    $C_k = C_k \cup \mathcal{X}$
  **else if** $I_D(\mathcal{X}_{i_1}; \mathcal{X}_{i_2}) > CDF^{-1}_{\chi^2(|val(\mathcal{X}_{i_1})-1||val(\mathcal{X}_{i_2})-1|)}(1-\rho_N)/(2N\ln 2)$
  **then**
    $\mathcal{X} \in \text{neighbors}(C_k)$
  **end if**
**end for**
**return** $C_k$
```

5.5 Empirical Evaluation of the Proposed Approaches

The approximations of the Chow-Liu algorithms proposed in this chapter (Section 5.4) are compared against the Chow-Liu algorithm. These two algorithms are the random edge sampling (RES) and the cluster-based edge sampling (CES) algorithms. These methods respectively take as input a number of sampled edges K for RES and two percentiles ρ_C and ρ_N for CES.

The behavior of these algorithms is first compared in terms of the number of edges considered, number of edges in common with the Chow-Liu tree, and the sum of the mutual information associated to each edge of the generated tree.

These experiments were performed on 10 DAG-200-10 distributions, i.e. a Bayesian network whose underlying graph is a DAG with maximum 10 parents for each one of the 200 binary variables (see Appendix A.1 for details). A Markov tree defined over those variables has therefore $p-1 = 199$ edges, and there are 19 900 possibles edges.

The gain in computational complexity will be measured by the run-time of the algorithms, the loss in accuracy by the number of edges common with the Chow-Liu tree and by the sum of the mutual information associated to the edges.

The comparison between those algorithms will be carried out in a configuration where they sample roughly the same number K of edges. Since this is the main determinant of the complexity ($\mathcal{O}(KN + K\log K)$) and directly relates to the quality of the approximation, both algorithms can be expected to behave similarly.

The influence of the parameters will then be discussed.

5.5.1 General Comparison

RES and CES are compared against the Chow-Liu algorithm on 10 DAG-200-10 problems times 10 learning sets.

To have a fair comparison between RES and CES, they are configured to sample approximately the same number of edges. In CES, the number of edges considered is not controlled directly. To obtain the experimental setup described above, two values are selected for ρ_C and ρ_N, respectively

	CL	CES	RES	RT
Run time	1	0.35	0.38	0.01

Table 5.2: Relative learning time (with respect to CL) of RES, CES and RT, on 10 DAG-200-10 problems, averaged on on 4 learning sets times 100 trees. A run time of 1 corresponds to approximately 1.7 seconds.

0.005 and 0.05, the mean number of edges sampled over several runs of CES is measured, and used as the parameter K for RES. The values of the parameters of CES are "arbitrary", in the sense that they were chosen based on typical values used for independence testing and not based on e.g. cross-validation.

The results are respectively displayed in Figures 5.5 (number of edges similar to $\mathcal{T}_{CL}(D)$) and 5.6 (sum of the mutual information values associated to the edges of the tree) for each of the 10 target distributions, and show the superiority of the models provided by CES over those generated by RES, for an equal number of edges: CES is consistently closer to the CL tree. Moreover, the run time of the algorithms (Table 5.2) indicates that CES is slightly faster than RES. This is plausible, since CES is a deterministic algorithm (except for the initial cluster center) while RES must perform many random draws and account for edges already used. A more clever implementation of this algorithm might however shrink the difference.

5.5.2 Effect of the Parameters

For both the random and the cluster-based edge sampling, the parameter(s) have a direct influence on the precision and the run time.

Random edge sampling

For random edge sampling, the effect of the parameter is straightforward. Run time, accuracy (with respect to the learning set) and number of edges similar to a Chow-Liu tree increase with the number of edges sampled. Run time is directly controllable through the number of edges sampled.

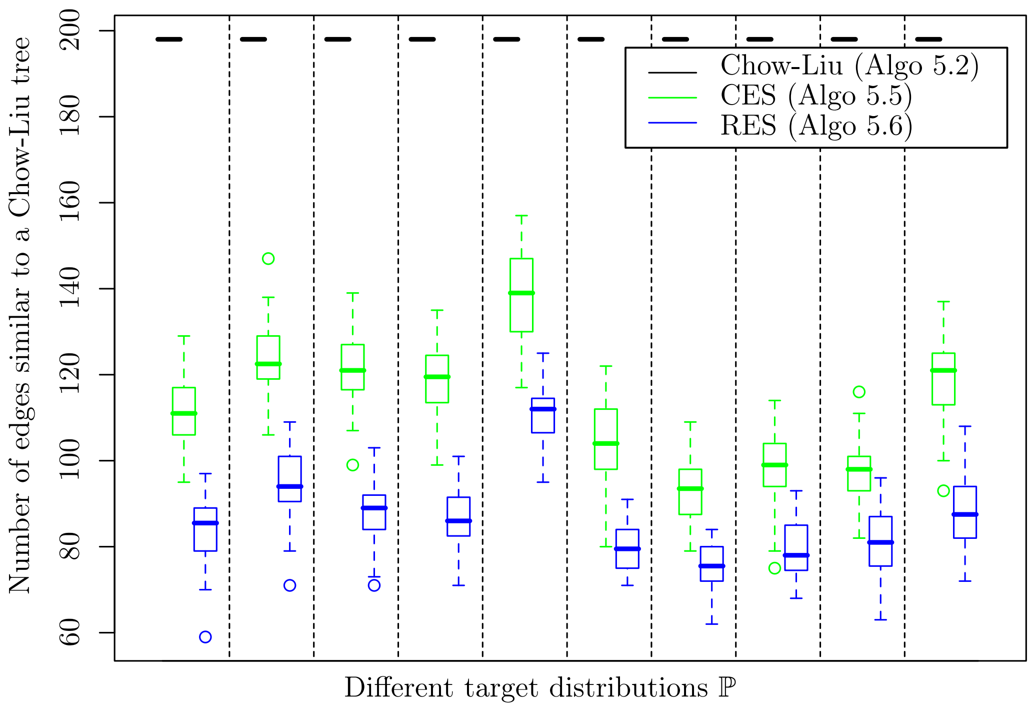


Figure 5.5: Number of common edges of RES and CES with the Chow-Liu tree (on top), on 10 DAG-200-10 target distributions, based on 4 learning sets times 10 trees for each distribution.

For a given number of edges and number of observations, the run time is independent from the learning set.

Cluster-based Edge Sampling

The two parameters of the cluster-based method have a more complicated influence on the number of edges sampled. The run time of the method is dictated by the number of edges sampled, and the precision of the resulting model also tends to increase with this number of edges.

The edges sampled by this method are composed of two subsets: the edges considered during the construction of the clusters and those considered during the exploitation of the clusters. The latter subset can be further divided into the edges located inside a cluster and the edges between neighboring clusters. The size of all subsets depends on the parameters of the

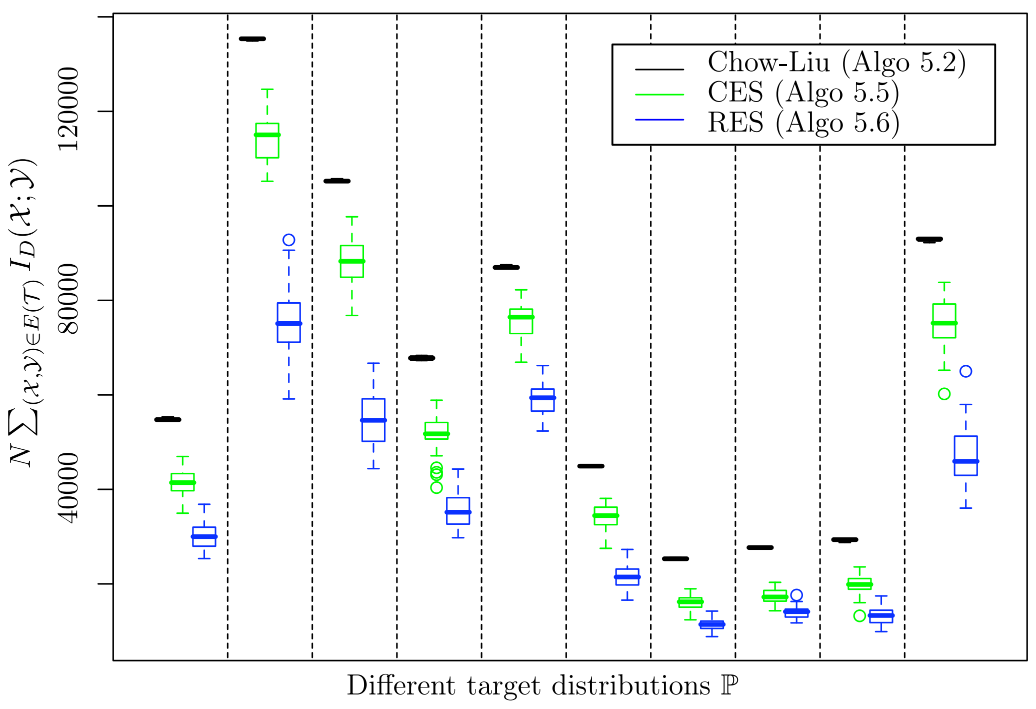


Figure 5.6: Sum of N*the mutual information (computed on the learning set) associated to the edges of the trees produced by RES, CES and CL, on 10 DAG-200-10 target distributions, based on 4 learning sets times 10 trees for each distribution.

algorithm and on the learning set.

The first parameter, ρ_C, can vary between 0 or 1. It is the only parameter influencing the first two subsets of edges mentioned in the previous paragraph.

The number of edges sampled during construction increases with the number of clusters: when a new cluster is constructed, $p' - 1$ mutual information are computed, where p' is the number of yet unclustered variables. The number of clusters monotonically decreases when ρ_C increases. When $\rho_C = 0$, independence is always accepted, there are p clusters of 1 variable and all mutual information values are computed to build the sequence of clusters. When $\rho_C = 1$, independence is always rejected, there is only one cluster and only $p - 1$ mutual information values are computed during construction.

On the other hand, the number of edges computed inside all clusters decreases with the number of variables in each cluster. This number of edges tends to evolve in a direction opposite to the number of edges sampled during construction. When $\rho_C = 1$, there are p clusters and there is no edge to compute in any cluster. When $\rho_C = 0$, there is only one cluster, all possible edges are inside this single cluster and will be considered, either during construction ($p-1$ edges) or exploitation of the edges inside clusters ($(p-1)(p-2)/2$ edges).

To summarize, at the extreme values 0 or 1 of the parameter ρ_C, all edges are considered. The value of ρ_C should therefore lie between these two extremes, so that only a subset of mutual information values are computed.

The second parameter ρ_N can vary between 0 and ρ_C: a value greater than ρ_C is equivalent to the same value as ρ_C (no independence is accepted, except if the variable is inside the cluster). The last subset of edges, those considered during exploitation and between neighboring clusters, is influenced by both parameters of the method.

ρ_N influences the number of clusters and the number of variables inside each cluster whereas ρ_C influences the number of neighborhood relationships between these clusters. The number of edges considered tends to increase with the size of each cluster: the more variables there are in two clusters, the more mutual information values will be computed if these clusters are neighbors. The number of relationships increases when ρ_C decreases (for a given ρ_N). If $\rho_C = \rho_N$, there are no neighbors, because the two independence tests select the same pairs of variables. If $\rho_C = 0$, any cluster generated is a neighbor of any other cluster. In this case, all mutual information values will be computed.

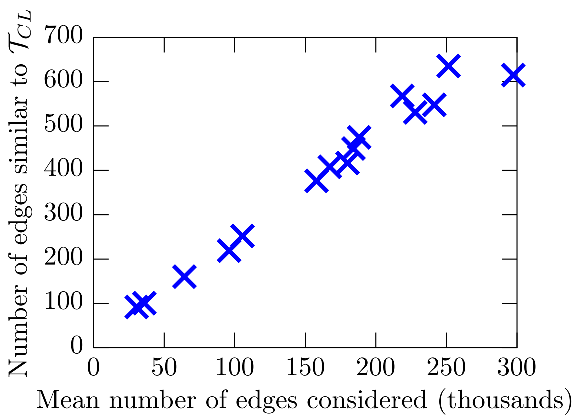
To better visualise the influence of these two parameters, the algorithm was applied on a set of 6 DAG-1000-5 target distribution times 5 learning sets (see appendix A.1), for a varying number N of observations (200, 600 and 1000). The two parameters take any possible combination of values in $\{0.1, 0.05, 0.01, 0.005, 0.001\}$. Table 5.3 lists the mean number of edges considered by CES during construction and exploitation, Table 5.4 the average number of common edges with a Chow-Liu tree, Table 5.5 the average divergence to the target distribution, and Table 5.6 the average run time of the algorithm.

The evolution of the total number of edges sampled with ρ_C and ρ_N is visible in Table 5.3. In particular, the following can be observed:

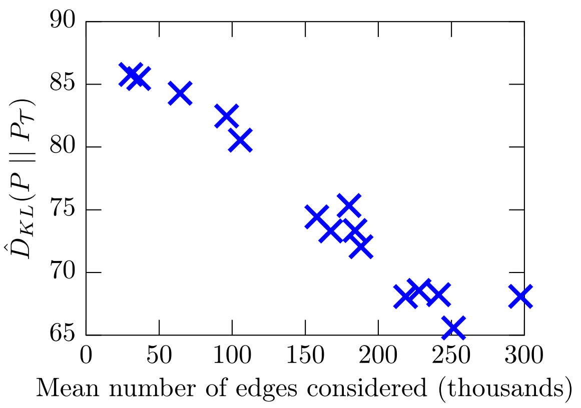
Table 5.3: Mean number of edges considered by the CES algorithm for varying ρ_C and ρ_N, in the same conditions as for Table 5.4. The three numbers in each cell are the number of edges considered during construction, during exploitation, and the sum of those two numbers.

				ρ_N		
		0.1	0.05	0.01	0.005	0.001
		9220				
	0.1	26831				
		36052				
		17474	17483			
	0.05	279874	13153			
		297349	30636			
		61496	61496	61494		
ρ_C	0.01	179881	96452	2934		
		241378	157948	64429		
		94459	94452	94437	94491	
	0.005	133422	72928	11059	1641	
		227881	167381	105496	96132	
		179423	179479	179387	179514	179427
	0.001	72109	39139	8773	4574	549
		251533	218618	188160	184089	179976

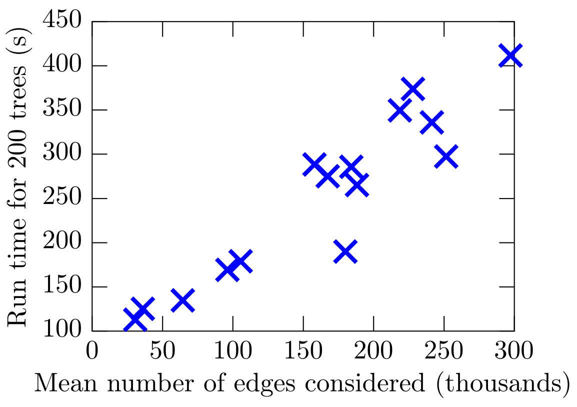
- the number of edges sampled during construction is constant with ρ_C (the first number of any cell is approximately constant in a row);
- the number of edges sampled during construction decreases when ρ_C increases (the first number of any cell decreases when going down);
- the number of edges exploited inside the clusters decreases with ρ_C (on the diagonal, $\rho_C = \rho_N$, so all edges considered during exploitation [second number of any cell] are inside the clusters);
- the number of edges exploited between clusters increases with ρ_N (on any row, the second number increases when going left).



(a) The number of edges similar to the Chow-Liu tree tends to increase with the number of edges considered.



(b) The KL divergence (evaulated using a test set) to the target distribution tends to decrease with the number of edges considered.



(c) The run time tends to increase with the number of edges considered.

Figure 5.7: Results of the CES algorithm for various parameter values, in the same conditions as for Table 5.4. Each point displayed in a graph is a value of the corresponding table.

Table 5.4: Mean number of edges similar to the Chow-Liu tree of the result of the CES algorithm for varying ρ_C and ρ_N. These numbers are an average over 5 target distribution of 1000 variables times 6 learning sets of $N = 600$ samples times 200 Markov trees. Similar results were observed for $N = 200$ or 1000.

		ρ_N				
		0.1	0.05	0.01	0.005	0.001
	0.1	100.6				
	0.05	615.0	91.9			
ρ_C	0.01	547.9	376.4	160.3		
	0.005	530.6	408.0	252.1	218.9	
	0.001	635.4	567.6	474.4	449.0	416.2

Table 5.5: Mean KL divergence to the target distribution of the result of the CES algorithm for varying ρ_C and ρ_N, in the same conditions as for Table 5.4.

		ρ_N				
		0.1	0.05	0.01	0.005	0.001
	0.1	85.5				
	0.05	68.1	85.8			
ρ_C	0.01	68.2	74.4	84.3		
	0.005	68.6	73.3	80.6	82.5	
	0.001	65.6	68.1	72.1	73.4	75.3

The results contained in the other three tables mostly depends on the number of edges considered by the algorithm. To visualise this dependency, these values are displayed in Figure 5.7 as a function of the corresponding total number of edges considered by the algorithm. Figure 5.7a corresponds to Table 5.4, Figure 5.7b to Table 5.5 and Figure 5.7c to Table 5.6. From these figures it is clear that, for the trees generated by CES:

- the number of edges similar to $\mathcal{T}_{CL}$ tends to increase with the number of edges considered,
- the divergence KL to the target distribution decreases with the number of edges considered,

Table 5.6: Indicative run time (in seconds) of the CES algorithm for varying ρ_C and ρ_N, in the same conditions as for Table 5.4. Because the experiments were run in a shared environment, the time reported corresponds to the average over all target distributions of the minimum run time for learning 200 trees on a learning set of a given size. The environment of the test differs from the one used in Table 5.2.

		ρ_N				
		0.1	0.05	0.01	0.005	0.001
	0.1	125.1				
	0.05	411.6	112.7			
ρ_C	0.01	335.9	288.4	134.8		
	0.005	373.9	275.2	179.1	169.3	
	0.001	297.4	349.5	265.2	285.8	189.8

- the run time of the algorithm increases with the number of edges considered.

5.6 Conclusion

This chapter has discussed several modifications of the Chow-Liu algorithm, that resulted either in a regularized or an accelerated variant.

Regularization is a direct competitor to the methods developed in this thesis. Therefore a golden standard has been developed to allow a comparison of the two approaches. It is based on a regularization of the number of edges in the Chow-Liu algorithm, and selects the optimal number of edges K using the test set. The score of this method is used as a bound over the score of any regularization method. It corresponds to a regularization by an oracle. The score of this method are computed and reported for the problems used as benchmarks in this thesis.

The complexity of the Chow-Liu algorithm was also analyzed: the log-quadratic term corresponds to the number of edges considered by the algorithm as building blocks for the trees. Two algorithms were developed and presented in this chapter to accelerate the Chow-Liu algorithm by considering only a subset of possible edges, trading-off accuracy for speed. The first method proposed samples a given number of edges at random while the other clusterizes the variables and then exploits this structure to target interesting edges. The former method allows a finer control of the running time of the algorithm, since it directly depends on the parameter of this method, while the second method yields more accurate models for a given number of edges. Controlling this number is however more difficult in the second method.

These results highlight the importance of the set of edges considered to build a Markov tree. The number of edges influences the complexity of the method and the precision of the resulting model on the learning and on the test set. The quality of the edges also influences the precision of the resulting model, for a given number of edges.

These two observations, and the overal understanding of the Chow-Liu algorithm and its variants brought by the present chapter, will guide the new algorithms for learning mixtures of Markov trees, discussed in the next chapter.

Chapter 6

Variance Reducing Mixtures of Markov Trees

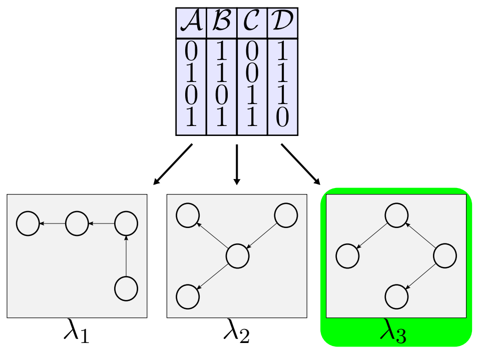
This chapter describes algorithms for building mixtures of Markov trees iteratively by specializing the meta-algorithm introduced in Chapter 4, and reproduced in Algorithm 6.1 for convenience. This meta-algorithm result from the application of the perturb and combine framework, described in Section 2.6.2, to the Chow-Liu algorithm, presented in Section 3.3.4. In the present chapter, algorithms for learning mixtures of Markov trees are constructed by specializing this meta-algorithm.

Algorithm 6.1 Meta-algorithm for learning a mixture [ALDW08, Amm10]

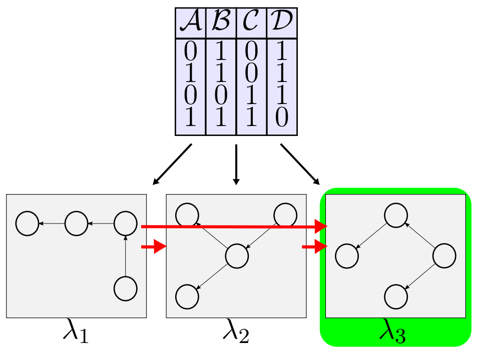
```
Input: $\boldsymbol{\mathcal{X}}$; $D$; $m$
$\boldsymbol{\mathcal{T}} = \emptyset$; $\boldsymbol{\lambda} = \emptyset$; $\boldsymbol{\theta} = \emptyset$
for $j = 1 \to m$ do
    $\boldsymbol{\mathcal{T}}[j]$ =SampleTreeStructure($\boldsymbol{\mathcal{X}}$,$D$ [,$\boldsymbol{\mathcal{T}}$])
    $\boldsymbol{\theta}[j]$=LearnParameters($\boldsymbol{\mathcal{T}}[j]$,$D$)
    $\boldsymbol{\lambda}[j]$ =LearnWeight($\boldsymbol{\mathcal{T}}[j]$,$D$)
end for
Normalize($\boldsymbol{\lambda}$)
return ($\boldsymbol{\mathcal{T}}$,$\boldsymbol{\theta}$,$\boldsymbol{\lambda}$)
```

These algorithms are developed to reduce the variance with respect to a single optimal tree learned by the Chow-Liu algorithm. They are therefore compared to the original Chow-Liu algorithm and to a gold standard regularized algorithm, another method to reduce the variance, described in Section 5.1.1.

The methods presented in this chapter are divided into two categories, based on whether the Markov tree structures are constructed independently from each other or not. This is highlighted by the optional argument $[,\boldsymbol{\mathcal{T}}]$ of the subroutine. In the first category, this argument is absent, each tree structure is independent from the others. It will therefore be denoted as the *independent Markov trees* category. In the second category, this argument is provided to the subroutine, and each tree structure is influenced by all or some of the previous structures computed. By extension, I will also include in this category algorithms exploiting information produced during the computation of previous structures rather than or in addition to the structures themselves. Since trees depend on each other, this method will be called *sequence of Markov trees*. Both categories are illustrated in Figure 6.1.



(a) Independent structures



(b) Sequence of structures

Figure 6.1: The two meta-algorithms for learning mixtures of Markov trees are differentiated based on the generation of tree structures. The Markov tree highlighted in green is being constructed.

An alternative interpretation of these two categories can be given by considering that the trees generated constitute Monte Carlo samples from a probability distribution over the space of tree structures. This probability distribution is a function of the learning set and of the subroutine SampleTreeStructure chosen. The first category of methods would sample the structures independently from each other, while the second would modify the distribution based on the previous structures.

The category of a method directly influences its possibility to be parallelized. Methods of the first category can easily be parallelized at a very high level. Since each tree structure is independent from the others, each Markov tree can be learned by a different core. For algorithms of the second category, parallelization is more difficult and depends on the dependencies between the tree structures.

Different variants of the algorithm are created by combining different versions of the three subroutines. These versions can either be the orginal or perturbations of different steps of the Chow-Liu algorithm, and different weighting strategies. Of course, a combination of methods is also possible. The perturbations will be described independently, and a list of all the methods constructed by combining them will be given before each set of experiments.

Each category is then (mostly) discussed independently from the other, in respectively Sections 6.1 and 6.2. Each section contains a description of the different algorithms used to build a Markov tree structure, and an experimental study of the resulting mixture models.

6.1 Mixtures of Independent Markov Trees

This section will cover new and existing variations of the construction of tree structures, and in particular of the chow-Liu algorithm, for mixtures of independent Markov trees. These methods are also evaluated empirically (Section 6.1.6).

All the randomizations schemes described here are meant to be used inside meta-algorithm 6.1, described in the previous section. An overview of all the methods constructed will be given before the experimental analysis. To ease the identification of the contribution of the present work, the methods that were originally developed and/or considered by me will be marked by a star *.

The Chow-Liu algorithm can be perturbed by modifying the learning set at different stages of the algorithm, the procedure learning the tree structure or the method for estimating the parameters. The vast majority of modifications were considered for the structure learning part of the method. In addition to randomizing the Chow-Liu algorithm, the selection of the weights associated to each Markov tree can also modified to yield different mixture. These four points are now discussed.

6.1.1 Perturbation of the Learning Set

Perturbing the learning set is a popular method to introduce randomization in an algorithm. The modifications considered here are all based on bootstrap aggregation, a well-known method to reduce the variance. It was already described in Section 2.6.2, and has been applied to Markov trees in [ALDW09b]. It consists in the application of an algorithm on a bootstrap replicate D', i.e. a learning set of N' samples drawn from the original set D. I will consider here the typical value $N' = N$. The complexity of constructing a replicate is therefore $\mathcal{O}(N')$.

Three variants are considered at this step of the meta-algorithm:

- both the structure and the parameters are computed based on the original learning set;
- both the structure and the parameters are computed based on a bootstrap replicate;
- the structure is computed based on a bootstrap replicate, the parameters using the original learning set.

The latter method is slightly less randomized then the second. It only randomizes the structure and uses for any structure the best parameters possible, based on the learning set.

6.1.2 Generating a Tree Structure

The algorithms considered are those presented in Chapter 5. Here is a short summary of these algorithms:

- CL (Algorithm 5.2): The original Chow-Liu algorithm, constructs the structure of a Markov tree minimizing the negative log-likelihood of the learning set;

- CES (Algorithm 5.6): cluster-based edge sampling, approximates CL by considering only some edges, sampled by constructing and then exploiting a clustering of the variables to target edges of strong weight;

- RES (Algorithm 5.5): random edge sampling, approximates CL by considering only a subset of edges, uniformly selected in the set of all $p(p-1)/2$ possible edges;
- RT (Algorithm 5.4): random tree, a tree structure randomly generated.

6.1.3 Estimation of the Parameters

Parameters learning for a Bayesian network is straightforward, and several methods were discussed in Section 3.3.3. The Laplace approximation was selected for this thesis, a choice discussed in Section 3.5.1.

While only one algorithm is used, it may take as input two different learning sets, either the full learning set, or a bootstrap replicate.

6.1.4 Weighting Scheme

Two main strategies have been considered to weight the trees in a mixture generated in the perturb and combine framework for variance reduction: uniform and "Bayesian" weights [Amm10]. As their name suggests, the uniform weighting schemes assigns the weight $1/m$ to every model while the Bayesian weight of each tree is proportional to the probability of this tree structure given the learning set:

$$\lambda_j^{\text{Bayesian}} \propto \mathbb{P}(\mathcal{T}_j|D) \ . \tag{6.1}$$

Computing $\mathbb{P}(\mathcal{T}_j|D)$ exactly is not necessary, since the weights will be normalized. In practice, the BD score, proportional to this probability and easier to compute, is used. This score was discussed in Section 3.3.5, and introduced for mixtures of Markov trees in [ALDW08].

In practice, the uniform weighting strategy is reported to usually result in a better accuracy [ALDW09b, Amm10] for a low number of samples. In my own experiments, the uniform weighting scheme resulted to a superior accuracy than Bayesian weights as well. Therefore, uniform weights will be the only strategy presented in this work. To be more accurate about the merit of each weighting scheme, tree generation strategies close to random structure seems to favor Bayesian weighting schemes whereas mixtures based on algorithms targeting good structures have a better accuracy when the weights are uniform.

A possible explanation might be that the best weighting strategy for a given tree sampling subroutine is the weighting strategy that makes the mixture the best approximation of Bayesian inference. Performing exact Bayesian inference is possible with weighting strategies, but for distinct tree sampling subroutine. Consider the following two mixtures.

- In the first mixture, the structures are sampled directly from the posterior distribution $\mathbb{P}(\mathcal{T}|D)$ and weighted uniformly.
- in the second mixture, the structures are sampled uniformly and weighted by the posterior probability.

As m tends to infinity both mixtures converge to Bayesian learning.

If the tree sampling distribution is close to Bayesian structure sampling, but the trees are weighted by their posterior probability, the resulting mixture is further away from Bayesian learning than if uniform weighting scheme is used. Using Bayesian weights if the distribution is already close to the Bayesian posterior distribution makes the peak(s) of this distribution sharper. This seems to agree with the accuracy observed in practice.

However the methods targeting structures of high posterior probability (i.e. close to Bayesian structure sampling) are usually much better than random structure sampling for finite m. There are many (p^{p-2}) tree structures over p labeled vertices [Cay89], and few have a significant posterior probability. The probability of sampling a good structure with a uniform sampling strategy is therefore very low. As a result, using a Bayesian weighting scheme most often puts a weight close to 1 on a single structure. In addition, this structure is not necessarily good, by comparison with structures generated by a subroutine that constructs with a high probability a structure close to a tree generated by the Chow-Liu algorithm.

6.1.5 Overview of Methods Considered

These different subroutines are combined to instantiate Algorithm 6.1. Each method is denoted by a different code. This code is constructed as follow:

1. "M" , for mixture;
2. a code for "CL", "CES", "RES" or "RT", depending on the base algorithm used to construct a tree structure (see Section 6.1.2);

CL	the Chow-Liu algorithm (Algorithm 5.2)
CES	Cluster-based Edge Sampling (Algorithm 5.6)
RES	Random Edge Sampling (Algorithm 5.5)
RT	Random Tree (Algorithm 5.4)

3. "-O", "-B" or "-B^2", depending on the randomization scheme applied on the learning set (see Section 6.1.1):

-O	no bootstrap
-B	bootstrap for the structure only
-B^2	bootstrap for structure and parameters.

For example, MCL-B denotes an algorithm that constructs a mixture (M) by applying the Chow-Liu algorithm (CL) on a bootstrap replicate, but learns the parameters on the original learning set (-B). This is further explained by Algorithm 6.2, which describes each algorithm corresponding to any code described above. The code is an input of the algorithm (argument "method") and configures the algorithm.

In addition, the different algorithms to construct a mixture of Markov trees evaluated in this chapter are presented together in Table 6.1. In this table, each code is related to the algorithm used to construct the structures, and to the perturbation scheme on the learning set. Moreover, each method I developed during this thesis is marked by a *.

Learning set	Tree structure			
	CL	CES*	RES*	RT
Original	CL	MCES-O*	MRES-O*	MRT-O
Bootstrap replicate	MCL-B^2		MRES-B^2	MRT-B^2
Mixed*	MCL-B*		MRES-B*	MRT-O

Table 6.1: Possible variations for the construction of one tree (both the structure and the parameters) in Algorithm 4.1 are composed of a tree structure learning algorithm and a randomization scheme of the learning set. A star indicates the contributions of this thesis.

Algorithm 6.2 Algorithms for learning a mixture of independent Markov trees

```
Input: $\boldsymbol{\mathcal{X}}$; $D$; m; method
$\boldsymbol{\mathcal{T}} = \emptyset$; $\boldsymbol{\lambda} = \emptyset$; $\boldsymbol{\theta} = \emptyset$
for $j = 1 \to m$ do
                                                    {Section 6.1.1}
  if method $= \ldots - B$ or $\ldots - B^2$ then
    $D'$ =bootstrap($D$)
    $D_{\text{struct}} = D'$
  end if
                              {SampleTreeStructure (Section 6.1.2)}
  if method $= MCL - \ldots$ then
    $\boldsymbol{\mathcal{T}}[j]$ =CL($\boldsymbol{\mathcal{X}}$,$D_{\text{struct}}$) {Algorithm 5.2}
  else if method $= MCES - \ldots$ then
    $\boldsymbol{\mathcal{T}}[j]$ =CES($\boldsymbol{\mathcal{X}}$,$D_{\text{struct}}$) {Algorithm 5.6}
  else if method $= MRES - \ldots$ then
    $\boldsymbol{\mathcal{T}}[j]$ =RES($\boldsymbol{\mathcal{X}}$,$D_{\text{struct}}$) {Algorithm 5.5}
  else if method $= RT - \ldots$ then
    $\boldsymbol{\mathcal{T}}[j]$ =RT($\boldsymbol{\mathcal{X}}$,$D_{\text{struct}}$) {Algorithm 5.4}
  end if
                                 {LearnParameters (Section 6.1.3)}
  if method $= \ldots - O$ or $\ldots - B$ then
    $\boldsymbol{\theta}[j]$=LearnParameters($\boldsymbol{\mathcal{T}}[j]$,$D$)
  else if method $= \ldots - B^2$ then
    $\boldsymbol{\theta}[j]$=LearnParameters($\boldsymbol{\mathcal{T}}[j]$,$D'$)
  end if
                                     {LearnWeight (Section 6.1.4)}
  $\boldsymbol{\lambda}[j] = 1/m$
end for
return ($\boldsymbol{\mathcal{T}}$,$\boldsymbol{\theta}$,$\boldsymbol{\lambda}$)
```

A few remarks must be made about these methods. Since the Chow-Liu algorithm always produces the same structure[1], MCL-O would produce a mixture where each tree is an identical Chow-Liu tree. This method is therefore not considered, and denoted by CL. Furthermore, MRT constructs structures independently from the learning set, so MRT-O and MRT-B are identical and denoted by MRT-O.

Existing methods were respectively proposed in [ALDW08] (MRT-O), in [ALDW09b] (MCL-B^2, MRT-B^2) and in [ALW10b] (MRES-B^2). MRT-B^2 was the most accurate of these three methods, and was found to be more accurate than CL. The new methods described were already published in [SLW10] (MCT-B*, MCES-B*, MRES-O*), although MRES-O* was also considered in [ALW10b], and in [ALSW10] (MRES-B*).

6.1.6 Experiments

The algorithms described in the previous section were applied to synthetic problems of the class DAG-1000-15[2] for settings corresponding to growing m and different number of samples ranging from 100 to 1000. The values reported here are an average of the results obtained on 10 learning sets times 10 target distributions, for each setting.

The results and conclusions of this round of experiments are presented in two parts. The interest of learning the parameters of the Markov trees on the original learning set rather than on bootstrap replicate is established in the first part. In the second part, the parameters θ_j of the models (the marginal probability distribution of each variable conditionally to its parents) are then always estimated on the original learning set. The influence of the configuration parameters on the different algorithms for constructing a tree structure are also evaluated.

[1]To be more precise, this is only true if there is no tie between the mutual information values of the edges considered. Moreover, the distribution defined on a given (undirected tree) may differ based on the root selected if Bayesian parameter estimation is used. The prior (see Section 3.3.3) is equivalent to additional samples, so the choice of the root and the orientation of the edges may modify the number of additional samples used to estimate the marginal conditional probability of each variable. However the slight differences caused by the choice of the root and the prior are usually negligible.

[2]I use the notation DAG-xxx-yyy to denote synthetic, randomly generated target probability distributions. DAG stands for “Directed acyclic graph”, xxx corresponds the number of variables and yy is the maximum number of parents of each variables. The sampling process of these distributions is described in Appendix A.1.

In the method based on the cluster-based edge sampling (CES), this algorithm was applied by using for ρ_C (respectively ρ_N) the percentile associated to 0.005 (respectively 0.05) of the χ^2 distribution. As in Section 5.5, and to assess the interest of mixtures based on CES versus those based on the random edge sampling (RES) algorithm, this latter method was constrained to sample the same percentage (35% here on average) of all possible edges as CES, except in Section 6.1.6 where the number of edges sampled is varied. The baseline is a single Markov tree built using the Chow-Liu algorithm (CL).

The accuracy of the resulting distributions $\mathbb{P}_{\mathcal{T}}$ is quantified using a Monte-Carlo approximation of the Kullback-Leibler divergence with respect to the target distribution $\mathbb{P}$:

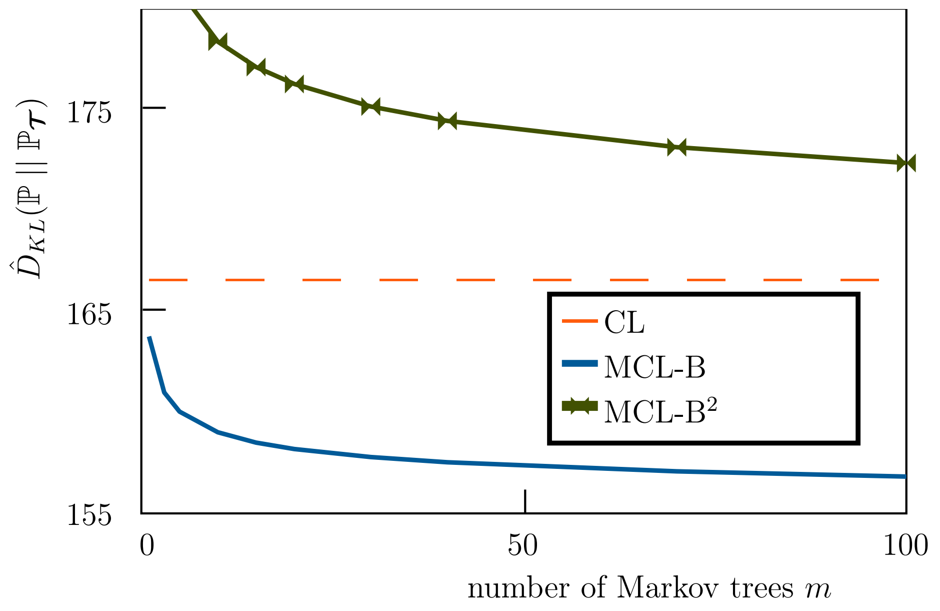
$$\hat{D}_{KL}(\mathbb{P}, \mathbb{P}_{\mathcal{T}}) = \sum_{\mathbf{x} \sim \mathbb{P}} \log \frac{\mathbb{P}(\mathbf{x})}{\mathbb{P}_{\mathcal{T}}(\mathbf{x})} \ ,$$

see Section 2.3.2 for details. Two evaluations are performed, each with a different set of 60,000 samples, so that convergence of the estimator can be evaluated.

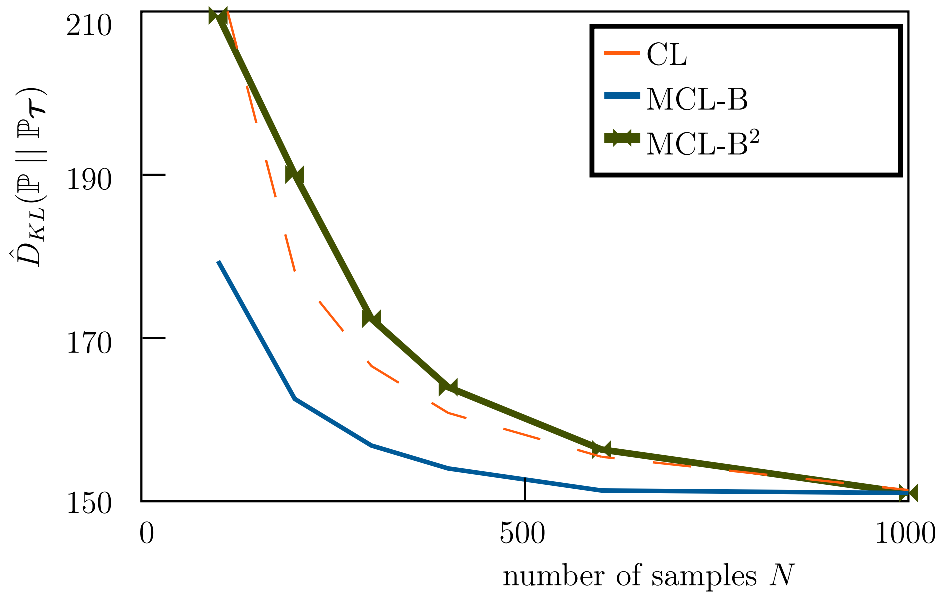
Estimation of the Parameters of the Models (MCL-B^2 vs MCL-B)

This section compares two bootstrap approaches of the Chow-Liu algorithm: the original approach where both structures and parameters of the aggregated models are learned based on bootstrap replicates (MCL-B^2) and a variant where only the structures are learned on replicates while the parameters are learned from the full learning set (MCL-B). Figures 6.2a and 6.2b illustrates the accuracy of these two methods for respectively $N = 300$ and an increasing m; and $m = 100$ and an increasing N. The Chow-Liu algorithm is provided as a reference. In both figures MCL-B appears closer to the target distribution than both CL and MCL-B^2.

MCL-B^2 is initially ($m = 1$, Figure 6.2a) less accurate than CL. My interpretation in terms of bias and variance is that learning a tree on a bootstrap replicate increases both the bias and the variance. However, MCL-B has a better accuracy than CL at the beginning of the curve. This may be due to a lower variance of single trees produced by the methods.



(a) For $N = 300$ and growing m.



(b) For growing N and $m = 100$.

Figure 6.2: MCL-B^2 is more accurate than MCL-B (averages over 10 target distributions and 10 datasets). Learning the structures based on bootstrap replicates and the parameters from the original learning set is a better strategy than bootstrapping the entire Chow-Liu algorithm.

I believe this improvement to be related to regularization. In Section 5.1.1, I explained how [LXHG+11] combines density estimates based on two independent sets of samples to compute edge scores and regularize the tree structures. The two sets of samples used in MCL-B to construct each Markov tree are not independent, but still different from each other: a bootstrap replicate only contains approximately 66% of the original samples. However,estimating mutual information on a bootstrap replicate is likely to perturb the relative ordering of the edges whose weights are very noisy. The MWST algorithm is therefore less likely to select for inclusion in the tree the edges with the highest noise over the full learning set.

For a fixed N, the accuracy of both mixtures improves with m, meaning that combining models reduces the variance. However, This decrease in the KL divergence of MCL-B^2 is however not fast enough to overtake MCL-B. Even with $m = 100$ trees, MCL-B stays ahead.

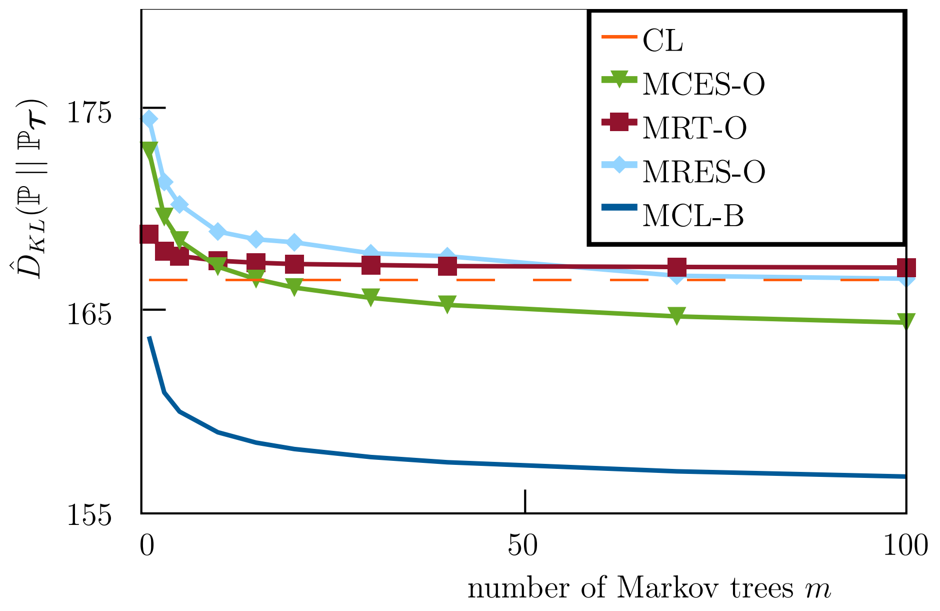
Figure 6.2*b* shows that this situation does not change with N. From $N = 100$ to $N = 1000$, where both methods are close to to the accuracy of the Chow-Liu algorithm, MCL-B is better than MCL-B^2.

Learning the parameters based on the original learning set is therefore better, and this approach will be followed for all subsequently reported experiments.

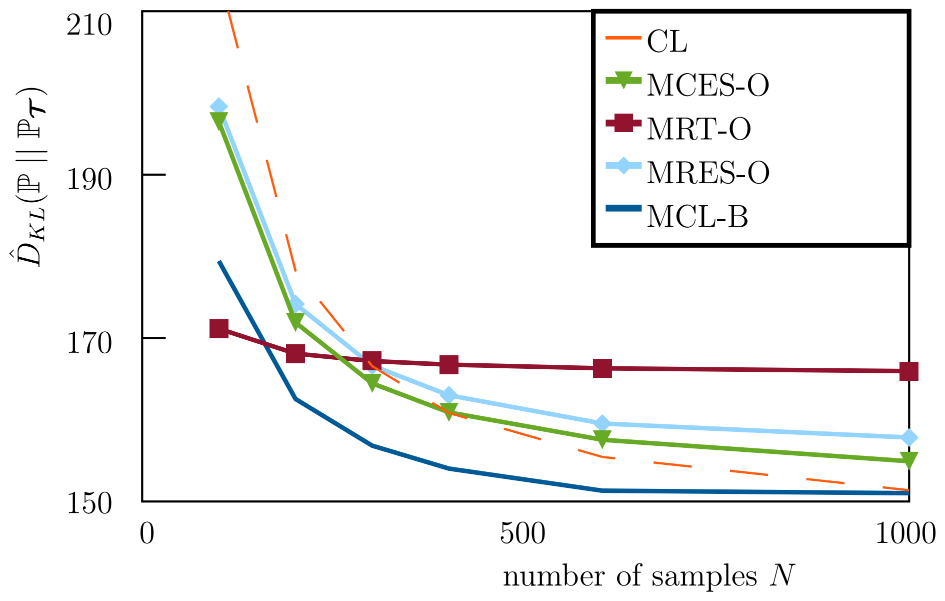
Methods Comparison (CL vs MCES-O vs MRT-O vs MRES-O vs MCL-B)

The next experiments show that the accuracy loss is in line with the gain in complexity in the algorithms: the two novel methods based on edge subsampling, cluster based edge subsampling (MCES) and random edge subsampling (MRES) are better than the less complex random structure sampling but worse than the more complex bagging of the Chow-Liu algorithm. Stronger randomization is productive when the number of samples N is much smaller than the number n of variables, which is the rule in very high-dimensional problems.

Figures 6.3 display the $\hat{D}_{KL}$ values of models built with all algorithms for growing mixture sizes m (6.3*a*), sample sizes N (6.3*b*) and degree of randomization. From Figure 6.3*a* it can be observed that the more sophisticated methods tend to converge slower. From Figure 6.3*b*, one can see that bagging yields the best performances except for $N < 50$, and that MRES-O



(a) For $N = 300$ and growing m.



(b) For growing N and $m = 100$.

Figure 6.3: Performances (averages over 10 target distributions with 1000 variables and 10 datasets), for random edge sampling (◇, light blue), cluster-based edge sampling (▽, green), tree structure bagging (dark blue), random tree structures (□, maroon), and a single Chow-Liu tree (dashed, red).

(respectively, MCES-O) outperforms the single Chow-Liu tree for $N < 200$ (respectively, $N < 400$).

From both figures it appears clearly that the more clever detection of the problem structure (the edges whose associated mutual information is larger) by the vertex clustering method (green $\bigtriangledown$) yields in all cases a worthy improvement over the naive random edge sampling strategy (light blue $\Diamond$), for a similar number of edges. While the improvement is small, it is significant and it even comes with a slightly reduced computational cost, see run time in Table 6.2. Note however that tuning the parameters of MCES-O so that it will terminate within a given computational budget is much more difficult than for MRES-O, so this latter method is still interesting.

MRT-O	MRES-O	MCES-O	MCL-B
2,063 s	64,569 s	59,687 s	168,703 s

Table 6.2: Indicative CPU times for training the 10×10 mixtures of size $m = 100$, cumulated on 100 data sets of 1000 samples (MacOS X; Intel dual 2 GHz; 4GB DDR3; GCC 4.0.1)

Influence of the parameters of the algorithms Figure 6.4 displays the effect of a modification of the number of edges used by the random edge sampling algorithm (MRES-O). Without surprise, when more edges are considered by MRES-O, the curve of the mixture is closer to the curve of the Chow-Liu method (CL).

At a low number of observations, MRES-O (with $m = 100$) is better than CL, and the fewer edges the better the mixture. For larger sample size, the opposite holds. The limit between these two distinct behaviors seems to be around $N = 250$ observations. Below that point, more diversity in the structure generated is preferable whereas with more observations considering many edges lead to a more accurate mixture.

Around this point, the MRES-O methods (with different parameters) also start achieving a worse accuracy than CL. The number of samples where a MRES-O method (for a given number of edges sampled) becomes worse than CL seems to increase with the number of edges sampled.

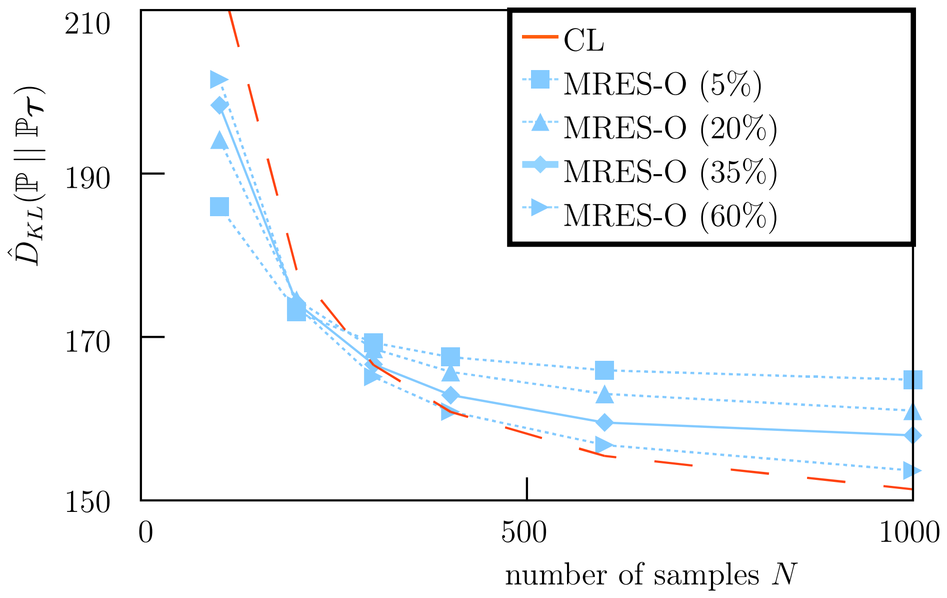


Figure 6.4: KL divergences estimated by Monte-Carlo (average values over 10 target distributions with 1000 variables and 10 datasets). Effect of the %age of randomly selected edges: 60, 35, 20, 5 ($\triangleright$, $\lozenge$, $\triangle$, $\square$) (m=100).

Significance of the results To check the soundness of the conclusions drawn previously, Figure 6.5 indicates the variability of our accuracy assessments. For the 10 target probability distributions (arranged horizontally) and each method, I show a box plot of 20 $\hat{D}_{KL}$ values obtained using 10 learning samples of size 300 times 2 different test sets of 60000 observations. The distributions are ordered by the averaged accuracy of CL for each distribution.

This figure highlights that the accuracy of the different methods can be discriminated for each distribution. The relative ordering of the methods observed previously on the averaged values remain generally unchanged. The Chow-Liu algorithm has however the worst accuracy on 8 out of 10 target distributions, while on the averaged results CL is not the worst method. This is because these results are highly influenced by the target problem with the largest gap between the accuracies of the different methods, and because in this target problem (the 5th from the left in Figure 6.5, and to a lesser extent the 3rd one) CL is not the worse method. This however does not invalidate the analysis performed before on the averaged results of the

10 distributions: the analysis would be similar if it was carried out on any single distribution out of those 10. However, the values of N where two accuracy curves cross would vary for each distribution. In particular, as N increases from 100 to 1000, the relative ordering of the methods evolve as illustrated by Figure 6.3b.

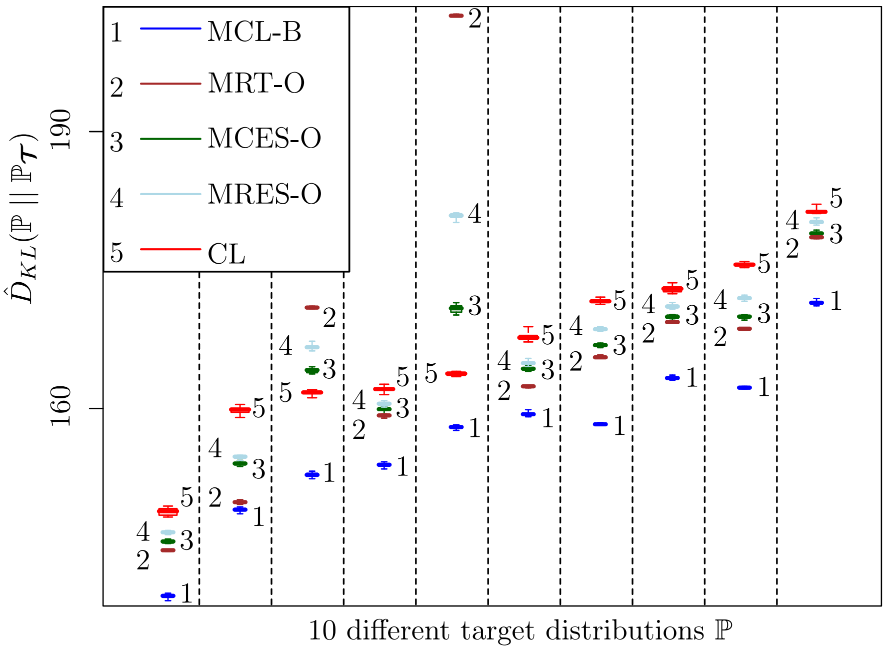


Figure 6.5: KL divergences estimated by Monte-Carlo (boxplots values over 10 learning sets times 2 test sets, for 10 target distributions with 1000 variables). Variabilities of accuracies by target distribution (m=100, N=300).

6.2 Mixtures as a Sequence of Markov Trees

In this section the tree structures are no longer constructed independently from each other, but as a sequence, sharing information across trees. As in Section 6.1, these structures are combined in a mixture by the procedure described by meta-Algorithm 6.1.

In the previous section, it was shown that a mixture of Markov trees can be better than a single Chow-Liu tree (CL), that selecting good edges is important, and that the best performing mixture was the mixture of bagged Chow-Liu trees (MCL-B). However, this method requires the computation of m Markov trees by the Chow-Liu algorithm. The extra computational cost with respect to learning one single Chow-Liu tree may prove problematic on very large problems, so attempts were made to decrease the complexity by considering only a subset of edges in each instance of the chow-Liu algorithm. Although this successfully sped up the algorithms, the accuracy was also decreased. Exploring all (or at least most) good edges therefore seems necessary to reach a good accuracy.

In this section are discussed methods that try to speed up the computation of the current tree structure by exploiting the computation of previous trees of the mixtures. In order to retain a good accuracy, these methods try to share information between trees to target good edges. In particular, these methods focus on approximating the bootstrap procedure MCL-B, that considers all edges and was the best mixture so far. Therefore, the methods developed here will mostly be compared against MCL-B (and not against other mixtures of independent Markov trees) since it tends to be the best performing method in the experiments of Section 6.1.6.

Moreover, I will only consider learning the structures of Markov trees on a bootstrap replicate and their parameters on the full learning set (M…-B methods). Indeed, [ALW10b] and complementary experiments reported in [ALSW10] showed that bootstrapping the structure (M…-B methods) is usually better than applying a randomized structure learning algorithm on the full learning set (M…-O methods).

The present section will first cover one existing method based on the concept of sharing information between trees [ALW10b], called inertial search heuristic, and abbreviated here by ISH. It extends the random edge subsampling algorithm (MRES-O) presented in Section 6.1 by considering the edges of the previous tree structure in addition to a set of edges randomly sampled.

Next, I will describe two alternative methods I developed. These two approaches exploit the first application of the Chow-Liu algorithm to gather information about the interesting edges. The two methods developed are respectively based on subsampling the set of candidate edges (as in MRES-O), or on a statistical test to detect irrelevant edges, and both methods avoid considering all candidate edges in the subsequent runs of the Chow-Liu algorithm on subsequent bootstrap replicates. These two methods are respectively presented in Sections 6.2.2 and 6.2.3, and were introduced in

[SAL+11]. The second approach can be seen as applying a structural regularization to identify a skeleton comprising only potentially relevant edges, and then restricting the search of optimal Markov trees on subsequent bootstrap replicates within that smaller envelope instead of the complete set of all possible edges. It may hence also be beneficial in terms of accuracy in very small sample size conditions.

Finally, these methods are empirically evaluated on synthetic and well-known learning sets, and the results are analyzed in Section 6.2.5.

6.2.1 Inertial Search of Mixtures of Markov Trees (MISH-B)

This algorithm [ALW10b] was designed to improve the computational complexity of the bagging of Chow-Liu trees method (MCL-B), i.e. $\mathcal{O}(p^2 \log p)$. This improvement is achieved by limiting to a specified number K the number of variable pairs and mutual information computed and considered for each MWST construction, as does MRES-B. Unlike in MRES-B however, constructing $\mathcal{T}$ is done in MISH-B by a sequential procedure (see Algorithm 6.3 for a specification):

- for the first tree, a random subset of K edges is considered,
- and then for each subsequent tree $\mathcal{T}_i$, the considered subset is initialized by the edges of the previous tree, $\mathcal{S} = E(\mathcal{T}_{i-1})$, and completed with an additional random subset of $K - |E(\mathcal{T}_{i-1})|$ edges.

The sequence of tree structures generated by this approach is thus a Markov Chain: each structure depends on the edges of the previous tree. The parameter K controls the computational complexity of the method, which is $\mathcal{O}(mK \log K)$. I will consider for this parameter values of the form $K = Cp \ln p$ as suggested in [ALW10b] (with $C = 1$ in most of our simulations) leading to a complexity approximately of $\mathcal{O}(mp \log(\log p))$. It is therefore almost loglinear in the number of variables.

Algorithm 6.3 Inertial search heuristic for Markov trees (MISH-B) [ALW10b]

```
    Input: $\boldsymbol{\mathcal{X}}$; $D$; $m$; $K$
 2: $\boldsymbol{\mathcal{T}} = \emptyset$; $\boldsymbol{\lambda} = \{1/m\}_{i=1}^{m}$; $\boldsymbol{\theta} = \emptyset$
    for $j = 1 \rightarrow m$ do
 4:    $D'$ =bootstrap($D$)
       $\mathcal{S} = \emptyset$
 6:    $MI = [0]_{n \times n}$
       if $\boldsymbol{\mathcal{T}} \neq \emptyset$ then
 8:       $\mathcal{S} = E(\boldsymbol{\mathcal{T}}[j-1])$
       end if
10:    for $k = 1 \rightarrow |\mathcal{S}|$ do
          $(i_1, i_2) = GetIndices(\mathcal{S}[k])$
12:       $MI[i_1, i_2] = MI[i_2, i_1] = I_{D'}(X_{i_1}; X_{i_2})$
       end for
14:    for $k = |\mathcal{S}| + 1 \rightarrow K$ do
          $(i_1, i_2)$ = drawNewRandomEdgeUniformly()
16:       $MI[i_1, i_2] = MI[i_2, i_1] = I_{D'}(X_{i_1}; X_{i_2})$
       end for
18:    $\boldsymbol{\mathcal{T}}[j]$ = MWST($MI$)
       $\boldsymbol{\theta}[j]$=LearnParameters($\boldsymbol{\mathcal{T}}[j]$,$D$)
20: end for
    return ($\boldsymbol{\mathcal{T}}$,$\boldsymbol{\theta}$,$\boldsymbol{\lambda}$)
```

6.2.2 Warm-start Inertial Search Heuristic (MWISH-B)

While Algorithm 6.3 is of log-linear complexity in p and gradually improves as new trees are added to the model, it consists essentially in an exploration of the matrix MI of mutual information. Without bagging (i.e. by using $D' = D$ at all iterations), this algorithm would eventually converge to the Chow-Liu tree, since Tarjan's red rule [Tar83] implies that the lightest edge of any cycle is not part of the MWST. However, the number of iterations needed to fully explore the matrix essentially increases with p, since

$$\lim_{p \to \infty} \frac{\text{Edges considered at each iteration}}{\text{Total edges}} = \frac{\mathcal{O}(p \log p)}{\mathcal{O}(p^2)} = 0 \ , \tag{6.2}$$

and hence the algorithm will take longer and longer to converge as p increases (see also the experimental results of Section 6.2.5).

Therefore I modified this method, by changing the first iteration so as to compute the Chow-Liu algorithm. This directly explores the full matrix of mutual information values, and starts the sequence with a very good set of edges ($E(T_{CL})$); see Algorithm 6.4).

Algorithm 6.4 Warm start inertial research procedure (MWISH-B) [SAL+11]

```
Input: $\boldsymbol{\mathcal{X}}$; $D$; $m$; $K$
$\boldsymbol{\mathcal{T}} = \emptyset$; $\boldsymbol{\lambda} = \{1/m\}_{i=1}^{m}$; $\boldsymbol{\theta} = \emptyset$
for $j = 1 \to m$ do
  $D'$ =bootstrap($D$)
  if $j = 1$ then
    return Chow-Liu($\boldsymbol{\mathcal{X}}$,$D'$)          {equivalent to $K = p(p-1)/2$}
  else
    lines 6 to 18 of Algorithm 6.3.
  end if
  $\boldsymbol{\theta}[j]$=LearnParameters($\boldsymbol{\mathcal{T}}[j]$,$D$)
end for
return ($\boldsymbol{\mathcal{T}}$,$\boldsymbol{\theta}$,$\boldsymbol{\lambda}$)
```

The complexity of this method is $\mathcal{O}(p^2 \log(p) + mK \log(K))$ where K is the number of edges considered at each iteration after the first one. As in Algorithm 6.3, $K = p \ln p$ by default. In practice the gain in convergence speed strongly compensates for the increased complexity needed for computing the first term (see experimental results of Figure 6.8 and Table 6.5).

Alternatively, both methods could be viewed as a stochastic walk in the space of Markov tree structures that at convergence will attain the set of good structures. Algorithm 6.3 however starts very far from this set while the variant I proposed in Algorithm 6.4 starts from a more sensible initial guess, losing the loglinear complexity. This method therefore lies between bagging and the inertial approach: the method is quadratic as the former, but only a loglinear term in p is multiplied by the number of terms.

Another possible variant of this method is obtained by always combining randomly sampled edges with the edges of the first tree, rather than with those of the previous one. This particular variant will be denoted by MWSI'.

6.2.3 Skeleton-based Mixtures of Markov Trees (Mskl-B)

This algorithm constitutes an alternative approximation to the mixture of bagged Chow-Liu trees (MCL-B). Mskl-B was also developed with the aim of speeding up learning without sacrificing accuracy. As in the previous approximation methods considered so far, this new method considers only a subset of edges. Unlike those previous method, the set $\mathcal{S}$ of edges considered by Mskl-B in constructed deterministically. More specifically, this new method

- uses a filtering step obtained as a by-product of computing a first Markov tree (using the full learning set D) to construct a set $\mathcal{S}$ of strong edges (i.e. with a high associated mutual information),
- and only considers this set $\mathcal{S}$ of edges for subsequently generated trees, so as to avoid considering poor candidate edges. Because $\mathcal{S}$ is determined once and for all, the set of candidates edges is identical for all trees. Therefore, subsequent trees can only be built using bootstrap replicates and not the full learning set.

This process is detailed in Algorithm 6.5.

$\mathcal{S}$ contains strong edges rather than weak edges. Indeed, strong edges are more interesting to consider, because the Chow-Liu algorithms will build a maximum-weight spanning tree, and as shown in Section 5.5 and 6.1.6, consider strong edges lead to better Markov trees. Moreover, if the difference between two edges is big, the relative ordering of those edges with respect to their weight will probably stay the same, even when those weights are perturbed by computing them on a bootstrap replicate. Therefore, the strongest edges have a higher probability to be in a Markov tree constructed by considering all edges than a weak edges.

Since the Chow-Liu algorithm computes the mutual information value of all edges, $\mathcal{S}$ can be constructed as a byproduct of this algorithm. The computations performed for building the first tree on the full data set are used to identify the pairs of variables whose mutual information is high, that is above a threshold. $\mathcal{S}$ contains those latter pairs of variables. Only

Algorithm 6.5 Skeleton-based approximation of bagged Chow-Liu trees (Mskl-B) [SAL+11]

```
**Input:** $\boldsymbol{\mathcal{X}}$; $D$; $m$; $\rho$
$\boldsymbol{\mathcal{T}} = \emptyset$; $\boldsymbol{\lambda} = \{1/m\}_{i=1}^{m}$; $\boldsymbol{\theta} = \emptyset$
$\mathcal{S} = \emptyset$
                                                            {1st tree}
$MI = [0]_{p \times p}$
**for** $i_1 = 1 \to p-1$ **and** $i_2 = i_1 + 1 \to p$ **do**
  **if** $I_D(\mathcal{X}_{i_1};\mathcal{X}_{i_2}) > CDF^{-1}_{\chi^2(|val(\mathcal{X}_{i_1})-1||val(\mathcal{X}_{i_2})-1|)}(1-\rho)/(2N\ln 2)$ **then**
    $MI[i_1,i_2] = MI[i_2,i_1] = I_D(X_{i_1};X_{i_2})$
    $\mathcal{S} = \mathcal{S} \cup (i_1,i_2)$
  **end if**
**end for**
$\boldsymbol{\mathcal{T}}[1]$ = MWST($MI$)
$\boldsymbol{\theta}[1]$=LearnParameters($\boldsymbol{\mathcal{T}}[1]$,$D$)
                                                            {subsequent trees}
**for** $j = 2 \to m$ **do**
  $D'$ =bootstrap($D$)
  **for** $k = 1 \to |\mathcal{S}|$ **do**
    $(i_1,i_2) = GetIndices(\mathcal{S}[k])$
    $MI[i_1,i_2] = MI[i_2,i_1] = I_{D'}(X_{i_1};X_{i_2})$
  **end for**
  $\boldsymbol{\mathcal{T}}[j]$ = MWST($MI$)
  $\boldsymbol{\theta}[j]$=LearnParameters($\boldsymbol{\mathcal{T}}[j]$,$D$)
**end for**
**return** ($\boldsymbol{\mathcal{T}}$,$\boldsymbol{\theta}$,$\boldsymbol{\lambda}$)
```

structures spanning the graph composed of the edges in S will be considered, and these models, including the first one, can thus be Markov forests rather than trees.

Considering edges whose mutual information is above a threshold is equivalent to a structural regularization of the Chow-Liu method, so as to penalize model complexity in terms of its number of edges by modifying its optimization criterion, a process already discussed in Section 5.1.1. So, and as in the cluster-based edge sampling (CES) algorithm (Section 5.4.3), the comparison to a threshold is replaced by a hypothesis test to check the independence of each pair of variables. Only the edges where independence is rejected are included in $\mathcal{S}$. This test is performed by comparing the quantity $2p(\ln 2)I_D(X_i;X_j)$ (χ-square distributed under independence, with a degree of freedom of $(|Val(X_i)|-1)(|Val(X_j)|-1)$ ito a critical value depending on a postulated p-value, say $\rho = 0.05$ or smaller.

If the quantity $2N(\ln 2)I_{\mathbf{D}}(X_i;X_j)$, estimated from the full set of observations considered by the method, is smaller than the χ-square statistic threshold computed for ρ, an arc between X_i and X_j will not be in $\mathcal{S}$ and will never be included in the forest by the modified algorithm. The size of S and thus the run-time of the algorithm depends on both ρ and this first data set, and is thus not directly controllable, unlike in the inertial methods (MISH-B and MWISH-B). It is however straightforward to modify the skeleton approach (Mskl-B) so that it takes as input a given number of edges K and select the best K edges. To identify the best K edges, compute the matrix of mutual information values (or the associated $p-value$), then make $\mathcal{S}$ the set of the edges associated to the K largest element of the matrix MI.

The complexity of Algorithm 6.5 is $\mathcal{O}(p^2+mK(\rho)\log(K(\rho))$, i.e. similar to that of Algorithm 6.4 (where $K = p\ln p$): its first term is also independent of the mixture size m and its second term now depends on the effect of the chosen value of ρ on the number of candidate edges $K(\rho) \equiv |\mathcal{S}|$ retained in the skeleton (the smaller ρ, the smaller $K(\rho)$, in a learning set dependent fashion).

6.2.4 Overview of the Methods Considered

The algorithms evaluated in the upcoming experiments (Section 6.2.5) are listed in Table 6.3.

The convention for the names of the methods is similar to what was used for mixtures of independent trees (see Table 6.1). In particular, a * identifies each contribution of this thesis. MCL-B* is the best method of independent trees, the one the approximations are trying to speed-up. It is therefore included here for comparison. MISH-B is a competing method. The two main methods presented here (MWISH-B* and Mskel-B*) led to two publications [SAL$^+$11, SAL$^+$12]. Slight modifications into a few methods led to the last 3 methods. The goal of these three methods is to highlight a specific point regarding the main algorithms during the experiments. These methods will be described in the experiments.

method	name	section
CL	Chow-Liu algorithm	5.1
rCL	“Gold standard” for regularization	5.1.2
MCL-B*	bagging of Markov trees	6.1.1
MISH-B	inertial search heuristic	6.2.1
MWISH-B*	warm-start inertial search heuristic	6.2.1
Mskl-B*	skeleton-based approximation	6.2.3
MWISH’-B*	variant of MWISH	6.2.5
MWISH”-B*	variant of MWISH	6.2.5
MCL”-B*	variant of bagging of Markov trees	6.2.5

Table 6.3: These algorithms construct mixtures of Markov trees or forests (or a single model, for the first two reference methods) They are evaluated in experiments described in the following subsections.

The algorithms for generating a sequence of Markov trees presented in the previous sections are used as subroutine in the meta-algorithm 4.1. Since the goal of these methods is to approximate MCL-B, they are associated to weighting scheme and the perturbation of the learning set that worked best with bagging, namely uniform weights and the mixed bootstrap scheme where the structure is learned on a bootstrap replicate while the parameters are estimated based on the original learning set.

In addition to MCL-B*, these approaches will also be compared to the Chow-Liu algorithm (CL, Algorithm 3.1) and to the “gold” standard for regularization (rCL, Section 5.1.1).

6.2.5 Evaluation on Synthetic Problems

The mixtures of sequences of Markov trees are first evaluated on synthetic DAG-200-5 or five DAG-1000-5 problems (see Appendix A.1 for details) and 200, 600 and 1000 samples. For each number p of variables and each number N of samples, 6 learning sets times 5 target distributions (the same distributions for any N) were considered, and the values reported correspond for any learning algorithm corresponds to an average of these 30 values. 50000 independent observations were used to estimate the Kullback-Leibler divergence of the models constructed to the target probability distribution. The same set was used for all evaluation based on a given target distribution.

Results in Terms of Accuracies

Let us start by an evaluation of the relative accuracy performances of the different algorithms in the case of $p = 1000$ variables and $N = 200$ observations, the configuration with the highest ratio p/N considered. Figure 6.6 displays the accuracy of the different mixtures of trees learning algorithm as a function of the mixture size m (horizontal axis). It also displays the accuracy of a single model for the Chow-Liu algorithm (CL) and the “gold standard" for regularization (rCL). The skeleton-based approximation (Mskl-B, Algorithm 6.5) is tested here with 4 values of its parameter ρ: $1E^{-1}, 5E^{-2}, 5E^{-3}, 5E^{-4}$.

Looking first at $m = 1$ (initial values of all curves), we observe that all but two mixtures start at the same point as the CL tree: the inertial search heuristic (MISH-B) is significantly worse, while the first tree of the strongly regularized skeleton approximation (Mskl-B) at $\rho = 5E^{-4}$ is significantly better[3].

For larger values of m, all mixtures considered monotonically improve, some more quickly than others, and for sufficiently high values of m they are all quite superior to a single CL tree. For the differently parameterized Mskl-B methods, the smaller ρ, the lesser the improvement rate. Actually, for the mixture corresponding to $\rho = 5E^{-4}$, the improvement rate of the accuracy is so small that it this mixture quickly overtaken by the mixture of Bagged Chow-Liu trees (MCL-B) (at $m = 30$) and later on by Mskl-B with $\rho = 5E^{-2}$ (at $m = 60$). On the other hand, the warm-start inertial search heuristic (MWISH-B) and Mskl-B for ρ sufficiently large display comparable performances and the same convergence rate as MCL-B.

These results confirm the superiority of the model averaging approach with respect to Chow-Liu. They also suggest the interest of trying to limit the complexities of the individual trees in the skeleton-based approximation method (Mskl-B) and to correctly initialize the inertial approach, given their computational complexity advantage with respect to raw bagging (see Section 6.2.5).

The different mixtures however still fall short from the accuracy of the regularized model in this particular configuration. My interpretation in terms of bias/variance is that variance is in this problem setting an impor-

[3]Standard deviations of KL divergences, not reported for the sake of legibility, are about 10 times smaller than the average differences commented on.

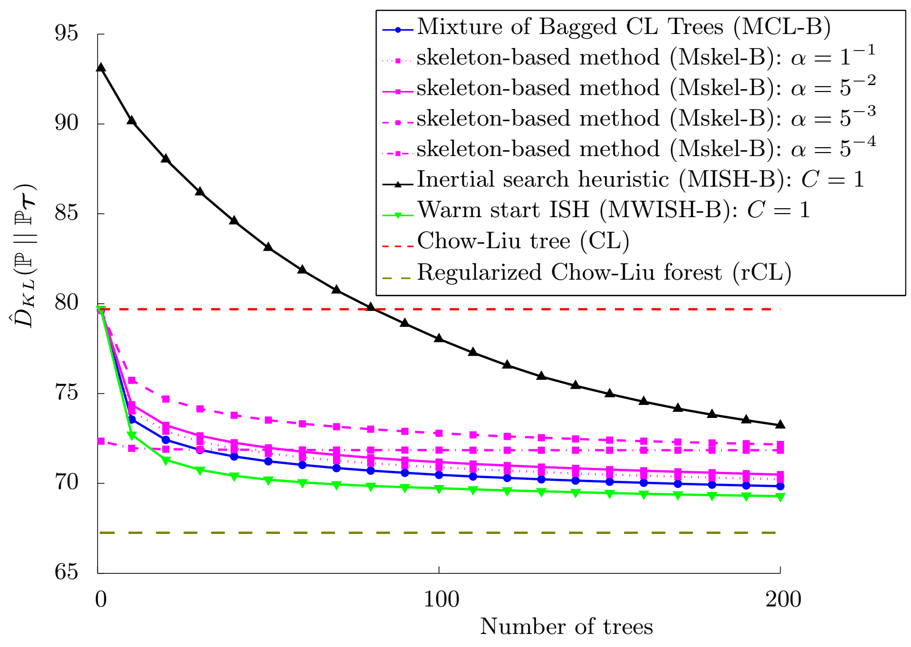


Figure 6.6: A comparison between all methods presented in this section shows the good quality of the approximation of the new model averaging methods (with $p = 1000$ and $N = 200$). Horizontal axis: ensemble size m; vertical axis KL divergence to the target density estimated by Monte-Carlo.

tant component of the error, because of the low number of samples. Regularization is performed optimally in rCL, and thus reduces this variance very well, at a level that mixtures cannot match quickly (because, unlike rCL, the parameters of the mixtures learning algorithm are not optimized on the test set). However a decrease of the accuracies of all mixtures with m is still noticeable on the graph. So for a much larger m, mixtures may achieve the same level of accuracy as rCL. Regularizing each tree in MCL-B may however be a faster method to reach or even surpass the accuracy of an optimally regularized model. In most other problem configurations considered in these experiments however, the regularized model is quickly outperformed by the mixtures, as will be seen in Sections 6.2.5 and 6.2.6.

Table 6.4: Impact of the parameter ρ on the number of edges in Mskl-B, averaged on 5 densities times 6 data sets for $p = 1000$ variables and $N = 200$ samples

	Numbers (% of the total) for $\rho =$			
	$1E^{-1}$	$5E^{-2}$	$5E^{-3}$	$5E^{-4}$
Edges in $\mathcal{T}_1$	998	997.9	993.2	626.8
Edges in $\mathcal{S}$	52278(10.5%)	26821(5.36%)	3311(0.66%)	683 (0.13%)

Effect of the Parameter ρ of Mskl-B

In order to allow a better understanding of the influence of ρ on the behavior of Mskl-B, Table 6.4 lists the average number of edges in the skeleton $\mathcal{S}$ (absolute value and % with respect to the maximum $p(p-1)/2 = 499500$) and in the first tree $\mathcal{T}_1$ of any mixture for different values of ρ. It comes as no surprise that those numbers are decreasing with ρ. Note how the number of edges in $\mathcal{T}_1$ is almost at the maximum ($p - 1 = 999$) for $\rho \in \{1E^{-1}, 5E^{-2}, 5E^{-3}\}$, whose curves start at the same performance as the CL tree, while the smallest ρ ($5E^{-4}$) leads to a much smaller tree.

For any value of ρ, including additional models in the mixture increases accuracy. However, this variance reduction effect diminishes with ρ. For $\rho = 4E^{-4}$, the skeleton has only a few edges more than the first tree. All tree structures of the mixture are really similar to each other, resulting in a very fast convergence in accuracy. The accuracy of the mixture is therefore very close to the accuracy of the initial forest. Reducing ρ also leads to an increase in the bias of the mixture: at $m = 200$ the accuracy is better for higher ρ.

My interpretation in terms of bias and variance is that the apparently atypical behavior of the curve for $\rho = 4E^{-4}$ is caused by the low size of $\mathcal{S}$, that strongly constrains the structure and the number of parameters of each term. The initially better accuracy is due to the large reduction in variance, larger than the increase in bias. For mixtures where each term has more freedom (larger ρ), the bias is initially smaller and the variance larger. Increasing the size of the mixture lowers this variance, and they overtake the mixture based on a smaller ρ.

Effect of the Learning Set Size N.

To further analyze the relative behavior of the different methods, I increased the number N of observations to 600 and 1000. The most noticeable change in the results, reported in Figures 6.7(d,e), is that the different methods now start from more diverse initial points: the CL tree becomes initially better than a tree learned on a bootstrap replicate. The advantage of the first Mskl-B tree with the smallest value ρ is decreasing. I deem that both observations are a consequence of the improved precision of the mutual information estimate derived from a larger data set.

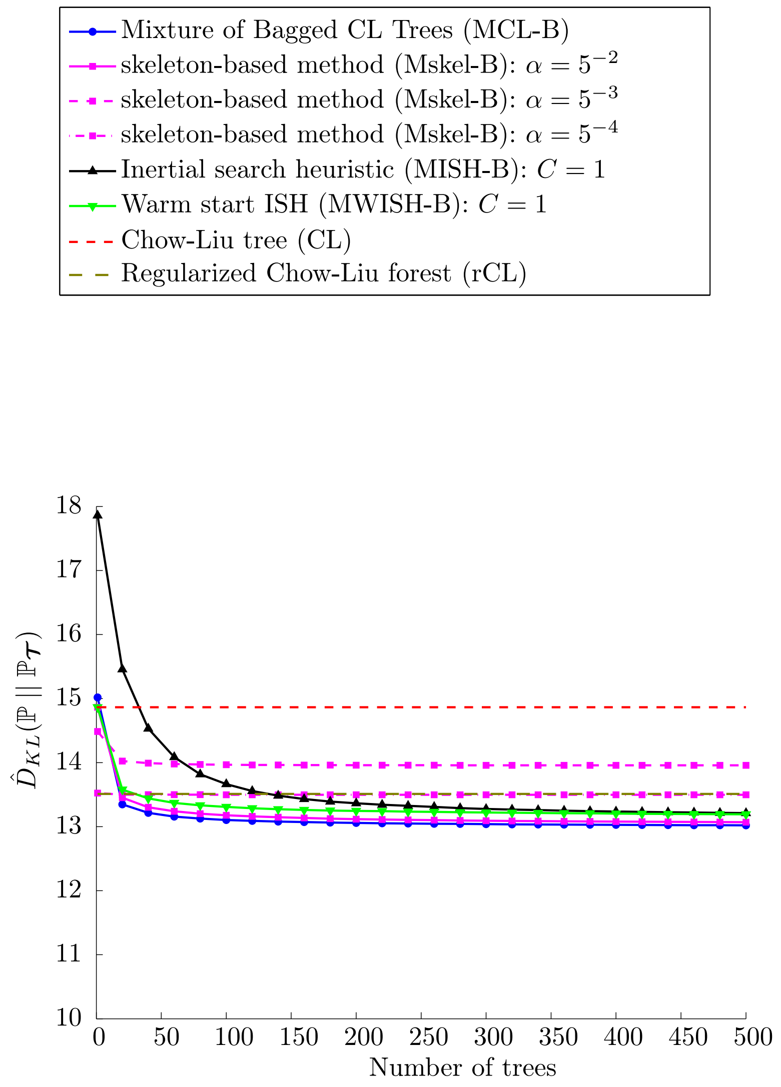
Now that the estimations of the "good" edges are better, reducing ρ seems to have an opposite effect on the improvement rate of Mskl-B. Notice that, while at $m = 100$, the lowest ρ still seems better on average, confidence intervals (one standard deviation) on the values reported (and not displayed) overlap for the different methods. This suggests the mixtures resulting from the different values of the parameters cannot really be distinguished.

MWISH-B is now doing far worse than the MCL-B. I conjecture that the larger sample size leads to less variation in the mutual information computed from bootstrap replicates, leading to slower moves in the space of tree structures for this method.

Also affected by the improved estimation of the mutual information values, the accuracy of the rCL is closer to the accuracy of CL, and is no longer better than the mixtures. In particular, the accuracy of this forest is similar to the accuracy of MCL-B and Mskl-B (for different ρ) when $N = 600$, and worse than the mixtures when $N = 1000$.

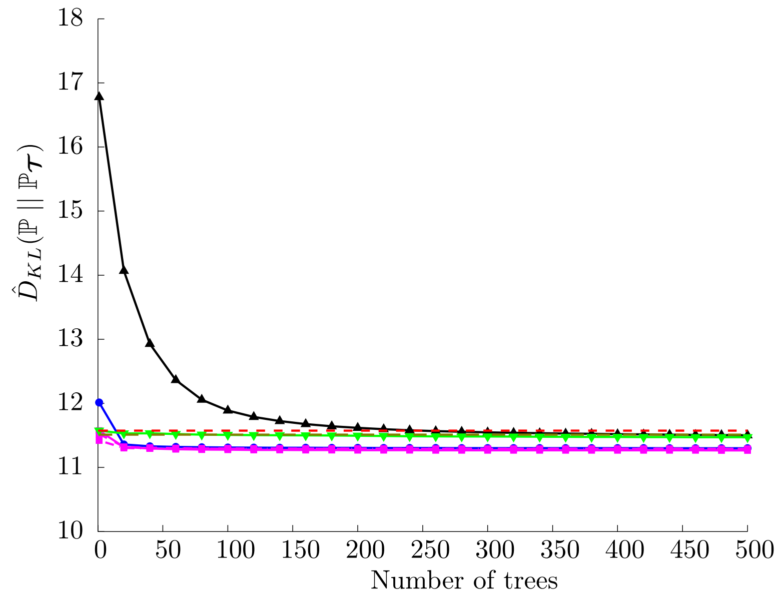
Effect of the Problem Dimensionality p.

Modifying the number of variables has mostly an effect on the inertial search heuristic (MISH-B) methods, since it impacts the relative number of edges considered at each iteration, and thus the exploration speed of the MI matrix. Smaller numbers of variables therefore should accelerate the convergence of this method. Figures 6.7(a,b,c), provide a global picture of the relative performances of the considered methods, with $p = 200$ and over a longer horizon $m = 500$ of averaging.

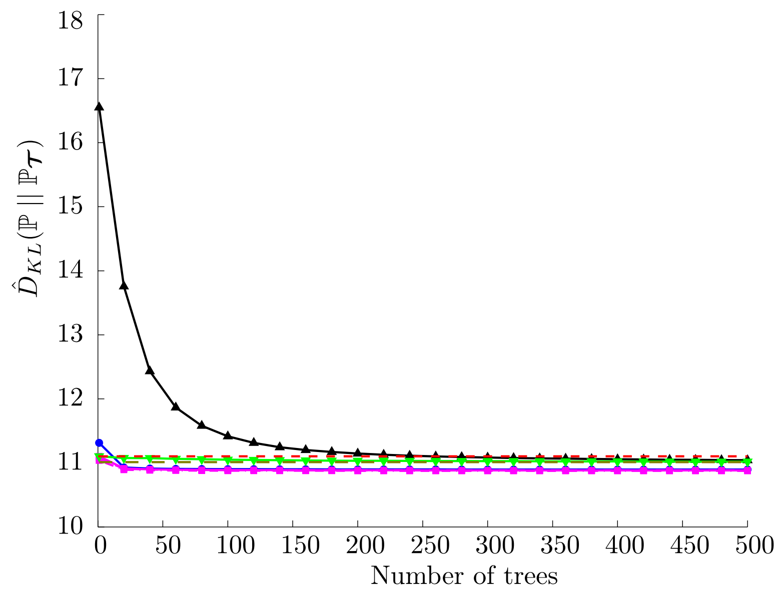


(a) 200 variables, 200 samples.

Figure 6.7: Overview of accuracy performances of the different algorithms listed in Table 6.4, with $p = 200$ or $p = 1000$ variables (left vs right), and for increasing sample sizes N (200, 600 and 1000, from top to bottom). Vertical axis: KL divergence to the target density estimated by Monte Carlo. Horizontal axis: number m of mixture terms used by the different methods (except for the CL and rCL method, using a single model)

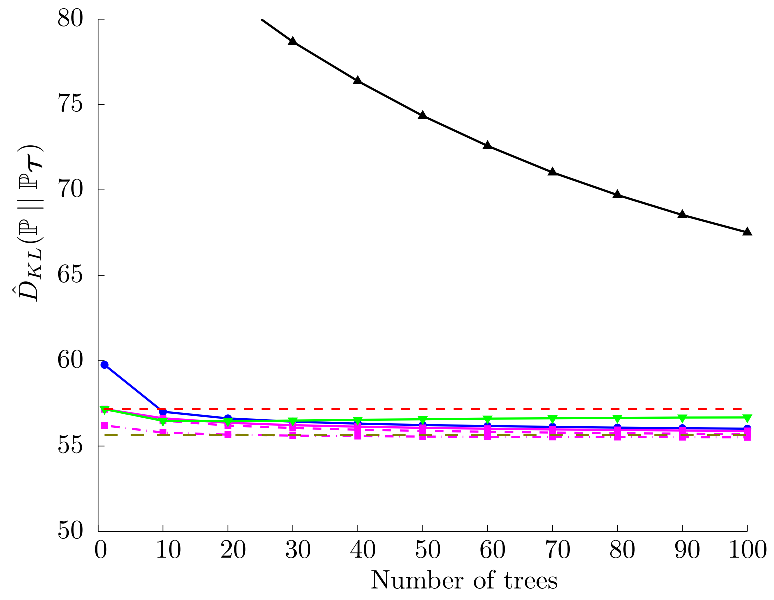


(b) 200 variables, 600 samples.

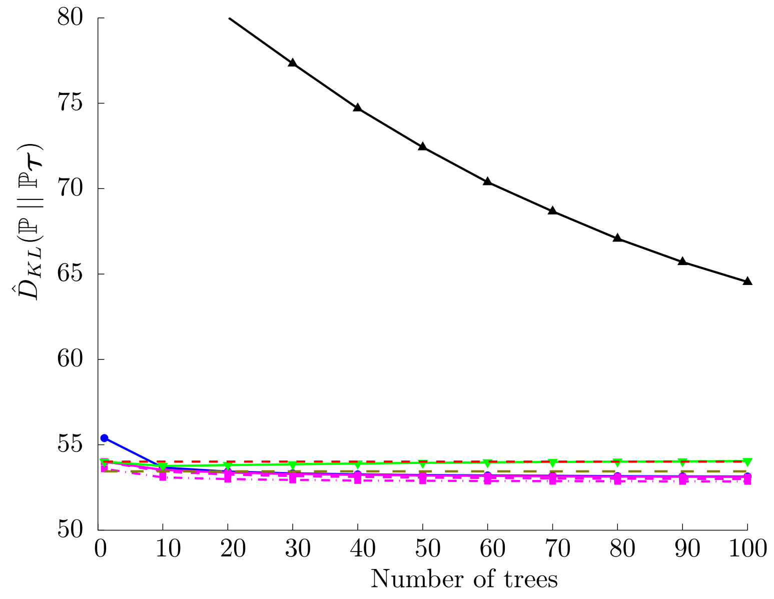


(c) 200 variables, 1000 samples.

Figure 6.7: (continued) Overview of accuracy performances of the different algorithms listed in Table 6.4, with $p = 200$ or $p = 1000$ variables (left vs right), and for increasing sample sizes N (200, 600 and 1000, from top to bottom). Vertical axis: KL divergence to the target density estimated by Monte Carlo. Horizontal axis: number m of mixture terms used by the different methods (except for the CL and rCL method, using a single model)



(d) 1000 variables, 600 samples.



(e) 1000 variables, 1000 samples.

Figure 6.7: (continued) Overview of accuracy performances of the different algorithms listed in Table 6.4, with $p = 200$ or $p = 1000$ variables (left vs right), and for increasing sample sizes N (200, 600 and 1000, from top to bottom). Vertical axis: KL divergence to the target density estimated by Monte Carlo. Horizontal axis: number m of mixture terms used by the different methods (except for the CL and rCL method, using a single model)

In these simulations, the regularized model is always worse than the best mixtures models, namely the mixtures of bagged Markov trees (MCL-B) and the skeleton approximation (Mskl-B). The warm-start inertial search heuristic (MWISH-B) is sometimes better than the regularized model, sometimes at the same accuracy.

Another observation is that both inertial methods converge to the same point. Therefore, and despite a better improvement rate at the beginning, sampling structures far from the optimal one (in MISH-B) does not improve the accuracy of the mixture over sampling them directly from the space of good structures (in MWISH-B). Based on this observation, one might actually be tempted to discard the first structures output by MISH-B, hoping for an improved convergence speed. I believe that considering all edges in the first step of the method (MWISH-B) is more productive, since the method is directly initialized in the neighborhood of good structures, without computing the weight of any edge more than once. Nevertheless, if the computational resources to compute a CL tree are not available, MISH-B remains a strong method.

General Accuracy With Respect to the Two References

The accuracy of the different mixtures of Markov trees was already compared to the accuracy of a single Chow-Liu tree (CL) and of a single “gold standard" for regularization (rCL) in the previous paragraphs. However these observations are important to assess the interest of mixtures of Markov trees to reduce the variance, and they constitute therefore one of the main contributions of the thesis. Therefore, they are sum up in the present section.

In the six problem settings considered in the present round of experiments (p =200 or 1000 variables times and either 200, 600 or 1000 samples), the bagged mixtures of Markov trees (MCL-B) and the skeleton based approximations (Mskl-B, with a sufficiently large ρ) always outperformed a single Chow-Liu tree. The inertial search heuristics, both the original method (MISH-B) and the warm-start variant (MWISH-B), also outperforms the accuracy of CL most of times, i.e. in five out of six problem configurations. Therefore, mixtures of Markov trees can indeed reduce the variance of a single Chow-Liu tree.

Moreover, in five out of six problem settings, Mskl-B and MCL-B also achieves the same accuracy of even outperform the “gold standard” for reg-

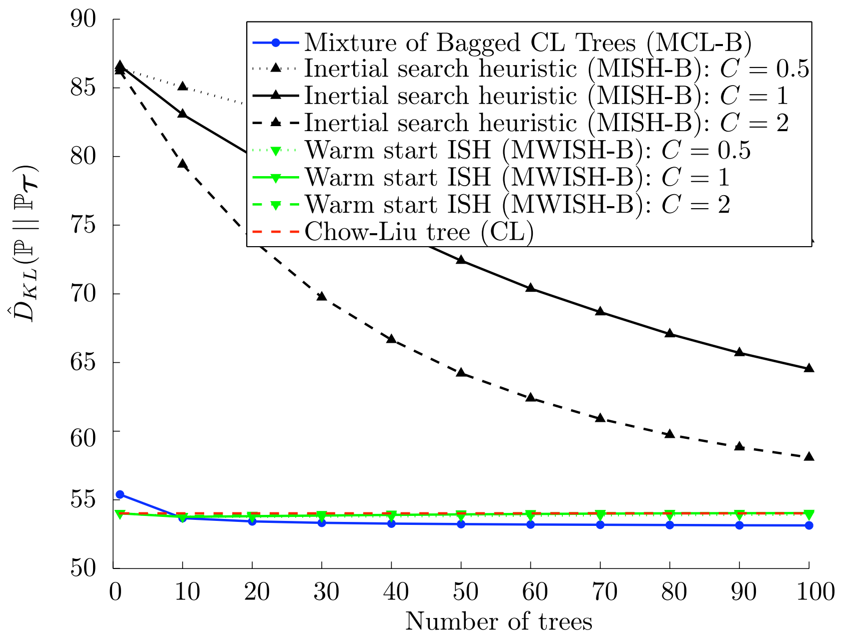


Figure 6.8: Modifying the number of edges considered at each step only affects MISH-B when all edges are not considered in the first iteration (shown here for $p = 1000$, $N = 1000$).

ularization (rCL), a forest where the threshold on the mutual information values is optimized on the test set. This is particularly important : the mixtures studied in this thesis may perform as well or even better in terms of accuracy than a regularization performed by an oracle.

Inertial Search Heuristics and the Effect of C

Modifying the number of edges considered at each iteration in both MISH-B and MWISH-B, as depicted in Figure 6.8, mostly affects the former method. Indeed, doubling ($C = 2$) or dividing by two ($C = 0.5$) the number of edges explored has a huge impact on its convergence, while it hardly affects MWISH-B.

This is a consequence of the modification of the speed of the exploration of the MI matrix. Since it is necessary to discover strong edges to obtain a good mixture, the rate of discovery of these edges strongly affects the

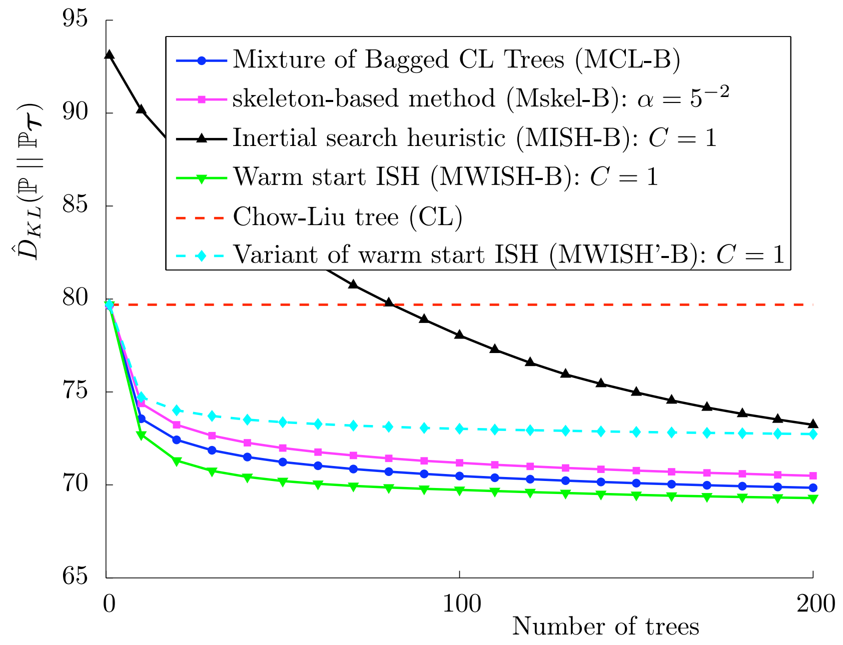


Figure 6.9: In the warm-start ISH method, using $\mathcal{S} = E(\mathcal{T}_1)$, the MWSI'-B variant, rather than $\mathcal{S} = E(\mathcal{T}_{i-1})$ (for tree $\mathcal{T}_i$) leads to a worse accuracy (with $p = 1000$ and $N = 200$). Horizontal axis: ensemble size m; vertical axis KL divergence to the target density estimated by Monte-Carlo and averaged over 5 target densities and 6 training sets.

convergence of this method.

Influence of the Edges Retained in MWISH-B

This section studies the accuracy of a possible variant of MWISH-B. Because of the importance of considering good edges, it may be tempting to always use the edges of the Chow-Liu tree rather than the edges of the previous tree. MWSI'-B is a variant of MWISH-B that does precisely that. In both cases, these edges are completed by random edges to reach the required number K of edges.

Figure 6.9 displays the accuracy of both variants of MWISH-B along with some other methods presented in this section. Both methods start at the same accuracy for the first tree, because both first compute the Chow-Liu tree. However, the original method quickly reaches a better accuracy.

This may be explained in terms of bias and variance. The search space of both methods are identical. The model biases are therefore equivalent. Only the search strategy varies: MWSI'-B will sample structures closer to the model output by the Chow-Liu algorithm than MWISH-B. The estimation bias may therefore differ slightly. If they differ, this bias is likely to be better for MWSI'-B. On the other hand, MWISH-B will sample more diverse structures, which may lead to a better reduction in the variance of the model.

Since MWISH-B has a better accuracy, the difference in the reduction in variance must be greater than the difference in the estimation bias, if it exists.

Effect of the Initial Tree

The influence of the distribution encoded by the first Markov tree of any mixture on its accuracy vanishes over time. Moreover, because information is transmitted from one tree structure to another, the first tree structure also influences the following distributions. In some methods considered so far, the first tree is computed on the original learning set (e.g. MWISH-B), and in some other methods, it is computed on a bootstrap replicate (e.g. MCL-B). This introduces a disparity between the method for a small number of terms m. This section highlights this difference in accuracy and studies it.

To allow a comparison between the two learning sets that can be used to construct the first tree, a variant of each method cited in the above paragraph is developed. If the first tree is built on the original learning set in the original method, it is built on a bootstrap replicate in the variant, and reciprocally. These new variants are respectively denoted by MWSI"-B and MCL"-B. Figure 6.10 displays the performance of these 4 methods in terms of the average KL divergence with the target models for $p = 200$ and $N = 600$.

At first there is a significant difference. The Chow-Liu tree is more accurate than a tree whose structure is learned on a bootstrap replicate. Therefore, the methods where the first tree = Chow-Liu (MCL"-B and MWISH-B) originally have a better accuracy. This difference can be either because of a larger bias (the model is an average further away from the optimal Markov tree when it is built on a bootstrap replicate) or a larger variance (the model

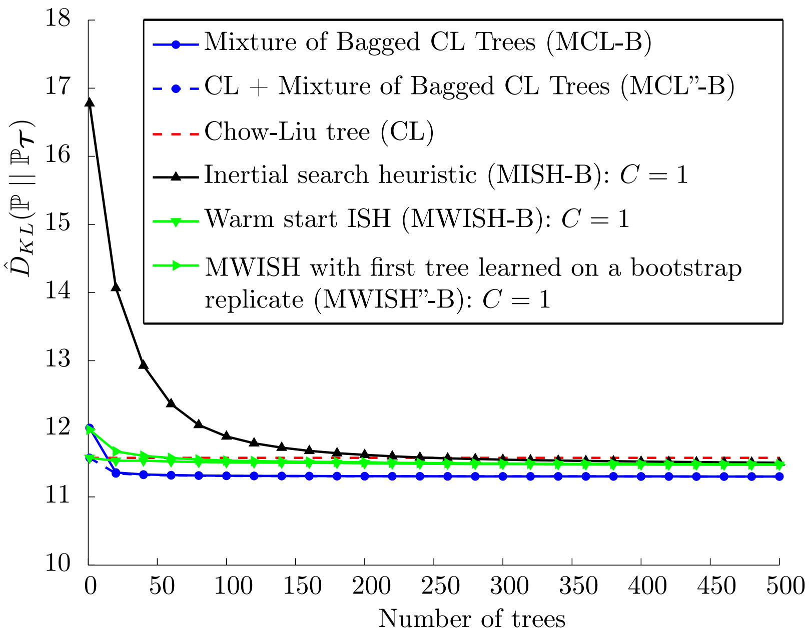


Figure 6.10: Whether the first tree of any method is constructed on the original learning set or on a bootstrap replicate only influences the precision of the mixture for a low number of trees. This is illustrated here for two methods, MWISH-B and MCL-B. The first tree of these methods are respectively built on the original learning set and a bootstrap replicate. MWSI”-B and MCL”-B are therefore introduced: these methods are similar to MWISH-B and MCL-B, but respectively construct the first tree on a bootstrap replicate and on the original learning set.

varies more when built on a replicate), or both. However, because the parameters of the model are not learned on a bootstrap replicate but on the original learning set, the first tree can be better than Chow-Liu when its structure is learned on a bootstrap replicate (see Section 6.1.6), although this is not the case here.

As the number of trees increases, the difference between the accuracies of the pair of mixtures built by two variants of the same methods disappears. The convergence between these two mixtures is faster for MCL-B (10 terms) than for MWISH-B (100 terms). This is because the first tree of MWISH-B influences the following structures, since its edges are reused, while all trees are independent in MCL-B. The coupling between the models in the sequential mixtures makes the influence of the first tree more important than in mixtures where the trees are independent, so the convergence takes longer. However in both cases, I am considering a sufficiently large number m of Markov trees to ensure the first tree does not impact the asymptotic (in m) comparison of the methods.

Computing Times

Our experiments were performed on a grid running ClusterVisionOS and composed of pairs of Intel L5420 2.50Ghz processors with either 16 or 32 GB of RAM. Due to the environment, run time for a method can vary a lot, and I therefore decided to report relative minimum running time for every method. Those results are displayed in Tables 6.5 and 6.6 for respectively 200 and 1000 variables / 500 and 100 trees. Results for Mskl-B are reported for $\rho = 0.005$.

Table 6.5: Serial minimum computing times, given for $p = 200$ variables and $m = 500$ trees - except for the Chow-Liu algorithm

Method	Complexity	running time $N =$		
		200	600	1000
CL	$p^2 \log(p)$	1	3.07	5.3
MCL-B	$mp^2 \log(p)$	532	1531	2674
MISH-B	$mp \log(p)$	45	186	432
Mskl-B	$p^2 + mK(\rho) \log(K(\rho))$	21	82	191
MWISH-B	$p^2 \log(p) + mp \log(p)$	45	192	406

Those numbers show that the proposed methods (lower part of the table) are roughly an order of magnitude faster than the standard MCL-B method, and this relative speed-up is stronger in the higher dimensional case. Also, as we saw from the accuracy results, these methods converge as quickly as MCL-B.

If one is considering parallelizing those methods at a high level, namely by computing trees individually on different cores, MCL-B and Mskl-B are the best candidates, since the trees in these methods are independent in the

Table 6.6: Serial minimum computing times, given for $n = 1000$ variables and $m = 100$ trees, except for the Chow-Liu algorithm

Method	Complexity	running time $N =$ 200	600	1000
CL	$p^2 \log(p)$	37	98	174
MCL-B	$mp^2 \log(p)$	5037	11662	19431
MISH-B	$mp \log(p)$	181	800	1433
Mskl-B	$p^2 + mK(\rho) \log(K(\rho))$	139	612	1005
MWISH-B	$p^2 \log(p) + mp \log(p)$	218	766	1359

former method and independent conditionally on the first tree in the latter. In the two ISH methods, each tree depends on the previous one in a Markov chain-like dependency, and parallelizing is hence more difficult. But, at a lower level, all algorithms could take advantage of the parallelization of the computation of a MWST.

Overall, the Mskl-B method appears as the most appealing method; it always combines fast convergence (as fast as MCL-B) when the number of terms of the mixture is increased and, from the computational point of view, it is also the most efficient one among those that we investigated, about 20-30 times faster than MCL-B in realistic conditions; furthermore it is easy to parallelize. Neverteless, the inertial heuristic with warm start (MWISH-B) is competitive as well.

Note that convergence speed may vary between methods, and some might require fewer iterations before performance (almost) stabilizes.

6.2.6 Evaluation on Realistic Problems

The algorithms presented here were also applied on a set of more realistic problems described in Appendix A.3. These experiments evaluate the interest of these mixtures on problems closer to reality, and in various conditions. The quality of the model is here evaluated by the negative loglikelihood of an independent test set of 5000 observations. Only results for a selection of the most accurate methods (MWISH-B, MCL-B and Mskl-B) are reported here.

An overview of the results obtained on these networks is presented in Table 6.7, where the best result of each configuration is displayed in bold. The values obtained by the perfectly regularized forest (rCL) developed in Section 5.1.2 are copied in this table to allow an easy comparison. Mskl-B is only tested with the non-optimized value $\rho = 0.05$. In addition, Figure 6.11 displays the evolution of the accuracy of the mixtures with respect to the number of terms m for a few selected target distributions.

Table 6.7: Negative log-likelihood achieved by the different learning algorithms on the realistic problems described in appendix . These values are averaged on 5 learning sets of observations.

Distribution	p	N	CL	rCL	MCL-B	MWISH-B	Mskl-B
Alarm10	370	200	166.65	166.65	**163.59**	206.34	166.80
Alarm10	370	500	136.37	136.28	**135.31**	181.82	135.61
Child10	200	200	135.29	135.08	**133.94**	159.74	134
Child10	200	500	131.71	131.71	**131.01**	156.11	131.02
Gene	801	200	485.21	483.6	482.80	585.79	**482.66**
Gene	801	500	477.48	476.75	**473.75**	572.13	473.79
Hailfinder10	560	200	550.85	**547.64**	551.75	651.75	549.89
Hailfinder10	560	500	523.81	**523.26**	523.61	628.45	523.31
Insurance10	270	200	210.1	210.1	**206.77**	243.57	215.23
Insurance10	270	500	198.87	198.87	**195.47**	234.98	202.01
LungCancer	800	200	435.72	**435.46**	437.41	497.96	436.01
LungCancer	800	500	424.69	424.44	418.31	493.42	**418.30**
Munin	189	200	42.614	36.987	41.799	41.749	**35.566**
Munin	189	500	37.66	35.414	37.656	38.131	**35.140**
Pigs	441	200	390.75	390.75	**387.19**	428.55	387.24
Pigs	441	500	385.59	385.59	**382.22**	423.71	382.26

Once again, rCL rarely yields the best accuracy, showing the advantages of mixtures of Markov trees.

Focusing on mixtures only, the accuracy of MISH-B is weaker and regularly clearly worse than the accuracy of the other two mixtures. Mskl-B on the contrary is often close to MCL-B in terms of accuracy (9 out of 16 configurations, illustrated by Figures 6.11a and 6.11b). Differences nevertheless appear in a few configurations: Mskl-B is sometimes worse (3 out of 16 configurations: Alarm10 for $N = 200$ and Insurance10, illustrated by Figure 6.11c) and sometimes clearly better (4 out of 16 configurations: Munin and Hailfinder10, illustrated by Figure 6.11d).

When Mskl-B is worse than MCL-B, regularization is not necessary (values for rCL and CL are equal). The difference on the first term of the mixture may be explained by the fact that Mskl-B removes some edges that are necessary, although of poor weight. On the other hand, rCL knowns these edges must be included in the structure thanks to its knowledge of the test set. The first term of Mskl-B thus suffers from a larger bias, and the reduction in variance brought by the regularization cannot compensate it. As the number of trees is increased, the variance is reduced, but it cannot compensate for the bias. However, the fact that regularization is not needed does not necessarily indicates that Mskl-B will have poor accuracy. Indeed, Mskl-B improves upon rCL e.g. on the distributions Pigs or Child10 for $N = 500$.

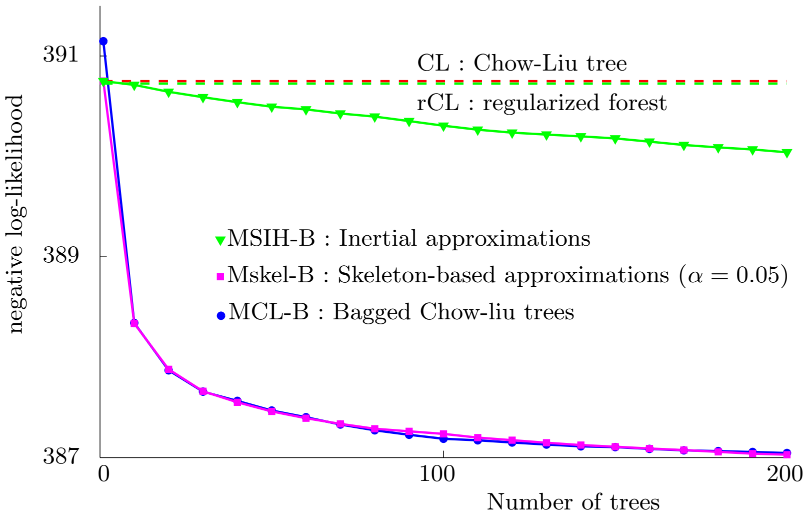
Similarly, the superior accuracy for Mskl-B over MCL-B seems to be concomitant with a large difference between CL and rCL, i.e. when regularizing the Chow-Liu tree improves it by a large margin. In this case, the better accuracy of Mskl-B is likely mostly related to the reduction of the variance of each term due to the regularization rather than to the reduction in variance due to model averaging. Nevertheless, these two reduction can be combined, and model averaging can further improve the accuracy. Indeed, Mskl-B is the best method in 4 out of 16 configurations.

Mskl-B has therefore two advantages with respect to MCL-B. The first is the speed-up in the construction of the mixture, the second the regularization itself, that allows to increase the accuracy of the model. Note that this regularization is sometimes counterproductive as mentioned above.

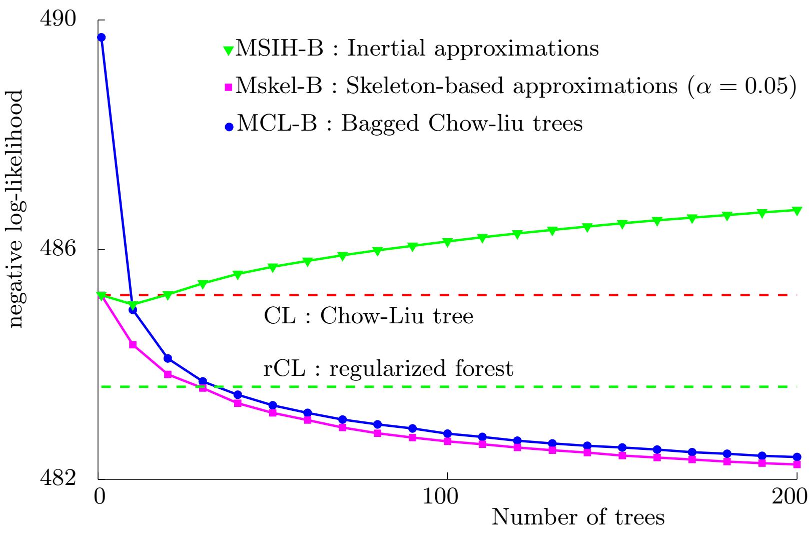
Therefore it should be interesting to select ρ based on each different problem, despite the use of an hypothesis test. Cross-validation could be used for this purpose.

6.3 Conclusion

In this chapter, several new algorithms for learning mixtures of Markov trees in a variance reduction setting have been presented. All these algorithms

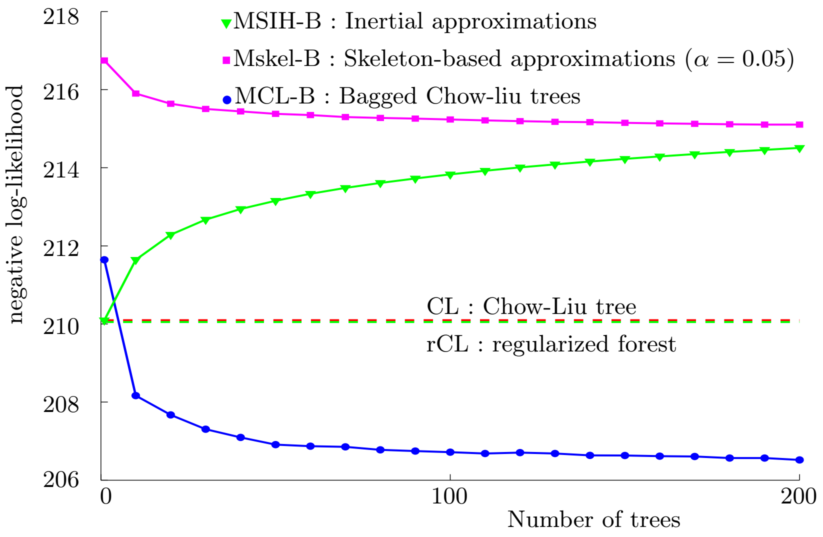


(a) Pigs, $N = 200$ samples.

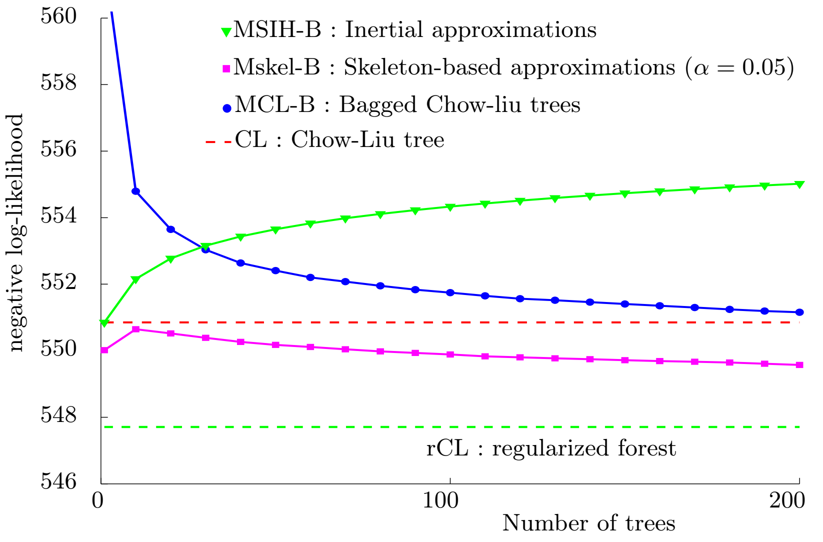


(b) Gene, $N = 200$ samples.

Figure 6.11: Illustration of the behavior of the mixtures on realistic distributions for increasing m.



(c) Insurance10, $N = 200$ samples.



(d) Hailfinder10, $N = 200$ samples.

Figure 6.11: (continued) Illustration of the behavior of the mixtures on realistic distributions for increasing m.

are instantiations of the common meta-algorithm for learning mixtures of Markov trees based on the perturb and combine framework, but they can be divided into two categories, based on the algorithm used to construct the tree structures underlying each term of the mixture. In the first category, each tree is independent from any other structure, while in the second each tree depends on previous structures generated.

In the first category, new algorithms were developed based on the new randomization of the Chow-Liu algorithms presented in the previous chapter, and by modifying the bootstrapped Chow-Liu algorithm so as to estimate the parameters of each Markov tree on the original learning set rather than on the bootstrap replicate. This latter variant produced the most accurate mixture, while others have a lower run-time. In general, the better the edges considered to construct each tree, the better the structure. A trade-off between accuracy and complexity is therefore possible by varying this number of edges.

In the second category, approximations of the bagging algorithm were developed with the goal of decreasing complexity without sacrificing accuracy. The most interesting method consists in filtering the edges based on an independence test during the computation of the first model of the mixture. Only pairs of variables for which independence was rejected are considered in subsequent run of the Chow-Liu algorithm on bootstrap replicates. In our experimental study, this method leads to a speed-up of one order of magnitude while closely matching the performance of the original bootstrap algorithm.

Both existing and new methods were evaluated on synthetic and realistic learning sets for varying p and N. One of the main observation is that mixtures are increasingly better than a single Chow-Liu tree when the number of samples shrink and/or when the number of terms in the mixture increases. With respect to regularization however, the mixtures could not always achieve the same accuracy as the forest regularized by an oracle, although in most situations mixtures were better. It remains to be seen how close to this optimal model a forest can be regularized without an oracle. Moreover, it would be interesting to tune the parameter ρ of the mixture method to the learning set, and to contrast this with regularization. A possible method for both tasks would be cross-validation.

Many other mixtures could be developed. For example, it could be interesting to progressively increase the number of candidates edges as the number of terms in the mixture grow. The first terms of the mixture would be centered around good structures and have a lower bias, whereas terms coming later in the mixture would be allowed more freedom, and could therefore reduce the variance, at the cost of a higher bias and a larger computing time.

The mixtures considered so far has proved effective at reducing the variance of a single Chow-Liu tree. The next chapter investigates whether these models can be used to improve upon a bias-reducing mixture of Markov trees, where each term of the mixture can be viewed as a mode of the distribution.

Chapter 7

2-level Mixtures of Markov Trees

This chapter combines the two frameworks for building mixtures of Markov trees. As explained in Chapter 4, learning a mixture of Markov trees from a set of observations can be performed to reduce either the bias or the variance of a single Chow-Liu tree. This thesis has so far focused on the latter, starting on trees in Chapter 5 and upgrading to mixtures in Chapter 6. In this chapter, a new two-level algorithm is proposed, where the upper level seeks to reduce the bias while the lower level seeks to reduce the variance. This algorithm is evaluated empirically on learning sets generated from a mixture of Markov trees and from the synthetic and realistic probability densities already used.

The two frameworks to build mixtures of Markov trees were already combined in the literature. [KK06] define a Markov-Chain Monte Carlo method over the space of k-clusterings of the observations and construct one term of the mixture on each cluster, by exact Bayesian averaging over the space of Markov trees [MJ06]. [KS07] use a similar approach to [KK06] but allow k to vary and sample one tree from the Bayesian posterior. In both cases, the MCMC procedure is used to sample a set of mixtures, and these mixtures are then averaged. While both papers report an improvement over the original mixture of [MJ01], these methods are of cubic complexity in the number of variables, thus deteriorating the scalability of Markov trees.

This chapter studies a more scalable (quadratic) alternative to combine these two frameworks, by replacing Markov trees in the first framework (maximum likelihood estimation of the mixture based on the EM algorithm) by a mixture of trees generated by the second framework (model averaging). Rather than learning a single Markov tree by the Chow-Liu algorithm for each term of the upper mixture or using a Bayesian averaging, randomized procedures developed in the previous chapter are used to replace each term by a lower level mixture. The combination order is thus the opposite of [KK06, KS07], a choice essentially motivated by computational considerations.

This chapter starts by a brief recall of the two frameworks for building mixtures of Markov trees in Section 7.1. The proposed approach is described in Section 7.2, its interest and variance reduction capacity are experimentally studied in Section 7.3. In addition, these results are used to compare the two frameworks, showing their complementarity.

7.1 Two Types of Mixtures of Markov Trees

Working in the class of Markov trees is algorithmically efficient, but can be either too restrictive, i.e. have a large bias, when samples are plentiful, or lead to overfitting, i.e. have a large variance, when only few samples are available, see Section 2.4 for further discussion.

Those two frameworks are briefly recalled below, each with an exemplative algorithm. Both algorithms are also used as building blocks in the combination proposed in Section 7.2.

7.1.1 Mixture for Reducing the Bias

Constraining a Bayesian network to be a Markov tree when the target distribution is more complex than such a tree and for a sufficiently large number of samples will cause some relationships between variables to be missed and the distribution won't be perfectly estimated, an error called the bias. Using a mixture rather than a single Markov tree results in an improved modeling power [MJ01] and a higher achievable likelihood.

This optimization can be done e.g. by the EM algorithm [DLR77, MK08] as detailed in Section 2.6.1, or by an adaptation of fractional covering [KK09]. The weights are considered as the marginal probabilities $\mu_k = \mathbb{P}(Z = k)$ of Z, a hidden variable selecting one distribution $\mathbb{P}_{\mathcal{T}_k}(\boldsymbol{\mathcal{X}}) = \mathbb{P}(\boldsymbol{\mathcal{X}}|Z = k)$:

$$\mathbb{P}_{\boldsymbol{\mathcal{T}}}(\boldsymbol{\mathcal{X}}) = \sum_{k=1}^{m} \mathbb{P}(Z = k)\mathbb{P}(\boldsymbol{\mathcal{X}}|Z = k) \ . \tag{7.1}$$

Algorithm 7.1 MT-EM: ML mixture of Markov trees

```
Input: data $D$, mixture size $m$
Initialize ($\boldsymbol{\mathcal{T}}$,$\boldsymbol{\theta}$,$\boldsymbol{\mu}$)
repeat
  for $k = 1 \to m$ and $i = 1 \to N$ do
    $\gamma_k(i) = \dfrac{\mu_k \mathbb{P}_{\boldsymbol{\mathcal{T}}[k]}(\mathbf{x}_{D_i})}{\sum_{k=1}^{m} \mu_k \mathbb{P}_{\boldsymbol{\mathcal{T}}[k]}(\mathbf{x}_{D_i})}$
  end for
  for $k = 1 \to m$ do
    $D'_k = (\mathbf{x}_{D_i}, \gamma_k(i))_{i=1}^{N}$
    $\boldsymbol{\mathcal{T}}[k]$ = Chow-Liu($D'_k$)
    $\boldsymbol{\theta}[k]$=LearnParameters($\boldsymbol{\mathcal{T}}[k]$,$D'_k$)
    $\mu_k = \dfrac{\sum_{i=1}^{N} \gamma_k(i)}{N}$
  end for
until convergence
return ($\boldsymbol{\mathcal{T}}$,$\boldsymbol{\theta}$,$\boldsymbol{\mu}$)
```

This estimate of $\mathbb{P}(\boldsymbol{\mathcal{X}}, Z)$ is optimized by alternating between estimating a distribution of the hidden variable Z for each observation $D[i]$ and optimizing both $\mathbb{P}(Z = k)$ and $\mathbb{P}(\boldsymbol{\mathcal{X}}|Z = k)$. This process is described in Algorithm 7.1, where $\gamma_k(i)$ can be seen as the probability that $Z = k$ for observation $D[i]$ and according to the current estimate of $\mathbb{P}(\boldsymbol{\mathcal{X}}, Z)$. The vectors $(\gamma_1(i), \ldots, \gamma_m(i))_{i=1}^{N}$ define a soft partition of D into m weighted learning samples D'_k, where each D'_k is constructed by associating a weight $\gamma_k(i)$ to each $D[i]$. By comparison, all observations in D have a weight of one. In the second step of the iteration, the Chow-Liu algorithm is applied to each D'_k to obtain T_k and μ_k is estimated based on the $\gamma_k(i)$.

7.1.2 Mixture for Reducing the Variance

Chapter 6 has shown how a possible lack of data can be compensated by applying the “perturb and combine" framework to Markov trees. In this approach, m different plausible models are generated and their predictions averaged. A generic meta-algorithm (Algorithm 4.1) was presented and specialized into many different methods for learning such a mixture of Markov

Algorithm 7.2 MT-Bag: bagging of Markov trees ($\equiv$ MCL-B)

```
  **Input:** data $D$, mixture size $m$
  $\boldsymbol{\mathcal{T}} = \emptyset$; $\boldsymbol{\theta} = \emptyset$
  $\lambda = \{1/m, \cdots, 1/m\}$
  **for** $k = 1 \to m$ **do**
    $D'$= bootstrap($D$)
    $\boldsymbol{\mathcal{T}}[k]$ = Chow-Liu($D'_k$)
    $\boldsymbol{\theta}[k]$=LearnParameters($\boldsymbol{\mathcal{T}}[k]$,$D$)
  **end for**
  **return** ($\boldsymbol{\mathcal{T}}$,$\boldsymbol{\theta}$,$\boldsymbol{\lambda}$)
```

tree. Bootstrap aggregation of the structure-learning part of the Chow-Liu algorithm was one of the most accurate methods. It will therefore be used in the construction of the 2-level mixture. The corresponding algorithm for the generation of a set of Markov trees is recalled in Algorithm 7.2.

The resulting set $\boldsymbol{\mathcal{T}}$ is associated to parameters learned on the full learning set and uniform weights to constitute a mixture.

7.2 Two-level Mixtures of Markov Trees

In this section a new algorithm is constructed by combining the methods presented in the previous section. Combining a bias and a variance reducing framework has also been considered in supervised learning, see e.g. [Bre99].

The EM algorithm builds a soft partition of size m of the learning set and one maximum-likelihood tree on each downweighted data set D'_k. Therefore each tree is built using a smaller effective number of samples than in the original data set. On the other hand, algorithms of the second category build mixtures that are increasingly better than a single tree when the number of samples shrinks.

I therefore propose to replace the Chow-Liu algorithm in Algorithm 7.1 by an algorithm constructing a mixtures of Markov trees reducing the variance. Therefore, each term of the ML mixture is a mixture of Markov trees rather than a single Chow-Liu tree. This allows each algorithm to operate in the configuration it excels in. The bias reduction of the ML algorithm operates on the whole set of observations at the upper level, and partitions in into subsets of observations. Each variance reduction algorithm works on one of those smaller subsets, at the lower level. The resulting model is thus a two-level mixture of Markov trees of the following form:

$$\mathbb{P}_{\boldsymbol{\mathcal{T}}}(\boldsymbol{\mathcal{X}}) = \sum_{k=1}^{m_1} \mu_k \mathbb{P}_{\boldsymbol{\mathcal{T}}_k}(\boldsymbol{\mathcal{X}}) \tag{7.2}$$

$$\mathbb{P}_{\boldsymbol{\mathcal{T}}_k}(\boldsymbol{\mathcal{X}}) = \sum_{j=1}^{m_2} \lambda_{k,j} \mathbb{P}_{\mathcal{T}_{k,j}}(\boldsymbol{\mathcal{X}}) \;\; \forall k = 1, \dots, m_1 \;\; , \tag{7.3}$$

where m_1 and m_2 are respectively the number of terms in the upper level EM optimization of the mixture and in the lower level mixtures $\boldsymbol{\mathcal{T}}_k$ learned on each D'_k; $\mathcal{T}_{k,j}$ is the jth Markov tree in $\boldsymbol{\mathcal{T}}_k$ and $\lambda_{k,j}$ denotes its weight.

Using algorithms for reducing the variance rather than the Chow-Liu algorithm during the iterative process of the EM algorithm substantially slows down each step and seems to alter the convergence, impeding or stopping it early (based on observations performed on early simulations). Hence, I perform the proposed adaptation of the lower level models only after a first run until convergence of the basic EM algorithm using single trees. The resulting method is specified by Algorithm 7.3.

Note that bagging is now performed on a weighted data set. The sampling procedure to construct the replicates in MT-Bag must be adapted. Observations are thus no longer drawn uniformly, but based on a multinomial distribution where the probability to sample the i^{th} observation equals $\gamma_k(i)/\sum_{i=1}^{N}\gamma_k(i)$. $\sum_{i=1}^{N}\gamma_k(i)$ (truncated to the closest integer) observations are drawn (and associated to a weight of 1) to form each replicate.

7.3 Experiments

A first set of experiments is performed by considering two different strategies for building the second level mixture. In addition to bagging of Markov trees (Algorithm 7.2, MT-EM-Bag), I also consider augmenting each original term (a Chow-Liu tree) by $m_2 - 1$ trees learned on bootstrap replicates, i.e. a mix of the first two methods (MT-EM-BagCl). In a second set of experiments, reported in Section 7.3.3, additional second-level mixtures are considered. The additional mixtures tested are the best performing mixtures of Chapter

Algorithm 7.3 MT-EM-Bag: two-level mixture

```
Input: data $D$, first & second mixture sizes $m_1$, $m_2$
$\{\boldsymbol{\mathcal{T}}, \boldsymbol{\theta}, \mu\} = \textbf{MT-EM}(D, m_1)$
for $i = 1$, $k = 1 \to N$, $m_1$ do {Recover observation weights of MT-EM}
    $\gamma_k(i) = \frac{\mu_k \mathbb{P}_{\mathcal{T}_k}(\mathbf{x}_{D_i})}{\sum_{k=1}^{m} \mu_k \mathbb{P}_{\mathcal{T}_k}(\mathbf{x}_{D_i})}$
end for
for $k = 1 \to m_1$ do
    $D'_k = (\mathbf{x}_{D_i}, \gamma_k(i))_{i=1}^{N}$
    $\{\boldsymbol{\mathcal{T}}_k, \boldsymbol{\theta}_k, \lambda_k\} = \textbf{MT-Bag}(D'_k, m_2)$
end for
return $((\boldsymbol{\mathcal{T}}_k, \boldsymbol{\theta}_k, \boldsymbol{\lambda}_k), \boldsymbol{\mu})$
```

6: the warm-start inertial search heuristic (MWISH-B) and the skeleton-based approximation (Mskl-B).

I evaluate the interest of these combinations against the baseline Algorithm 7.1 (MT-EM) described in Section 7.1. Since the EM algorithm may converge to local maxima and in order to remove the influence of its random initialization, every comparison between the methods is performed based on mixtures built on similar convergence points of the EM algorithm. In other words, for every run of the EM algorithm, I use its final soft partition of the learning set ($\gamma_k(i)$) to build two different two-level mixtures, one based on each algorithm considered, and I also use the weights μ_k returned by this algorithm to formulate the resulting model.

The evaluation is performed using three different target distribution settings:

1. 1 mixture model of three randomly generated Markov trees defined on 100 variables (appendix A.2);
2. 5 synthetic DAG-200-5 (appendix A.1);
3. 9 realistic Bayesian networks (appendix A.3).

The accuracy of any model $\mathbb{P}_{\mathcal{T}}$ constructed in the first two settings is quantified using a Monte Carlo estimate of the Kullback-Leibler divergence, with respect to the target distribution $\mathbb{P}$:

$$\hat{D}_{KL}(\mathbb{P} \mid\mid \mathbb{P}_{\mathcal{T}}) = \frac{1}{N'} \sum_{x \sim \mathbb{P}}^{N'} \log_2 \left(\frac{\mathbb{P}(\mathbf{x})}{\mathbb{P}_{\mathcal{T}}(\mathbf{x})} \right) \quad , \tag{7.4}$$

where N' the number of observations x for the estimate equals 10000 in the first case and 50000 in the second. The same samples are used for every evaluation related to a given target distribution. For the third setting, the accuracy is measured by the negative log-likelihood of an independent test set of 5000 samples.

In the first setting, the similarity of any learned mixture to the target mixture structure was also assessed in terms of the mean number of common edges. Because the ordering of the terms of the original mixture cannot be recovered during learning, this score is the weight of the maximum weighted bipartite matching between the terms in the estimated mixture and those of the target model, solved by the Hungarian algorithm [Kuh55]. The weight of a matching between a tree of the target model and a tree of a mixture learned by MT-EM is their number of common edges. The weight of a matching between a tree of the target model and a lower level mixture of a 2-level mixture is defined as the average number of common edges between the tree and the elements of that mixture. In both cases, an edge comparison does not take the orientation into account.

Unless stated otherwise, m_2 is always set to 10 for the lower level mixtures. Increasing this number should further reduce the variance without effect on the bias, and so is liable to further increase the accuracy of the two-level mixtures.

The EM algorithm is initialized by a uniform mixture of randomly generated Markov trees.

7.3.1 The Target is a Mixture of 3 Trees

This first set of experiments is performed on a target distribution that belongs to the class of models considered by the EM learning algorithm, here the set of mixtures of 3 Markov tree structures.

Learning sample sizes from 100 to 20000 were considered, and several initializations (905 to 50 runs, see Figure 7.3) were performed for each sample size.

The number of upper-level terms m_1 is always fixed to 3 (the size of the target mixture); the number m_2 of lower level terms is fixed to 10 to limit execution time.

Figure 7.1 displays the decrease of the KL divergence to zero as N increases, as expected since the target distribution is a mixture of 3 Markov trees. We observe that the two-level mixtures are better than the baseline, showing the interest of their variance reduction. Actually, MT-EM-Bag and MT-EM-BagCl achieve with N samples roughly the same accuracy as the baseline MT-EM method with $2N$ samples.

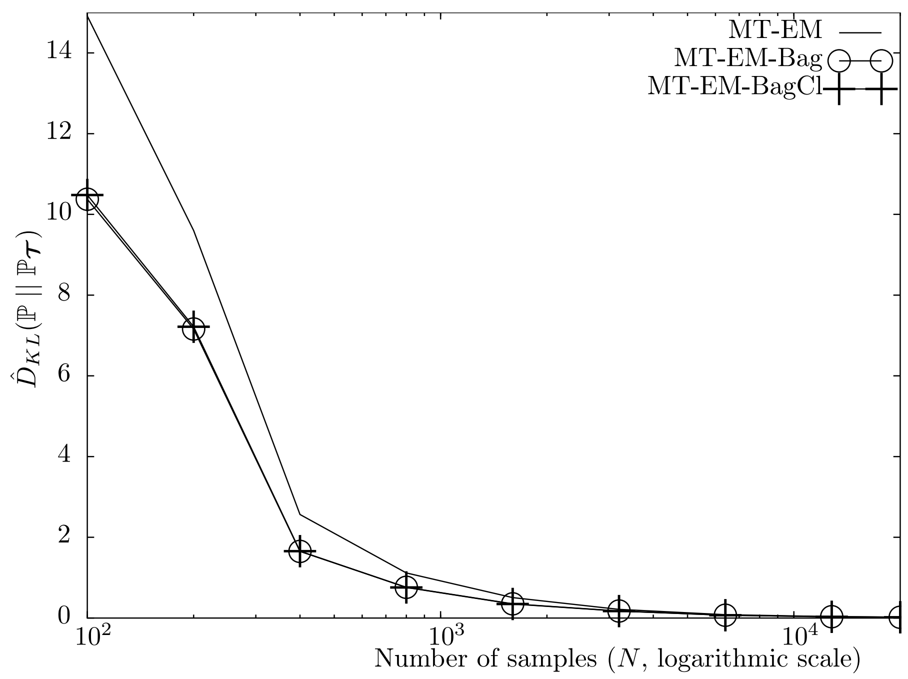


Figure 7.1: Setting 1: The Average KL divergence with respect to the target mixture of 3 trees converges to zero for all methods when the number of samples increases.

The advantage of the two-level mixtures over MT-EM is significant, as can be seen from the difference between the KL divergence of each method and the KL divergence of the corresponding EM method. This is illustrated in Figure 7.2, where each box-plot shows the distribution of KL values obtained over 400 runs of MT-EM on one learning set of size $N = 400$. Similar results are obtained for other sample sizes.

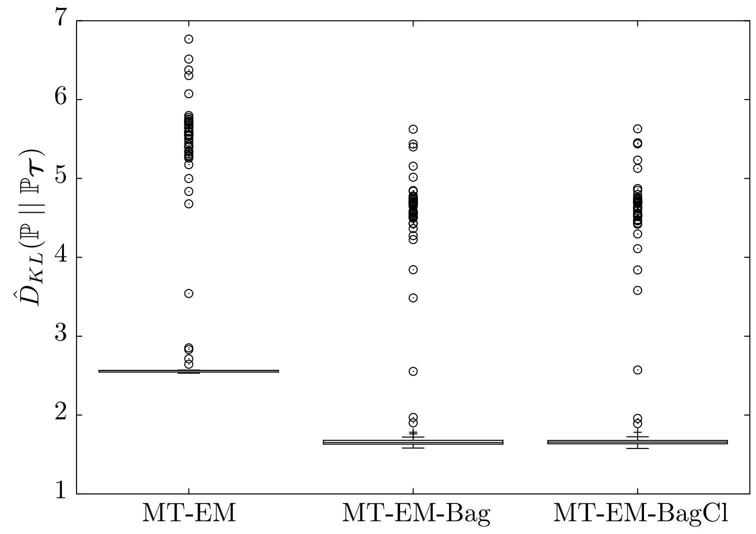


Figure 7.2: Setting 1: For each method, a box-plot showing the distribution of the KL divergences of the models it produces over 400 runs on a dataset of $N = 400$ samples.

In order to determine which one of the two-level methods is better, Figure 7.3 displays for each learning sample size the relative number of runs (and associated data soft partition) where each method achieves a lower KL divergence than the two others. No definite winner among the two proposed methods can be selected based on those results, but we observe that MT-EM is never better than both (actually, any) of them. However, the interest of using the original learning set to build the first tree of the mixture (MT-EM-BagCl) rather than a bootstrap replicate (MT-EM-Bag) increases with the number of samples.

The numbers of recovered edges displayed in Figure 7.4 reveal an expected increase with the number of samples, but also that the EM algorithm is here the best method. The two-level mixtures are thus not as effective to infer the structure of a mixture of Markov trees. However, this also indi-

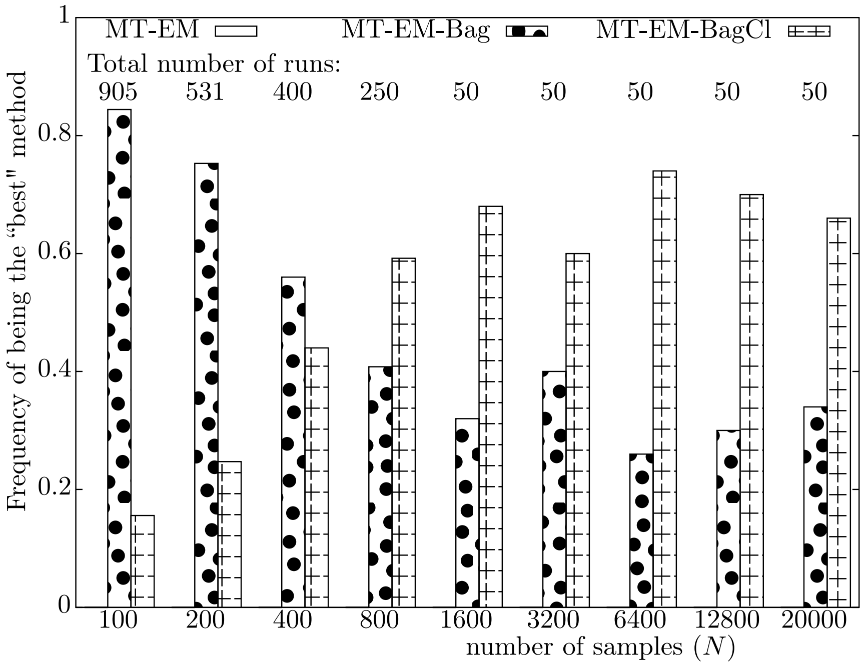


Figure 7.3: Setting 1: Relative number of runs where each method is the best, displayed by number of samples. MT-EM is never the best method.

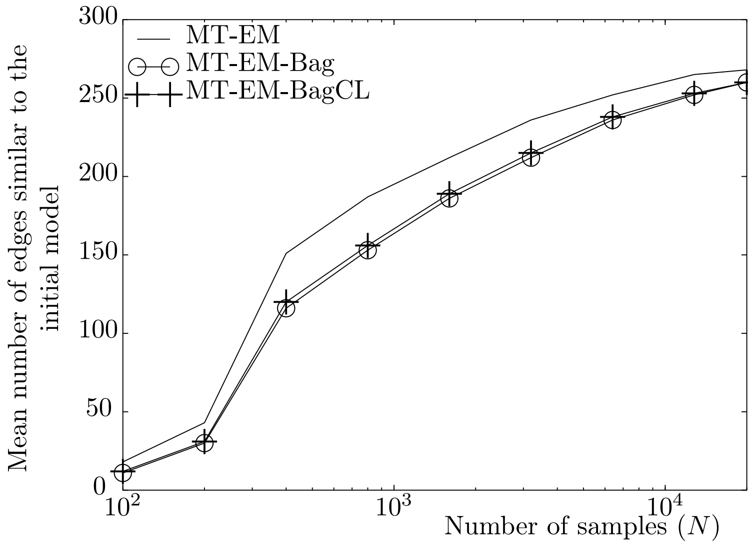


Figure 7.4: Setting 1: The average structure similarity increases with the learning sample size. Notice that MT-EM is better for structure recovery.

cates that the better accuracy of the proposed models is indeed due to the diversity introduced in the evaluation of each term by the mixtures.

7.3.2 The Target is a Bayesian Network

The previous section showed that the proposed algorithms perform well when the target model belongs to the class of mixtures of Markov trees. The present section shows that they are also interesting when the target distribution is encoded by a Bayesian network. Results are reported first for distributions encoded by synthetic Bayesian networks (appendix A.1) to study the influence of m_1, and then on benchmark problems from the literature (appendix A.3)

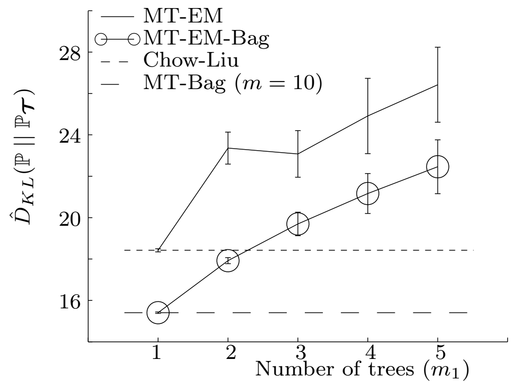
Synthetic BN Target Distributions

The number of terms m_1 in the EM method is a parameter of the method whose optimal value is problem dependent and could e.g. determined by cross-validation. In particular, this optimal value tends to increase with N, with the Chow-Liu algorithm ($m_1 = 1$) being better than EM ($m_1 > 1$) for very small N. In this section I investigate how varying N and m_1 impact the difference between the three methods and the underlying bias/variance trade-off.

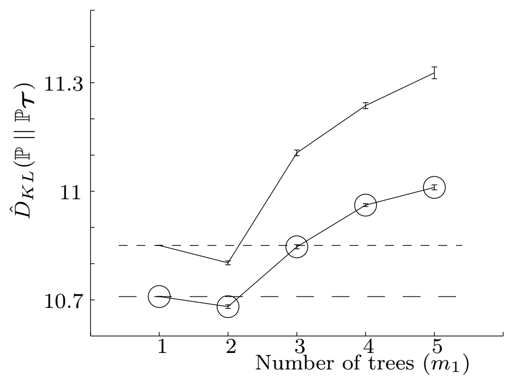
Figure 7.5 displays for growing values of N the mean KL divergence of MT-EM and MT-EM-Bag as a function of m_1 and with $m_2 = 10$. Note that when $m_1 = 1$, MT-EM and MT-EM-Bag are respectively equivalent to the Chow-Liu algorithm and MT-Bag (with $m = m_2 = 10$). The results of those methods are outlined by two constant lines on the figures. Those results were obtained on 5 networks times 6 data sets for each N.

At 120 samples the Chow-Liu algorithm is better than MT-EM: because of the low number of samples, the negative impact of an increased variance is not compensated by the gain in bias when increasing m_1. Using MT-EM-Bag rather than MT-EM (with $m_1 = 2$) leads to a better accuracy than Chow-Liu, but is worse than MT-Bag alone. The two-level mixtures cannot completely compensate for the increased variance brought by the partionning of the learning set at the first level of the mixture.

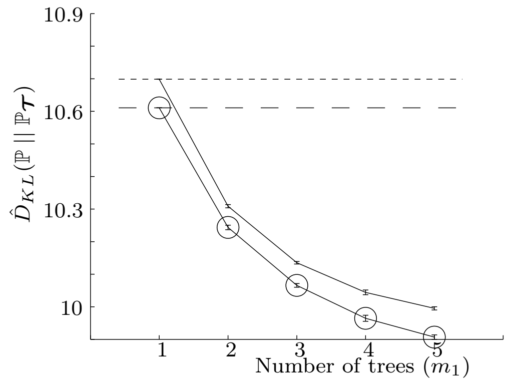
As the number of samples increases, the variance of all models decreases, and the smaller bias of larger mixtures progressively gives them the advan-



(a) $N = 120$ samples



(b) $N = 2000$ samples



(c) $N = 8000$ samples

Figure 7.5: Setting 2: On 1 run times 5 distributions times 6 sets, increasing m_1 ($m_2 = 10$) reduces the bias and increases the variance. While a mixture is advantageous only for N large enough, MT-EM-Bag is always better than MT-EM.

tage over Chow-Liu. Constructing an optimal mixture of size 2 is more interesting than a Chow-Liu tree for $N = 2000$ samples. At this point however, reducing the variance of Chow-Liu is still more interesting since MT-Bag is better than MT-EM for all m_1.

As more and more samples are available, the variance becomes even lower (MT-Bag is closer to Chow-Liu), and reducing the bias becomes preferable to lowering the variance. At 8000 samples, MT-EM is definitely better than MT-Bag and the KL divergence decreases when m_1 increases. Nevertheless, MT-EM-Bag is still better than MT-EM, showing that reducing the variance of each term is still interesting.

Moreover, we observe that the gap between MT-EM and MT-EM-Bag is widening as m_1 increases. This is logical, since increasing m_1 means building each second-level mixture on fewer samples, increasing the variance, a situation increasingly favoring it over a single Chow-Liu tree.

Realistic BN Target Distributions

I repeated the experiments performed with the mixture of trees on more complex and realistic target problems, i.e. 9 reference high-dimensional networks originating from various domains. For each problem, I used 5 datasets times 5 random initializations for sample sizes of 200, 500 and 5 random initializations for one set of 2500 samples (constructed by combining the 5 sets of 500 samples).

I report in Table 7.1 the main characteristics of the data sets (data set name, number of variables p, and range of cardinalities $|X_i|$ of the variables), together with the number of runs where each method constructed a more precise mixture than the two others, for the three learning sample sizes of respectively $N = 200$, $N = 500$ and $N = 2500$. In these experiments, $m_1 = 2$ and $m_2 = 10$ unless other values are explicitly specified.

The first observation about these results is that the accuracies for $N = 200$ samples seem to paint a different picture than what was observed in the previous sections: on 4 distributions (Gene, Hailfinder10, Lung Cancer and Pigs) MT-EM is indeed almost always more precise than both two-level variants (MT-EM-Bag and MT-EM-BagCl) based on $m_2 = 10$ terms at the lower level. There are two possible explanations for this behavior: either the sample size is so small (the Chow-Liu algorithm is better than MT-EM) that the bias increase due to the subsampling of the bagging procedure

Table 7.1: Setting 3: Best methods on realistic data sets on 5 runs times 5 sets (respectively 10 runs times 1 set) of 200 or 500 (respectively 2500) samples, with $m_1 = 2$ and $m_2 = 10$. M1 $\equiv$ MT-EM, M2 $\equiv$ MT-EM-Bag, M3 $\equiv$ MT-EM-BagCl.

$\mathbb{P}$	p	$\lvert X_i \rvert$	$N = 200$			$N = 500$			$N = 2500$		
			M1	M2	M3	M1	M2	M3	M1	M2	M3
Alarm10	370	2-4	3	5	17	-	6	19	-	8	2
Child10	200	2-6	-	1	24	-	4	21	-	4	6
Gene	801	3-5	25	-	-	-	9	16	-	8	2
Hailfinder10	560	2-11	25	-	-	25	-	-	-	1	9
Insurance10	270	2-5	2	1	22	1	9	15	-	7	3
Link	724	2-4	3	7	15	-	13	12	-	8	2
Lung Cancer	800	2-3	25	-	-	8	-	17	-	8	2
Munin	189	1-21	1	15	9	6	5	14	-	5	5
Pigs	441	3-3	21	1	3	-	16	9	-	9	1
ALL			105	30	90	40	62	123	0	58	32

becomes strongly detrimental, and/or the small sample size combined with the high dimension p of the problems leads to a much higher variance of the lower level mixtures, given their relatively small size of $m_2 = 10$.

In order to rule out this latter hypothesis, a few additional runs of the three methods were performed on some target distributions for other values of m_2. In some complementary tests, the value of m_2 was gradually increasing up to 350. These results confirm that the accuracy of the two level mixtures keeps increasing with m_2. In particular, I found that at $m_2 = 350$, it has not yet converged to its minimum value, but in several cases outperforms then the MT-EM baseline, including on problems for which this does not happen for $m_2 = 10$ (e.g. on Lung Cancer and Hailfinder10).

As an illustration, Figure 7.6 reports the negative log-likelihood gain of the two-level mixtures with respect to MT-EM and for increasing m_2. Even though all two-level mixtures are worse than MT-EM for $m_2 = 10$, their accuracy improves with m_2. At $m_2 = 350$, 3 two-level mixtures are better than MT-EM, and the accuracy has not converged yet. Similar observations were made on Hailfinder10, Insurance10, Child10, for m_2 increasing up to 100. On Munin, the two-level mixture is still improving with m_2 but only marginally. Based on these observations, the high variance, rather than the bias, seem to have the main reason why MT-EM was better than than the two-level mixtures in a few experimental settings.

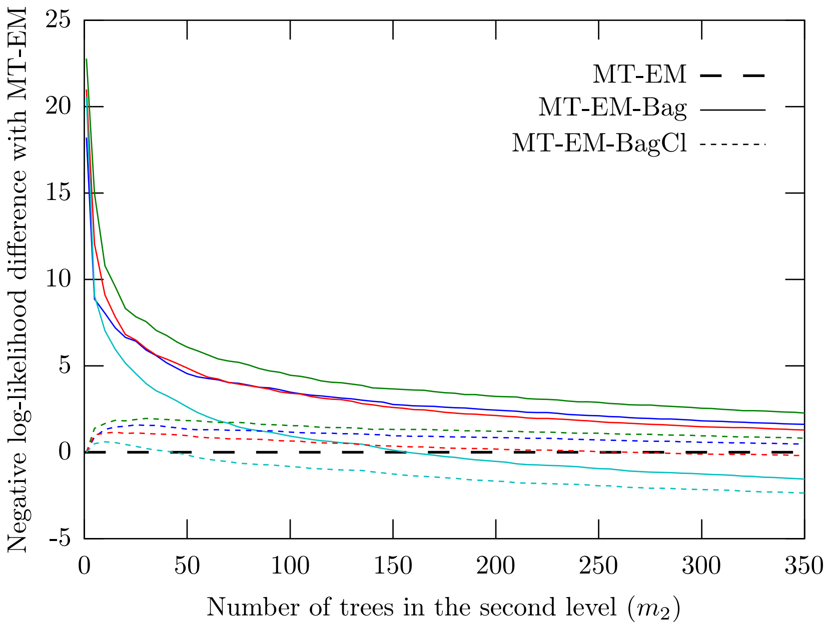


Figure 7.6: Setting 3: The accuracy of the two level mixture keeps decreasing as m_2 increases, on the LungCancer distribution. The mixture has not converged yet, and for $m_2 = 350$ and $N = 200$, 3/8 are better than MT-EM, an improvement over the number (0/25) observable when $N = 200$ (Table 7.1).

The second observation we make from the results of Table 7.1 is that as the learning sample size N increases to 500 and then 2500, the two-level mixtures become systematically better than the baseline MT-EM method. For the intermediate sample size of $N = 500$, MT-EM-BagCl is clearly better, while for $N = 2500$ MT-EM-Bag is more often the best method.

7.3.3 Variations of the Second Level Mixture

The experiments described so far show the advantage in terms of accuracy of the two-level mixtures over MT-EM, when the best variance reducing mixture (bagging) is used at the second level. Now, the best two alternative methods developed in the previous chapter will be tested in place of bagging: the warm start inertial heuristic (MWISH-B) and the skeleton-based approximation method (Mskl-B).

The objective is both to see how well these two methods fare in the context of two-level mixtures and to compare them to bagging in a different setting, to check the conclusions of the previous chapter.

As with MT-Bag, two variants of each method will be tested, depending on the learning algorithm used to learn the first model, either the original learning set or a bootstrap replicate. Table 7.2 contains a synthetic list of the different methods compared in the upcoming experiments.

Table 7.2: Overview of the different variants of the two-level mixtures evaluated.

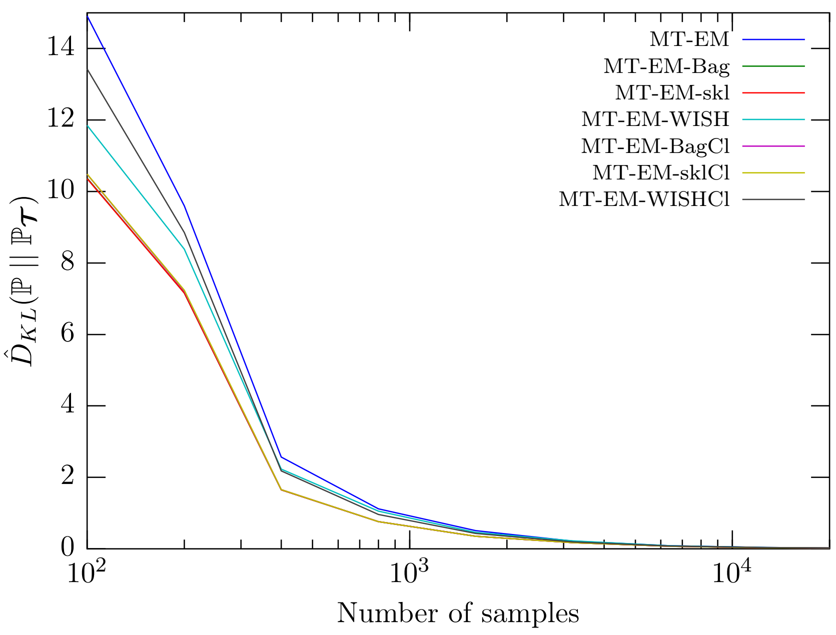
D for first structure	base algorithm for second level		
	MCL-B	Mskl-B	MWSI-B
bagging	MT-EM-Bag	MT-EM-skl	MT-EM-MWSI
original	MT-EM-BagCl	MT-EM-sklCl	MT-EM-MWSICl

These methods will be evaluated both on the mixture of three Markov trees and on the realistic distributions already used.

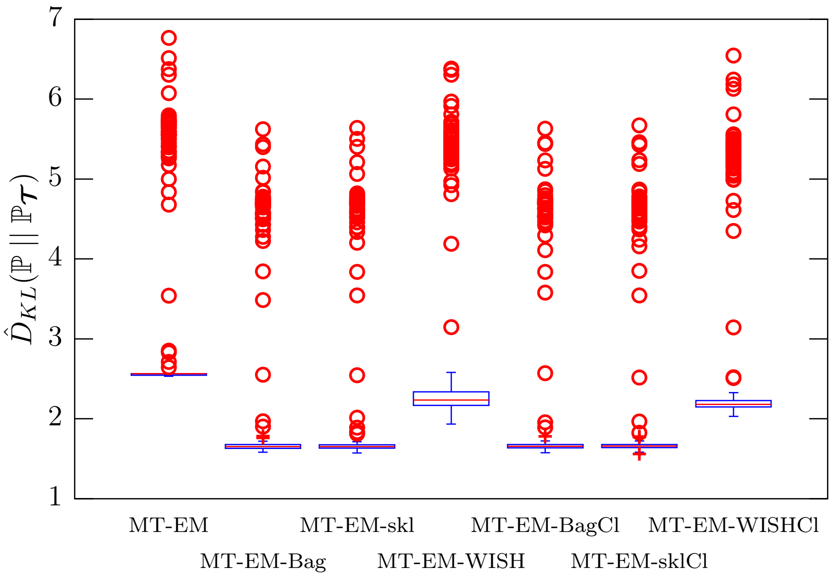
The Target is a Mixture of 3 Trees

Figure 7.7 contains an illustration of the same results as those presented for 3 methods only (Section 7.3.1), but now with 4 additional methods.

- Figure 7.7a displays the average Kullback-Leibler divergence to the target model for an increasing number of samples.
- Figure 7.7b shows how the distribution of this same measure of accuracy varies from one run to the other on 400 observations.
- Figure 7.7c summarizes how often each method achieved the best accuracy, for an increasing number of samples.
- Figure 7.7d contains the average number of edges similar to the original model, for an increasing number of samples.

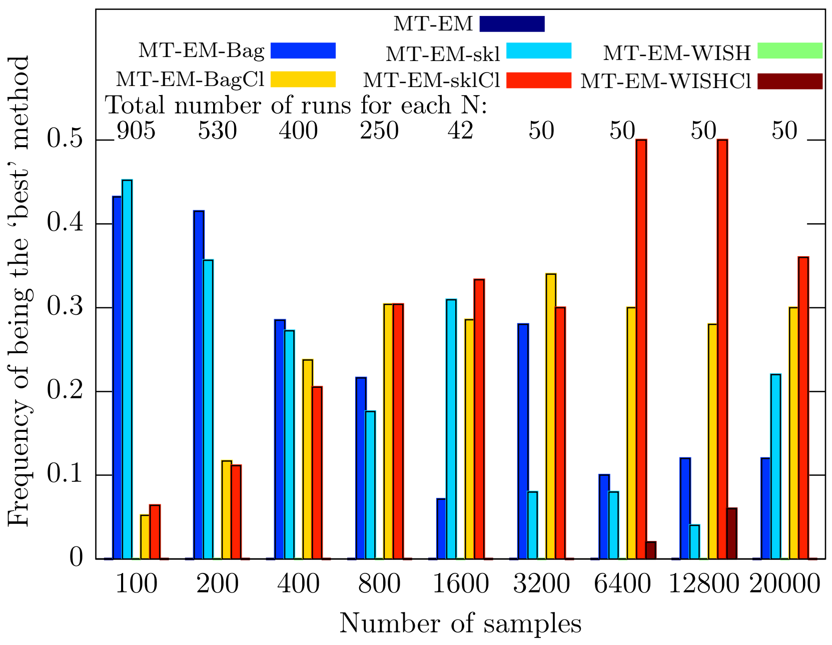


(a) Average KL divergence with respect to the target mixture of 3 trees.

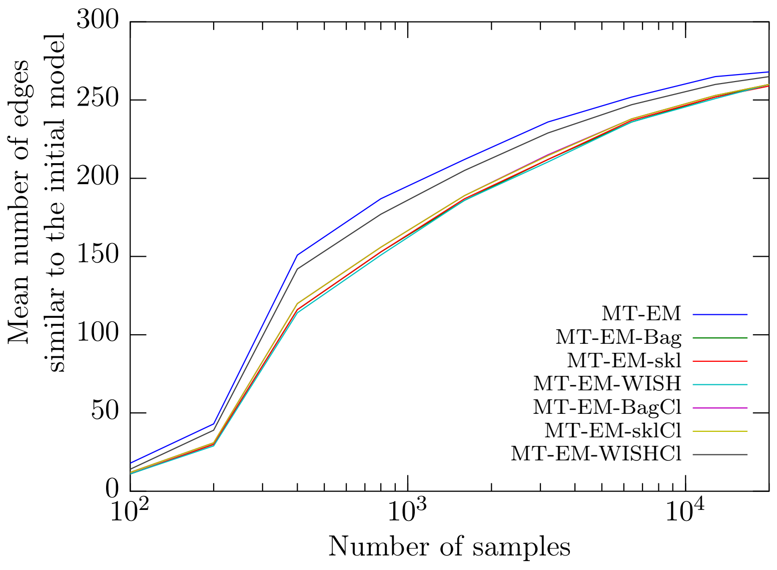


(b) Distribution of the KL divergences of the models produced over 400 runs on a learning set with $N = 400$

Figure 7.7: Setting 1: On 1 run times 5 distributions times 6 sets, 2-level mixtures based on Mskel-B and MCL-B (both with or without a first CL tree) perform similarly in terms of accuracy.



(c) Relative number of runs where each method is the best, displayed by number of samples



(d) Mean number of edges recovered with respect to the target mixture of 3 trees. MT-EM is the best method, followed by MT-EM-WISHCl. The other 5 methods are indistinguishable, and achieve a smaller score.

Figure 7.7: (continued) Setting 1: On 1 run times 5 distributions times 6 sets, 2-level mixtures based on Mskel-B and MCL-B (both with or without a first CL tree) perform similarly in terms of accuracy.

Based on these results, the seven methods considered can be separated into three subsets. The first category contains only MT-EM, the initial method. This method still achieves the worse accuracy of the seven methods, but the best edge structure similarity to the initial model. MT-EM-Bag, MT-EMskl and their variants form the second category. Finally, the two variants of MT-EM-WISH constitutes the last category.

In the second category, MT-EM-Bag and MT-EM-skl cannot be distinguished, and neither can MT-EM-BagCl and MT-EM-sklCl. The methods inside each of these pairs behave similarly. This is particularly visible in the accuracy plots (Figures 7.7a,b), where all four methods have the same accuracy, and in the plot displaying the best methods (Figure 7.7c). In this latter figure, the first pair of methods seems for a low number of samples superior to the second one, but this advantage first disappears, and then is inverted when the number of observations increases, exactly as what was observed for three methods only. The similar behavior of the two regular bagging methods (MT-EM-Bag and MT-EM-BagCl) and the two skeleton-based methods (MT-EM-skl and MT-EM-sklCl) means the latter is a good approximation of the former method.

Methods of the second category seems to lie in between. Their accuracy is worse than the third category, but better than MT-EM, although MT-EM-WISHCl is for a very few number of runs sometimes the best method (Figure 7.7c). The behavior of this category is however different for the edge similarity. MT-EM-WISH score is as bad as the score achieved by the second category of methods, but MT-EM-WISHCl is actually closer to MT-EM than the second category. This is likely due to the initialization of the method by the Chow-Liu tree, and to the reuse of these edges in subsequent trees. For a larger number of terms in the second-level mixture, the score of this method can be expected to converge to the score of MT-EM-WISH, at the same level than the second category.

Realistic Distributions

Table 7.3 lists the best performing methods in respectively 25, 25 and 5 runs, i.e. 5 runs times respectively 5, 5 and 1 learning set(s), for all target distributions and for respectively 200, 500 and 2500 learning samples. The number of runs on the learning sets of 2500 samples is therefore half the number used for the analysis of the first three methods considered. The convergence points of the EM algorithm are identical to those used in Section 7.3.2.

If the six two-level mixtures are grouped by their first tree (CL or not), the results are quite similar to those displayed in Table 7.1. The number of runs where MT-EM is the best method very slightly decreases, because it is compared to more methods. If anything, this status quo conforts the belief that the two categories of second level mixtures, defined in the previous section, each contain methods behaving similarly.

The methods based on WISH once again exhibit a poorer accuracy than methods of the second category. However, they achieve honorable results on the Link distribution, for 200 observations.

7.4 Conclusion

This chapter proposes a new method to combine the two frameworks for building mixtures. The research conducted in this thesis was motivated by the bias-variance decomposition of the error, and the capacity of mixtures of models to decrease the variance of a single model. New algorithms were presented in the previous chapter. Mixtures of Markov trees have however also been used to decrease the bias.

In this chapter, the two frameworks are combined, resulting in a two-level mixture model that can effectively reduce the error on both small and larger numbers of observations. Indeed, the experiments reported in this chapter show how reducing the variance leads to an increase in accuracy at low sample size and how mixtures targeting the bias worsen the model at low sample size but increase it when the number of samples increases.

The combined method proposed here combines the advantages of both methods, allowing a better accuracy than any mixture alone at certain number of samples. In particular, it improves the bias reducing mixture most of the time, and the improvement increases with the number of terms in this mixture.

Moreover, two-level mixtures based on either bagging or the skeleton approximation have a very similar behavior

Table 7.3: Best methods on realistic data sets on 5 runs times 5 sets of 200 (up) or 500 (down) samples, $m_1 = 2, m_2 = 10$.

N	Data set	MT-EM	-Bag	-skl	-WISH	-BagCl	-sklCl	-WISHCl
	Alarm10	2	2	6	-	8	7	-
	Child10	-	1	-	-	12	12	-
	Gene	24	-	-	-	-	-	1
	Hailfinder10	24	-	-	-	-	1	-
200	Insurance10	2	-	4	-	10	9	-
	Link	3	3	-	3	3	5	8
	Lung Cancer	25	-	-	-	-	-	-
	Munin	-	3	8	-	2	12	-
	Pigs	21	1	-	-	-	3	-
	Alarm10	-	3	6	-	3	13	-
	Child10	-	1	2	-	12	10	-
	Gene	-	6	3	-	9	7	-
	Hailfinder10	25	-	-	-	-	-	-
500	Insurance10	1	4	6	-	5	9	-
	Link	-	5	7	-	6	6	1
	Lung Cancer	7	-	-	-	10	8	-
	Munin	3	3	4	1	4	10	-
	Pigs	-	8	9	-	3	5	-
	Alarm10	-	3	2	-	-	-	-
	Child10	-	-	3	-	1	1	-
	Gene	-	1	3	-	1	-	-
	Hailfinder10	-	-	1	-	1	3	-
2500	Insurance10	-	4	-	-	-	1	-
	Link	-	2	2	-	-	1	-
	Lung Cancer	-	2	2	-	1	-	-
	Munin	-	-	2	-	1	2	-
	Pigs	-	2	3	-	-	-	-

Part III

Perspectives for multitarget prediction

Chapter 8

A Proposal for Mixtures of Tree-structured CRFs

The goal of standard supervised classification is to predict a single class or target variable $\mathcal{C}$ based on the known values of a set of observation or input variables $\boldsymbol{\mathcal{X}}$. However, in some practical applications, there are more than one variable to predict. This constitutes a multitarget prediction problem, also called multivariate classification. In this setting, the output of the learning procedure is a function associating to each configuration of $\boldsymbol{\mathcal{X}}$ a value for each variable of the variable set $\boldsymbol{\mathcal{C}}$, the class variables.

For example, one may want to associate to any pixel of an image a value 1 if the pixel belongs to an object of interest (a face, a car...) or 0 otherwise. In the biomedical domain, one may wish to predict, for any molecule, whether it will or will not be active on different types of cancer cells.

When $\boldsymbol{\mathcal{C}}$ is a set of discrete variables, the multivariate problem can be reformulated as a standard classification problem by defining a new class variable $\mathcal{C}$ taking a different value for each configuration of $\boldsymbol{\mathcal{C}}$. However the cardinality of $\mathcal{C}$ is exponential in the size of $\boldsymbol{\mathcal{C}}$, making this approach unsuitable when $|\boldsymbol{\mathcal{C}}|$ is large.

Another simple way to construct a multivariate classifier is to decompose the learning problem into $|\boldsymbol{\mathcal{C}}|$ standard classification problems, the ith one predicting the value of $\mathcal{C}_i \in \boldsymbol{\mathcal{C}}$ based on $\boldsymbol{\mathcal{X}}$. However, this latter approach does not take into account the relationships between the class variables, which may be necessary to reach a sufficient accuracy.

A Conditional Random Field (CRF) is a probabilistic graphical model that encodes a joint conditional probability density $\mathbb{P}(\boldsymbol{\mathcal{C}}|\boldsymbol{\mathcal{X}})$ [LMP01]. This type of model is thus particularly suited for multitarget prediction: to each input $\mathbf{x}$ is associated the most probable configuration $\mathbf{c}$, obtained by MAP inference on the distribution $\mathbb{P}(\boldsymbol{\mathcal{C}}|\boldsymbol{\mathcal{X}} = \mathbf{x})$.

As it is the case for other classes of probabilistic graphical models, constraining the structure of the CRF to a tree yields some advantages both for learning and for inference (also called prediction in this context), and recent research has tried to develop algorithms for learning such models, see e.g. [SCG09, BG10].

In this chapter, I report preliminary investigations about using a mixture approach to enhance those tree-structured CRFs. This new research direction is motivated both by the origin of this work, which was inspired by the success of perturb and combine framework in standard classification, and by the desire to construct mixtures of densities in a setting where the evaluation is more interpretable. Indeed, a percentage of misclassifications is easier to understand than a Kullback-Leibler divergence.

The first step (Section 8.1) consists in the formal definition of Conditional Random Fields and an introduction to the problem of learning them from a dataset. Then (Section 8.2), I focus on tree-structured CRFs and their interesting properties. Next, I discuss mixtures of Conditional Random Fields, and presents the state-of-the-art (Section 8.3). Finally, I propose a new meta-algorithm for learning mixtures of CRFs (Section 8.4), inspired by the approaches developed for mixtures of Markov trees, and discusses its subroutines. This meta-algorithm is neither instantiated nor empirically evaluated.

8.1 Conditional Random Fields

In contrast with Bayesian networks that encode a joint probability distribution over the variables, a conditional random field directly encodes a conditional probability distribution $\mathbb{P}(\boldsymbol{\mathcal{C}}|\boldsymbol{\mathcal{X}})$ [LMP01]. This model is thus well suited to the prediction of $\boldsymbol{\mathcal{C}}$ based on $\boldsymbol{\mathcal{X}}$.

Other classes of probabilistic graphical models encoding a joint distribution $\mathbb{P}(\boldsymbol{\mathcal{C}}, \boldsymbol{\mathcal{X}})$ can also answer a multivariate classification problem by performing inference to compute $\mathbb{P}(\boldsymbol{\mathcal{C}}|\boldsymbol{\mathcal{X}})$. The main advantage of CRF over those latter models is that no resource is wasted on the modelling of the distribution of the input variables $\boldsymbol{\mathcal{X}}$. In some instances, e.g. in image processing, when these variables $\boldsymbol{\mathcal{X}}$ are complex, modelling their joint distribution is very challenging. This motivated the development of conditional models [LMP01].

8.1.1 Definition

[SM07a] defines a CRF as follows. "Let $\mathcal{G}$ be a factor graph over $\boldsymbol{\mathcal{C}}$. Then $\mathbb{P}(\boldsymbol{\mathcal{C}}|\boldsymbol{\mathcal{X}})$ is a conditional random field if for any fixed $\boldsymbol{\mathcal{X}} = \mathbf{x}$, the distribution $\mathbb{P}(\boldsymbol{\mathcal{C}}|\boldsymbol{\mathcal{X}} = \mathbf{x})$ factorizes according to $\mathcal{G}$."

A Conditional Random Field can therefore be represented by a hypergraph $\{\{\boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{C}}\}; \{\boldsymbol{\mathcal{S}}_i\}_{i=1}^k\}$ defined on a set of nodes labeled by $\boldsymbol{\mathcal{C}} \cup \boldsymbol{\mathcal{X}}$. Each hyperedge i of the CRF connects a set of variables that will be denoted by $\boldsymbol{\mathcal{S}}_i$, with $\boldsymbol{\mathcal{S}}_i \subset (\boldsymbol{\mathcal{C}} \cup \boldsymbol{\mathcal{X}})$. Each hyperedge must contain at least one class variable: $\boldsymbol{\mathcal{C}}_{\boldsymbol{\mathcal{S}}_i} \equiv \boldsymbol{\mathcal{S}}_i \cap \boldsymbol{\mathcal{C}} \neq \emptyset$. Following a similar notation, $\boldsymbol{\mathcal{X}}_{\boldsymbol{\mathcal{S}}_i} \equiv \boldsymbol{\mathcal{S}}_i \cap \boldsymbol{\mathcal{X}}$.

A Conditional Random Field encodes a conditional probability distributions (one joint probability distribution over $\boldsymbol{\mathcal{C}}$ per configuration of $\boldsymbol{\mathcal{X}}$) $\mathbb{P}(\boldsymbol{\mathcal{C}}|\boldsymbol{\mathcal{X}})$ by a normalized product of factors $\phi_1, \ldots, \phi_k$ defined on its k hyperedges:

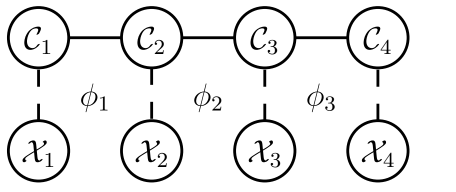
$$\mathbb{P}(\boldsymbol{\mathcal{C}}|\boldsymbol{\mathcal{X}}) = \frac{1}{Z(\boldsymbol{\mathcal{X}})} \prod_{i=1}^{k} \phi_i(\boldsymbol{\mathcal{S}}_i) \tag{8.1}$$

$$Z(\boldsymbol{\mathcal{X}}) = \sum_{Val(\boldsymbol{\mathcal{C}})} \prod_{i=1}^{k} \phi_i(\boldsymbol{\mathcal{S}}_i) \ . \tag{8.2}$$

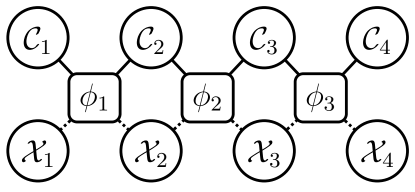
Each factor is a function $\phi_i(.) : \boldsymbol{\mathcal{S}}_i \to \mathbb{R}^+$. They are discussed in Section 8.1.1.

Each instance $\mathbf{x}$ of $\boldsymbol{\mathcal{X}}$ specializes the CRF into a different Markov Random Field defined over the classes $\boldsymbol{\mathcal{C}}$, whose structure $\mathcal{G}$ is the subhypergraph of the CRF induced by $\boldsymbol{\mathcal{C}}$: $\left\{\boldsymbol{\mathcal{C}}; \{\boldsymbol{\mathcal{C}}_{\boldsymbol{\mathcal{S}}_i}\}_{i=1}^k\right\}$ and whose parameters depend on $\mathbf{x}$. The normalization constant $Z(\boldsymbol{\mathcal{X}})$ also depends on $\mathbf{x}$. To assign a value to the classes based on the observations, inference must be performed in the specialised Markov Random Field. This first requires the computation of the partition function for the corresponding $\mathbf{x}$.

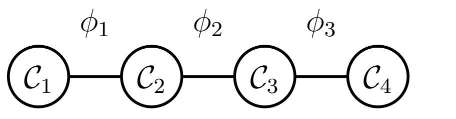
Because inference is performed only on the derived MRF mentionned in the previous paragraph, the CRF hyperedges are often divided in two components. In this work, the *structure* designates the hyperedges of the



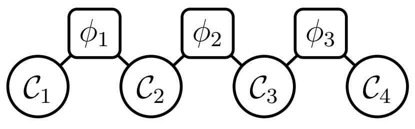
(a) One factor is associated to each pair of class variables linked by an edge. This factor depends on these output variables and on any input variable linked to any of these two class variables.



(b) The same CRF as in Figure 8.1a, in a more traditional factor graph representation.



(c) Once the values of the four observations are known, the model becomes a classical Markov random field.



(d) The same MRF as in Figure 8.1c, in a more traditional factor graph representation.

Figure 8.1: This pairwise CRF encodes a set of conditional probability distributions
$\mathbb{P}(\boldsymbol{\mathcal{C}}|\boldsymbol{\mathcal{X}}) = \frac{1}{Z(\boldsymbol{\mathcal{X}})}\phi_1(\mathcal{C}_1,\mathcal{C}_2,\mathcal{X}_1,\mathcal{X}_2)\phi_1(\mathcal{C}_2,\mathcal{C}_3,\mathcal{X}_2,\mathcal{X}_3)\phi_1(\mathcal{C}_3,\mathcal{C}_4,\mathcal{X}_3,\mathcal{X}_4)$.

Markov Random Field $\{\boldsymbol{\mathcal{C}};\{\boldsymbol{\mathcal{C}}_{\boldsymbol{\mathcal{S}}_i}\}_{i=1}^{k}\}$, and the *feature mapping* the association between input and output variables, i.e. the $\boldsymbol{\mathcal{X}}_{\boldsymbol{\mathcal{S}}_i}$ associated to each $\boldsymbol{\mathcal{C}}_{\boldsymbol{\mathcal{S}}_i}$.

When the structure of the MRF is a graph, i.e. when the potentials are defined on at most two output variables, the conditional random field is called a *pairwise* CRF.

To illustrate these notions, a CRF defined over four class variables and four input variables is represented in Figure 8.1.

A Closer Look at the Factors

Each factor is a function $\phi_i(.) : \boldsymbol{\mathcal{S}}_i \to \mathbb{R}^+$. A factor is often defined in terms of its energy function $\epsilon_i(\boldsymbol{\mathcal{S}}_i)$:

$$\phi_i(\boldsymbol{\mathcal{S}}_i) = \exp(-\epsilon_i(\boldsymbol{\mathcal{S}}_i)) \tag{8.3}$$
$$\epsilon_i(\boldsymbol{\mathcal{S}}_i) = \boldsymbol{w}_i^T \boldsymbol{f}_i(\boldsymbol{\mathcal{S}}_i) \ , \tag{8.4}$$

with $\boldsymbol{w}_i, \boldsymbol{f}_i(\boldsymbol{\mathcal{S}}_i) \in \mathbb{R}^{q_i}$. In that case, the probability distribution $\mathbb{P}(\boldsymbol{\mathcal{C}}|\boldsymbol{\mathcal{X}})$ of equation 8.1 can be written as a log-linear model

$$\mathbb{P}(\boldsymbol{\mathcal{C}}|\boldsymbol{\mathcal{X}}) = \frac{1}{Z(\boldsymbol{\mathcal{X}})} \exp\left(-\sum_{i=1}^{k} \boldsymbol{w}_i^T \boldsymbol{f}_i(\boldsymbol{\mathcal{S}}_i)\right) , \tag{8.5}$$

where $\boldsymbol{w} = \{\boldsymbol{w}_i\}_{i=1}^k$ are called the weights or the parameters of the model and $\{\boldsymbol{f}_i(\boldsymbol{\mathcal{S}}_i)\}_{i=1}^k$ the features.

Those features $\boldsymbol{f}_i(.) : \boldsymbol{\mathcal{S}} \to \mathbb{R}^{q_i}$ can be simple, e.g. correspond to the value of the variables $\boldsymbol{\mathcal{X}}$, or much more complicated, e.g. properties of a set of pixels in an image such as the shape, the texture etc.

8.1.2 Learning

Learning a CRF from a set of samples $D = \{\mathbf{x}_{D_s}, \mathbf{c}_{D_s}\}_{s=1}^N$ can be decomposed into three non-trivial steps [BG10]:

a constructing a feature mapping, i.e. an ensemble of candidate features $\boldsymbol{f}_i(\boldsymbol{\mathcal{C}}_{\boldsymbol{\mathcal{S}}_i}, \boldsymbol{\mathcal{X}}_{\boldsymbol{\mathcal{S}}_i})$ (and an associated subset of input variables $\boldsymbol{\mathcal{X}}_{\boldsymbol{\mathcal{S}}_i}$) for every prospective hyperedge $\boldsymbol{\mathcal{C}}_{\boldsymbol{\mathcal{S}}_i}$ of the underlying MRF,

b learning the structure (selecting a subset of features $\{\boldsymbol{f}_i(.)\}_{i=1}^k$ defined in step a, to keep in the final model; and thus selecting a subset of hyperedges)

c and learning the parameters $\{\boldsymbol{w}_i\}_{i=1}^k$ given a specified structure and features $\{\boldsymbol{f}_i(.)\}_{i=1}^k$.

The second step is sometimes trivial when the hyperedges associated to the features considered already define an acceptable structure, e.g. it is common in part of speech tagging (identify nouns, verbs adverbs etc in a written sentence) to use a chain CRF. As this structure arises naturally based on the problem, only pairwise features defined on neighboring words may be selected in the first step, directly resulting in a chain structure. When the hyperedges associated to the features do not define an acceptable structures, some of the hyperedges (and the related features) must be discarded.

The last two steps are sometimes performed together, e.g. the parameters of a model containing all candidate features are optimized based on a regularized score penalizing model complexity. The features whose corresponding weights are null are removed from the model.

Nevertheless, I will briefly present those three steps, with a focus on structure learning.

Feature Mapping

Determining the features $\boldsymbol{f}_i(\boldsymbol{\mathcal{C}}_{\boldsymbol{\mathcal{S}}_i}, \boldsymbol{\mathcal{X}}_{\boldsymbol{\mathcal{S}}_i})$ means associated a subset of input variables $\boldsymbol{\mathcal{X}}_{\boldsymbol{\mathcal{S}}_i}$ to every subset of output variables $\boldsymbol{\mathcal{C}}_{\boldsymbol{\mathcal{S}}_i}$ that will be considered as a (candidate) hyperedge in the structure learning phase.

Expert knowledge is often used to extract relevant subsets of variables and/or relevant features. As an example, when the problem considered is pixel labeling in an image, the features may be the average color, shape etc of the associated and neighboring pixels (the relevant subset of inputs).

When expert knowledge is unavailable, some automatic procedure can be used [BG10, MVC12].

Learning the Parameters

The problem of learning the parameters can be stated as finding the optimal values for the weights $\mathbf{w}$ based on a training set of instances D. However, solving this problem is a difficult task, because it requires many inference operations on the network [CG10, BG10], and because inference might be difficult, depending on the tree-width of the model (see Section 3.4.1 or [Dar09, KF09]).

For more details about parameter learning, see e.g. [LMP01, SM05, SM07b, DL10].

Learning the Structure

The structure of the CRF consists in the relationships between the class variables $\boldsymbol{\mathcal{C}}$. Given the relevant $\boldsymbol{\mathcal{X}}_{\boldsymbol{\mathcal{S}}_i}$ for each candidate hyperedge $\boldsymbol{\mathcal{C}}_{\boldsymbol{\mathcal{S}}_i}$ and the sample set, learning the structure means selecting the hyperedges to be included in the CRF.

As for the features, constructing the structure of a CRF is usually based on expert knowledge, and both the relationships between variables and/or the features are specified [BG10]. In natural language processing, chain CRFs or an extension of those models are often used, a grid structure is common in image labeling etc.

Learning the structure automatically is a difficult task, because it usually requires the evaluation of the parameters of several structures, a procedure that in itself is already difficult (see previous section) [SCG09, CG10, BG10].

Nevertheless, a few algorithms have been proposed to learn CRF structures. Two of these algorithms are presented here. Specific algorithms for learning tree-structured CRF are described in Section 8.2 and additional randomized procedures are described in Section 8.3.

Both algorithms jointly optimize the structure and the parameters by maximizing a regularised score (see Section 8.1.2) of the form:

$$score(D, \mathcal{G}, \boldsymbol{w}_{\mathcal{G}}) - \lambda||\boldsymbol{w}||, \tag{8.6}$$

where the first term measures the quality of the model and the second penalizes model complexity. The subscript of $\boldsymbol{w}_{\mathcal{G}}$ emphasis that the weights not present in $\mathcal{G}$ are null.

[SMFR08] considers a complete graph and couples the optimization of the parameters with a L1 block regularizer, i.e. $||\mathbf{w}||$ is the L1-norm. The set of all parameters associated to an hyperedge forms a block, and a block where all the parameters are null translates into a removal of the edge from the graph.

[ZLX10] uses a L1-norm as well, but progressively increases the number of features used to compute the score. This is motivated by concerns for the quality of the results of the approximate inference algorithms used for parameter learning when they have to run on dense graphs, even if most feature weights will be null. Therefore they start with an empty set of features, and alternatively apply one step of a gradient descent algorithm and the addition in the model of the "best" K (a user defined parameter) features whose subgradients are the largest.

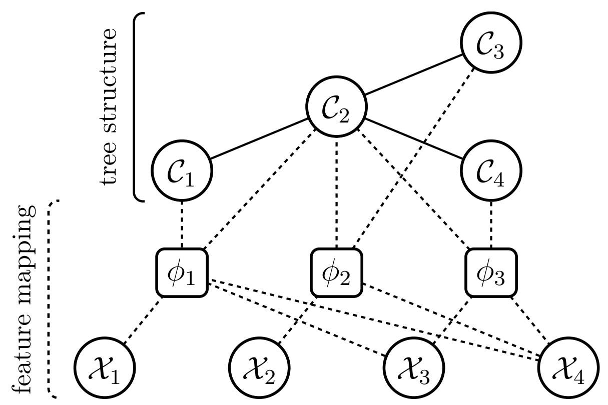


Figure 8.2: This pairwise tree-structured CRF is defined over 4 class variables $\boldsymbol{\mathcal{C}}$.

8.2 Tree-structured CRFs

A tree-structured CRF is a pairwise CRF whose structure is a tree. With respect to more general structures, tree-structured CRFs have the advantage that exact inference is tractable, as for Bayesian networks. Indeed, exact inference in a tree-CRF can lead to more accurate results than approximate inference on a model learned in a wider class of candidate structures [SCG09].

Tractability of inference in tree-structured CRFs motivated the recent development of algorithms to learn such models [SCG09, BG10]. The main ideas of those algorithms will be briefly presented below. Both algorithms output the graph $(\boldsymbol{\mathcal{C}}, \{\boldsymbol{\mathcal{C}}_{\boldsymbol{\mathcal{S}}_i}\}_{i=1}^{k})$ based on a learning set $\{\mathbf{x}_{D_s}, \mathbf{c}_{D_s}\}_{s=1}^{N}$. They both assume that the feature mapping is known, and that only pairwise or univariate potentials are considered. Potentials with more than 2 class variables are incompatible with the tree structure.

8.2.1 Learning a Tree-structured CRF via Graph Cuts

The algorithm proposed in [SCG09] is actually more general, since it produces a pairwise CRF of arbitrary tree-width k. When $k = 1$, it outputs a tree-structured CRF. It takes as input a weighted graph $\mathcal{G}$ and removes weak edges until the model has the required tree-width. It selects the edges

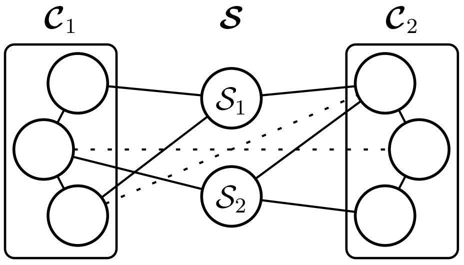


Figure 8.3: $\boldsymbol{\mathcal{S}}$ is the separator of the cut $(\boldsymbol{\mathcal{C}}_1, \boldsymbol{\mathcal{C}}_2|\boldsymbol{\mathcal{S}})$. Dashed lines are the cut edges.

to remove by using a minimum-weight cut algorithm. A cut $(\boldsymbol{\mathcal{C}}_1, \boldsymbol{\mathcal{C}}_2|\boldsymbol{\mathcal{S}})$ is a partition of $\boldsymbol{\mathcal{C}}$ into three disjoints subsets $\boldsymbol{\mathcal{C}}_1, \boldsymbol{\mathcal{C}}_2 \neq \emptyset$ and $\boldsymbol{\mathcal{S}}$ (illustrated in Figure 8.3). The weight of the cut is defined as

$$\sum_{O_j \in \boldsymbol{\mathcal{C}}_1, O_k \in \boldsymbol{\mathcal{C}}_2} w_{O_j, O_k}, \tag{8.7}$$

where w_{O_j,O_k} is the weight of the corresponding edge in $\mathcal{G}$, called the cut edges. The algorithm alternates between constructing a minimum-weight cut and removing the associated cut edges, until the model has reached the required tree-width.

When applying their methods to CRF, [SCG09] uses the conditional mutual information as edge-weight:

$$\begin{aligned}
w_{O_j,O_k} &= I(O_j, O_k|\boldsymbol{\mathcal{X}}) && (8.8)\\
&= \sum_{\mathbf{x}\in\boldsymbol{\mathcal{X}}} \mathbb{P}(\mathbf{x}) I(O_j, O_k|\mathbf{x}) && (8.9)\\
&= \sum_{\mathbf{x}\in\boldsymbol{\mathcal{X}}} \mathbb{P}(\mathbf{x}) \sum_{c_j \in \mathcal{C}_j} \sum_{c_k \in \mathcal{C}_k} \mathbb{P}(c_j, c_k|\mathbf{x}) \frac{\mathbb{P}(c_j, c_k|\mathbf{x})}{\mathbb{P}(c_j|\mathbf{x})\mathbb{P}(c_k|\mathbf{x})}. && (8.10)
\end{aligned}$$

Similarly to the optimization criterion of the Chow-Liu algorithm, using this weight optimizes the expected conditional log-likelihood of the learning set.

8.2.2 Alternatives to Conditional Mutual Information

[BG10] focuses on learning tree-structured conditional random fields based on the Chow-Liu algorithm [CL68]. Their main contribution is the development of alternatives to $I(O_j, O_k|\boldsymbol{\mathcal{X}})$ for scoring a prospective edge O_j, O_k. They are motivated by the fact that estimating $I(O_j, O_k|\boldsymbol{\mathcal{X}})$ based on a finite learning set is not scalable as the number of inputs increases, both from a computational and an accuracy point of view.

Instead, they propose two scores based on local inputs, i.e. $\boldsymbol{\mathcal{X}}_j$, $\boldsymbol{\mathcal{X}}_k$ and $\boldsymbol{\mathcal{X}}_{jk}$, the inputs respectively mapped to the prospective factors defined on $\mathcal{C}_j$, $\mathcal{C}_k$ and $\mathcal{C}_j \cup \mathcal{C}_k$. Those two scores are called Local Conditional Mutual Information (CMI) and Decomposable Conditional Influence (DCI).

The former score is the mutual information conditioned to the local relevant feature variables:

$$\text{CMI}(j,k) = I(\mathcal{C}_j, \mathcal{C}_k|\boldsymbol{\mathcal{X}}_{jk}) \tag{8.11}$$

$$\begin{aligned} &= \mathbb{E}(\log \mathbb{P}(\mathcal{C}_j, \mathcal{C}_k|\boldsymbol{\mathcal{X}}_{jk})) \\ &\quad - \mathbb{E}(\log \mathbb{P}(\mathcal{C}_j|\boldsymbol{\mathcal{X}}_{jk})) - \mathbb{E}(\log \mathbb{P}(\mathcal{C}_k|\boldsymbol{\mathcal{X}}_{jk})). \end{aligned} \tag{8.12}$$

The latter score differs by the (conditioned) marginal probability distribution of O_j and O_k , which are only conditioned on the relevant inputs for each variable:

$$\begin{aligned} \text{DCI}(j,k) &= \mathbb{E}(\log \mathbb{P}(\mathcal{C}_j, \mathcal{C}_k|\boldsymbol{\mathcal{X}}_{jk})) \\ &\quad - \mathbb{E}(\log \mathbb{P}(\mathcal{C}_j|\boldsymbol{\mathcal{X}}_{j})) - \mathbb{E}(\log \mathbb{P}(\mathcal{C}_k|\boldsymbol{\mathcal{X}}_{k})). \end{aligned} \tag{8.13}$$

Based on the analysis of some problematic target models and on their experiments, they conclude that the second score is preferable.

8.3 Mixtures of CRFs: State-of-the-art

Conditional random fields lie at the intersection between classification and density estimation. Mixture models and perturb and combine have successfully improved those two fields, but have only been slightly exploited for conditional random fields structure learning. In this chapter the concept of mixture of conditional random fields is investigated.

The perturb and combine framework has already been exploited for conditional random fields learning from a sample set, but mostly to learn a single model without constraint, i.e. a model that will not necessarily permit exact inference. For example, [PT10] uses random forests to perform inference on a CRF. More precisely, this article develops a Metropolis-Hastings algorithm for inference, and this algorithm relies on random forests to estimate the probability to accept a transition.

Mixtures of tree-structured CRFs have also been proposed. [CG10] builds a different tree-structured CRF for every possible input $\mathbf{x}$. To do so, they select, based on $\mathbf{x}$, one tree structure spanning a superstructure (containing at most pairwise relationships between class variables), and force all features associated to edges not belonging to the selected tree to zero. However, since inference is performed on only one CRF, it could arguably not be considered a mixture of CRFs.

[POB09] starts with a given feature mapping and structure for the CRF. In order to perform inference, it uniformly samples m tree-structured CRFs $\mathcal{T}_k$ spanning the given CRF, and learns their parameters by optimizing the sum of conditional likelihood of the sample sets according to each tree, i.e.

$$\text{logll}_{D,\mathcal{C}|\mathcal{X}}(\mathbf{w}|\boldsymbol{\mathcal{T}}) = \sum_{j=1}^{m}\sum_{s=1}^{N} \ln \mathbb{P}_{\mathcal{T}_j}(\mathbf{c}_{D_s}|\mathbf{x}_{D_s}, \mathbf{w}). \tag{8.14}$$

As in [CG10], the parameters associated to an edge are identical in all tree structured-CRFs that have this edge.

Prediction is either performed by

- estimating the parameters of the full CRF by reweighting the parameters estimated for the set of tree-structured CRFs, and performing inference on the full CRF,
- by performing inference on each tree-structured CRF $\mathcal{T}_k$ and applying a voting strategy on this set of m predictions.

[MVC12] generates a mixture of tree-structured CRFs to label images. Each label (e.g. tree, boring, big group) is a variable to predict, and such a variable can take two values: yes or no. In addition to tree-structured CRFs, they also assign a group of labels rather than a single label to each node of the CRF. The model is then no longer a tree-structured CRF, but a more general CRF with bounded tree-width.

A bayesian framework for learning CRF has also been developed in [QSM05, WP06, PW06]. They use an approximation of the posterior probability distribution of the parameters for a given structure through the use of power Expectation Propagation [QSM05] or the Laplace approximation [WP06] for $\log Z(\boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{C}}, \mathbf{w})$. While they both derive an analytic expression for the posterior probability of a configuration $\mathbf{c}$, exploiting it for performing MAP inference is not possible. Thus they respectively develop an iterative process to obtain an estimate of $\mathbf{c}$, or use this approximated distribution to sample a set of models and average their predictions. In addition, the Laplace approximation result was exploited to derive a bayesian score for a given structure [PW06].

8.4 Mixtures of Tree-structured CRFs: Proposed Meta-algorithm

This section is a first step towards developing perturb and combine learning algorithms for mixtures of tree-structured CRFs. A meta-algorithm for learning mixtures of CRFs for variance reduction is proposed. It uses as a building block the algorithm proposed in [BG10] for learning a tree-structured CRF.

A mixture of tree-structured CRFs encodes a conditional probability density of the form

$$\mathbb{P}_{\boldsymbol{\mathcal{T}}}(\boldsymbol{\mathcal{C}}|\boldsymbol{\mathcal{X}}) = \sum_{j=1}^{m} \lambda_j \mathbb{P}_{\mathcal{T}_j}(\boldsymbol{\mathcal{C}}|\boldsymbol{\mathcal{X}}) \ , \tag{8.15}$$

where $\mathbb{P}_{\mathcal{T}_j}$ is the conditional probability density encoded by the j^{th} tree-structured CRF. As for mixtures of Markov trees, $\mathbb{P}_{\boldsymbol{\mathcal{T}}}(\boldsymbol{\mathcal{C}}|\boldsymbol{\mathcal{X}})$ is a well-defined probability distribution if all weights λ_j are positive and sum to 1. The class variables can be predicted for a given input $\mathbf{x}$ by $\arg\max_{\boldsymbol{\mathcal{C}}} \mathbb{P}_{\boldsymbol{\mathcal{T}}}(\boldsymbol{\mathcal{C}}|\boldsymbol{\mathcal{X}} = \mathbf{x})$.

Performing this MAP inference operation on a mixture is however not easy [POB09]. Computing $\mathbb{P}_{\boldsymbol{\mathcal{T}}}(\boldsymbol{\mathcal{C}} = \mathbf{c}|\mathbf{x})$ is easy for a particular configuration $\mathbf{c}$ of the class variables, but the configuration maximizing this average density is the result of the interaction of the densities encoded by the different terms in the model, and cannot be computed by averaging the distributions. In particular, the MAP configuration may not maximize any term of the mixture.

The problem of aggregating the terms of the mixture is discussed in Section 8.4.2. Section 8.4.1 proposes a meta algorithm for learning mixtures of CRFs, based on the perturb and combine principle.

8.4.1 Learning

The meta-algorithm used in Chapter 6 to learn mixtures of Markov trees can be modified to generate a mixture of tree-structured CRFs. The resulting algorithm (Algorithm 8.1) differs from the original on two accounts. First, the set of variables is partitioned into two sets, the class and the input variables. Second, learning the graphical part of the model is divided into two steps, one for the features and one for the structure.

Algorithm 8.1 Learning a mixture of tree-structured CRFs

```
Input: $\boldsymbol{\mathcal{C}}$; $\boldsymbol{\mathcal{X}}$; $D$; $m$
$\boldsymbol{\mathcal{T}} = \emptyset$; $\boldsymbol{\lambda} = \emptyset$; $\boldsymbol{w} = \emptyset$
for $j = 1 \to m$ do
  $F[j]$ =SamplePairwiseFeatures($\boldsymbol{\mathcal{C}}$,$\boldsymbol{\mathcal{X}}$,$D$)
  $\boldsymbol{\mathcal{T}}[j]$ =SampleTreeStructure($\boldsymbol{\mathcal{C}}$,$\boldsymbol{\mathcal{X}}$,$F[j]$,$D$ [,$\boldsymbol{\mathcal{T}}$])
  $\boldsymbol{w}[j]$=LearnParameters($\boldsymbol{\mathcal{T}}[j]$,$D$)
  $\boldsymbol{\lambda}[j]$ =LearnWeight($\boldsymbol{\mathcal{T}}[j]$,$D$)
end for
Normalize($\boldsymbol{\lambda}$)
return ($\boldsymbol{\mathcal{T}}$,$\boldsymbol{w}$,$\boldsymbol{\lambda}$)
```

Randomization can be introduced into each of the four steps of the algorithms: SamplePairwiseFeatures, SampleTreeStructure, LearnParameters and LearnWeight.

- SamplePairwiseFeatures is the procedure responsible for generating a set of candidate features to be used inside the factors. As it essentially consists in generating good candidate features to predict one or several class variables, randomization introduced in this step can be inspired by the approaches employed in feature selection and classification. For example, one set of features considered by [MVC12] is the output of a set of $|\boldsymbol{\mathcal{C}}|$ SVM classifiers (each classifier is trained to predict the output of a different variable $\mathcal{C}_i$). For each candidate pairwise factor ϕ_i defined on $\mathcal{C}_{i_1}, \mathcal{C}_{i_2}$, one possible approach may be to construct features on the smallest subset of input variables whose mutual information to $\mathcal{C}_{i_1}, \mathcal{C}_{i_2}$ is the highest:

$$\boldsymbol{\mathcal{X}}_{\boldsymbol{\mathcal{S}}_i} = \arg \min_{|\boldsymbol{\mathcal{X}}'_{\boldsymbol{\mathcal{S}}_i}|} \max_{\boldsymbol{\mathcal{X}}'_{\boldsymbol{\mathcal{S}}_i} \in \boldsymbol{\mathcal{X}}} I(\mathcal{C}_{i_1}, \mathcal{C}_{i_2}; \boldsymbol{\mathcal{X}}'_{\boldsymbol{\mathcal{S}}_i}) \ . \tag{8.16}$$

- SampleTreeStructure is the subroutine that is probably the most interesting to randomize. It consists in selecting the edges between class variables to be included in the model, so as to improve the estimation of the probability density defined on the classes. It is thus closely related to the Chow-Liu algorithm and to the methods developed in this thesis, although the mutual information is replaced by the conditional mutual information or another related quantity (see Section 8.2.2). The different randomization schemes introduced in this thesis could be adapted to CRFs. However, new possibilities for randomization can also be considered. For example, it may be interesting to randomize the input variables conditioning each estimate of the conditional mutual information. A CRF could be constructed by conditioning all conditional mutual information estimates on the same set of randomly selected input variables, or by using a different set of randomly selected input variables for each mutual information. Exploring possible coupling between SampleTreeStructure and SampleFeatures might also be interesting.

- LearnParameters can also be randomized. A randomization similar to the one tested in this thesis, i.e. using a bagged replicate of the learning set, can be used. However, as opposed to mixtures of Markov trees, this step is not trivial for tree-structured CRF and is performed by gradient ascent. Therefore, some extra-randomization might be considered to reduce the computational complexity of this step.

- LearnWeight computes a weight associated to a given tree-structured CRF. As for Markov trees, uniform or Bayesian weights can be considered. I suspect uniform weights will work better if the structures are generated by methods similar to those developed in this thesis.

8.4.2 Aggregation

To predict the values of the class variables for a new set of observations, several strategies are possible to avoid a brute force approach. For problems with many class variables and/or a large cardinality, considering all solutions may be intractable.

In the context of uniformly weighted tree-structured CRFs, [POB09] proposed to perform a majority voting, for each variable, based on the MAP configuration $\mathbf{c}^{MAP,j}$ of each term j of the mixture. Each class variable $\mathcal{C}_k$ is assigned the value (adapted for a weighted mixture)

$$c_k = \arg\max_{c_{ki}} \sum_{j=1}^{m} \lambda_j \delta(c_k^{MAP,j} = c_{ki}) \tag{8.17}$$

$$\mathbf{c}^{MAP,j} = \arg\max_{\mathbf{c}} \mathbb{P}_{\mathcal{T}_j}(\mathbf{c}|\mathbf{x}) \ , \tag{8.18}$$

where $\delta(c_k^{MAP,j} = c_{ki}) = 1$ if $\mathcal{C}_k$ takes the value c_{ki} in the MAP configuration $c_k^{MAP,j}$ for the j^{th} CRF of the mixture.

They also proposed to perform a majority vote based on the marginals, without expliciting it, and report the accuracy was similar. I assume this voting strategy corresponds to

$$c_k = \arg\max_{c_{ki}} \sum_{j=1}^{m} \lambda_j \mathbb{P}_{\mathcal{T}_j}(\mathcal{C}_k = c_{ki}|\mathbf{x}) \ . \tag{8.19}$$

I suspect both strategies can lead to forbidden configurations. Indeed, each strategy selects the value of each variable without considering the values actually selected for other class variables. To avoid such a case, I suggest to perform a majority over all configurations of the class variables that are the MAP configuration for at least a tree of the mixture:

$$\mathbf{c} = \arg \max_{\mathbf{c} \in \{\mathbf{c}^{MAP,j}\}} \sum_{j=1}^{m} \lambda_j \delta(\mathbf{c}^{MAP,j} = \mathbf{c}) \tag{8.20}$$

or

$$\mathbf{c} = \arg \max_{\mathbf{c} \in \{\mathbf{c}^{MAP,j}\}} \sum_{j=1}^{m} \lambda_j \mathbb{P}_{\mathcal{T}_j}(C = \mathbf{c}|\mathbf{x}) \ . \tag{8.21}$$

Considering a small subset of candidate solutions is easy. The set of candidate solutions $\{\mathbf{c}^{MAP,j}\}$ can therefore be extended by the configurations computed by the voting strategies proposed by [POB09]. It may also be possible to test all solutions within a given Hamming distance[1] of any solution in $\{\mathbf{c}^{MAP,j}\}$, or to use other heuristics to search for additional solutions.

[1]The Hamming distance between two configurations is the number of variables whose values differ in the two configurations.

8.5 Conclusion

This chapter has investigated the possibility to construct mixtures of conditional random fields for multivariate classification. Conditional random fields have first been defined and characterized. Then, learning these models have been discussed. Next, existing approaches to learning and inference with conditional random fields, based on mixture models, have been described. In particular, a few algorithms relying and mixtures of tree-structured conditional random fields have been examined. Finally, the application of the perturb and combine to conditional random fields, in order to construct a variance reducing mixture of tree-structured conditional random fields, has been investigated.

A meta-algorithm for learning these mixtures has been constructed, by adapting the meta-algorithm for learning mixtures of Markov trees. The different subroutine of the new meta-algorithm have been discussed, in particular with respect to the different randomization schemes possible. Most schemes developed in this thesis seem to be transposable to this new meta-algorithm. Several new randomization schemes have also been proposed, based on the elements specific to the class of conditional random fields.

In addition, a major difference between mixtures of CRFs and mixtures of Markov trees has been identified. The aggregation mechanism, trivial when inferring a marginal probability in a mixture of Markov trees, deserves much more attention when performing MAP inference on a mixture of CRFs. While inferring the MAP configuration of each term is easy, computing the MAP configuration of the mixture is not. Therefore, new aggregation strategies have been proposed.

Chapter 9

Final Words

9.1 Overview

This thesis has explored new learning algorithms for mixtures of Markov trees, a model particularly interesting for density estimation in high dimensional problems. The methods developed in this thesis result from the application of the perturb and combine framework to probabilistic graphical models. They try to reduce the variance of a single Markov tree by randomizing the Chow-Liu algorithm, and by averaging several models produced by this randomized version.

The first step of this thesis was the analysis of existing Markov tree learning algorithms, and in particular of the Chow-Liu algorithm. Regularization, another method to reduce the variance, was discussed in the context of Markov trees, and a reference method was derived from this discussion. The Chow-Liu algorithm was also randomized, both to speed up its execution and to make the result stochastic. The only way to reduce its $\mathcal{O}(p^2 \log p)$ computational complexity is to subsample the number of edges considered. The experimental evaluation of the two algorithms developed (random subsampling or cluster-based subsampling) showed that sampling good edges was important to achieve accuracy.

In the second step, mixtures of Markov trees were constructed. Two classes of methods were considered. Algorithms of the first class construct a mixture by repeatedly applying a given Markov tree learning procedure. Therefore, the trees of the mixtures are independent from each other, conditionally on the learning set and on the Markov tree learning algorithm used.

Algorithms of the second class construct a mixture sequentially: each new Markov tree is related to the structures of the previous trees in the sequence. The Markov trees are therefore not independent anymore conditionally on the learning set.

The first category of mixtures consisted of structures constructed by a repeated application of the algorithms studied in the first part of the thesis, either on the learning set or on a bootstrap replicate. Bagged mixtures of Markov trees had already been shown to achieve a better accuracy than a Chow-Liu tree. However, I tried a mixed method where only the structure, as opposed to the parameters, are learned on a bootstrap replicate. This new mixed variant, coupled to the Chow-Liu algorithm, proved superior to any other method within the considered panel of methods. In addition, the effect of the number of samples on these mixtures was studied: the fewer samples, the stronger the improvement in terms of accuracy with respect to a Chow-Liu tree. Finally, and as for single Markov trees, the more and the stronger the edges considered by the algorithm, the better the mixture tends to be. Since the randomized algorithms for tree structure learning developed in the first step allow some control over the overall number of edges considered by an algorithm without sacrificing too much strength, these methods naturally offer an accuracy/complexity trade-off.

Because the strength of the edges considered influences the quality of the mixture and because the number of edges considered influences the run time of the learning algorithm, the second category tries to prevent the structure learning algorithms to waste time on weak edges once they have been identified. This requires sharing information between the different successive applications of the base learning algorithms. Therefore, the structures are constructed in a sequence, and each structure depends on the computation performed to construct the previous structures. Two algorithms using a different mechanism to transfer information were developed. They both consider all possibles edges for the first tree. The first mechanism, already used by an existing method, considers for each tree the edges of the previous structure, to which it adds a random subset of edges. The second mechanism selects a subset of edges by an independence test performed on the edges during the computation of the first tree, and considers only this subset of edges for any subsequent tree. This sharing of information yielded significant improvements. The second mechanism in particular was able to achieve an accuracy similar to the mixture of (mixed) bagged Markov trees with a learning time reduced by one order of magnitude. These methods were also compared to and found to be most of the time more accurate than the regularization reference, thus confirming the interest of the ensemble models.

The third step of the thesis was the combination of mixtures reducing the bias and mixtures reducing the variance. Replacing each term of an expection-maximization [EM] mixture (reducing the bias rather than the variance) by a bagged mixture of Markov trees significantly improves the accuracy by reducing the variance of each term. Moreover, the improvement in accuracy was observed to increase with the number of trees in the first level. Since the optimal value of this number tends to rise with the number of samples, marrying the two types of mixtures may improve the EM mixture for any sample size, thus expanding the range of problems where the variance reducing mixtures are useful.

Finally, another domain where the techniques designed in this thesis could be applied was investigated: multi-target classification with conditional random fields. This research domain lies in-between supervised learning, where perturb and combine was rather successful, and density estimation with probabilistic graphical models, the focus of this thesis. A review of the state-of-the-art showed that learning tree-structured conditional random fields is an active research topic, and that mixtures of tree-structured conditional random fields for variance reduction have been used recently, but not investigated in details. Based on available results, a meta-algorithm was proposed to apply the ideas developed in this thesis to these models.

9.2 Short Term Perspectives

Some additional experiments could be performed to refine the analysis of the contributions of this thesis.

First, it would be interesting to compare the mixtures of Markov trees constructed by perturbing and combining the Chow-Liu algorithm with complete Bayesian posterior averaging of Markov trees. Computing this posterior distribution has a cubic complexity in the number of variables, and it does not permit inference directly. Quantifying the accuracy of this distribution with respect to the target distribution would however be possible, and would constitute another interesting reference method, along with the Chow-Liu tree and the regularized forest. This distribution could also be used to sample Markov trees, and to generate a mixture where inference would be possible.

In addition, one could investigate why and on which type of problems the mixtures reduce the variance better than a regularized model or the Chow-Liu tree, and when the sequential mixtures are able to approximate bagging well. Understanding what determines the quality of a mixture could lead to the development of other learning algorithms, or to procedures for automatically selecting the parameters such as the number of edges considered by each tree learning algorithm. Potential informative quantities are the distribution over the score of the trees generated, over the edges they use (i.e. structure diversity), or over the mutual information values of these edges.

On the other hand, the 2-level mixture could be compared to other existing approaches designed to reduce the variance of an EM mixture, such as a Markov chain Monte Carlo over the partitions of the learning set. New methods can also be created by applying the perturb and combine principle directly at the mixture level, rather than on each term of a mixture. This could lead to more important reduction of the variance, but I suspect that optimizing each mixture by the EM algorithm will be cumbersome from the computational point of view.

Finally, the proposed mixtures of conditional random fields should be further developed and tested on both synthetic and realistic datasets. In particular, different aggregation strategies must be developed. Because there is a possibly large set of variables to predict rather than a single one, it might not possible to directly adapt aggregation strategies used in classical ensemble methods for classification. For example, the different models in the ensemble might each output a different MAP configuration.

9.3 Long Term Perspectives

The results of this thesis also generate new avenues of research, listed in this section.

Other target distributions could be considered. In all the experiments conducted, any target probability density generating the learning sets belongs to the class of Bayesian networks. It would be interesting to evaluate the accuracy of mixtures of Markov trees on learning sets generated by target distributions belonging to the class of Markov random fields. Because the independence relationships in a Markov random field can be encoded in the structure of a (non-trivial) ensemble of Markov trees, those mixture models might work better with Markov random fields than with Bayesian networks.

Another interesting question is whether this approach may be applied to other types of variables, continuous and normally distributed variables, or other types of numerical variables.

I believe some of the ideas developed in this thesis could be applied to normal distributions. A mixture of normal probability densities could be build in order to reduce the variance with respect to a single model. It would be worth investigating whether learning a mixture of normal probability densities, and then aggregating them, leads to a more accurate model than estimating a single structure directly by averaging approaches. Indeed, a few randomized procedures have already been developed to learn Gaussian graphical models, e.g. [CR06, SS05]. Several other approaches have also been developed for this task, in particular based on l_1 norm regularization [FHT08]. So the competition against existing methods is likely to be stiffer than for Markov trees over discrete variables. In addition, perturb and combine methods have also been used to improve the accuracy of the result of l_1 norm regularization [Bac08, MB10, WNRZ11]. The ideas developed in this thesis to accelerate perturb and combine algorithms could also be applied to improving these approaches.

The methods developed in this thesis can be directly used to learn a probability density over a set of numerical variables, by discretizing these variables and considering them as categorical variables. It would however also be interesting to see whether specific graphical models exist for numerical variables, and how they are related to Markov trees.

Other models than Markov trees may also be considered to build a mixture. The class of Markov trees was selected because inference is linear in the number of variables for this class of models. For some problems with an intermediate number of variables, it may however be acceptable to perform inference on models of bounded tree-width. The larger the bound, the larger the set of densities the models can encode. The reasoning behind the different learning algorithms developed in this thesis can be applied to models of bounded tree-width, allowing more freedom to adapt the modelling capacity of the output mixture.

The relationships between regularization and independence test based methods should be explored further. Different points can be investigated.

- Various algorithms use one or more parameters linked to a threshold on mutual information values. These parameters are for now not selected based on a learning set. Automatic procedures could therefore be developed to select the values of the parameters of several algorithms, based on the learning set.

  - This would *increase the accuracy of the algorithms developed.* Several algorithms for learning a single Markov tree or a mixture of Markov trees have one or more parameters corresponding to the risk of type I error in an independence test. This makes the selection of the parameters easier than a threshold on mutual information and allows a seamless adaptation to the number of variables, but my experiments suggest that optimizing these parameters on a learning set can still lead to an increase in accuracy.
  - This would *decrease the accuracy of the regularization reference.* The reference algorithm developed and considered in this thesis provides an upper bound on the accuracy achievable by a regularization of the Chow-Liu algorithm by a fixed threshold. Comparing the accuracy of mixtures to a regularized tree whose threshold (or number of edges) is optimized based on the learning set alone (and not on the sample size) may provide a more realistic picture of the respective interest of regularization and of the ensemble methods.

- Regularization could also be considered for each term of the mixture, even though both reduce the variance. Indeed, in some of my experiments, bagging the Chow-Liu algorithm led to a worse accuracy than the original algorithm while a regularization of this original algorithm resulted in a better accuracy. The two methods therefore do not seem completely exchangeable. Rather they are, at least slightly, complementary. However, in the methods developed in this thesis, restraining the number of edges considered has been used to reduce algorithm complexity rather than to reduce the variance. Jointly regularizing and randomizing the Chow-Liu algorithm may combine the variance reduction capacity of both methods, and lead to learning algorithms more accurate than any of these methods taken alone.

- Making the parameters of the algorithms, in particular those assimilated to a risk of type 1 error, evolve with the size of the mixture might be another direction for improvement. Those parameters constrain each model inside the mixture. Ideally, the first model of the mixture should be regularized to maximize the accuracy of this single model. In this case, the mixture of size 1 is optimal with respect to any other regularization used on a mixture of the same size and constructed with the same learning algorithm. However, the experiments performed showed that fewer constraints on the terms of the mixture may lead to a better accuracy for larger mixture. Therefore it is tempting to adapt the level of constraint to the number of terms in the mixture, so that the mixture is optimal for any size, with respect to any level of constraint.

9.4 Application Perspectives

In the long term, the methods developed in this thesis to estimate a probability density based on a set of observations can have potential applications in electrical engineering or bioinformatics.

In electricity, an interesting application would be the modelling of the probability distribution of the consumption at the different nodes of a power distribution network. Depending on external factors, the network can be in different modes of consumption. A model of the probability distribution could be used e.g. to extract those consumption modes from a learning set, or to identify a small number of nodes that can be monitored to determine the current mode of the system.

A model could also be constructed to encode the probability distribution of the evolution of those consumptions between two points in time. Based on this model, it could be possible to construct a Markov chain for longer term time prediction, e.g. forecasting the consumption at different points in time, based on current and/or past measures of the consumption.

To model time series, it may also be interesting to see whether mixtures could be combined with dynamic Bayesian networks. A dynamic Bayesian network is a Bayesian network that encodes a probability distribution over a sequence of variables. Could a mixture of tree-structured dynamic Bayesian networks be constructed? Where should the edges be allocated: to model the time transition, or to model the interactions between the variables of a given time point?

In bioinformatics, it would be interesting to learn a probability density over gene expression levels or other similar biological quantities related to the functioning of a cell. However biological networks usually have feedback loops, and Markov trees are likely to have trouble in modelling such loopy dependencies. This may in turn lead to a poor estimation of the probability density. However, most of those loops involve at least one gene coding for a transcription factor, i.e. a gene regulating the expression level of other genes. Therefore, an approach to modelling gene expression level might be to decompose the learning problem into a cyclical and a non-cyclical component, and later on to combine them to learn tree-structured models.

Appendix A

Experimental Problems

This appendix contains the description of the different target distributions used to generate the learning sets on which the algorithms presented in this thesis are evaluated.

All distributions are encoded by Bayesian networks, because these models are easier to manipulate. As opposed to other classes of PGMs such as Markov random fields, Bayesian networks don't require the computation of a partition function because they are naturally normalized (see Section 3.1 for details).

Generating an observation from a BN is carried out as follows. At first, the values of all variables are unknown. Let $\boldsymbol{\mathcal{S}}$ denote the set of all the variables $\mathcal{X}$ such that a value has already been assigned to all the parent variables $\boldsymbol{\mathcal{P}}a_{\mathcal{G}}^{\mathcal{X}}$ of $\mathcal{X}$ (initially this set is thus composed of the subset of variables which have no parents in the considered BN). Until $\boldsymbol{\mathcal{S}}$ is empty, take out one of its elements (the selection criterion doesn't matter), randomly generate a value for the corresponding variable using its conditional distribution with the already chosen values of its parent variables, and update $\boldsymbol{\mathcal{S}}$ by adding those variables not yet considered and for which the parent variables have all already been settled. Because the underlying graph is a directed acyclic graph, $\boldsymbol{\mathcal{S}}$ will remain non-empty until all variables have been assigned a value, and the procedure is guaranteed to terminate.

The experiments were carried out on three different types of target distributions. Randomly generated Bayesian networks are described in Section A.1 and randomly generated mixtures of Markov trees in Section A.2. Finally, a set of more realistic distributions, available from the literature, is presented in Section A.3.

A.1 Randomly Generated Bayesian Networks

A Bayesian network is composed of a graph and a set of parameters. In this section, both are randomly generated to specify a Bayesian network. Therefore, these distributions are sometimes denoted by the term "synthetic distributions" in the core of this manuscript.

The generation of the parameters is first described, followed by the description of tree-structured models and then by the generation of more general Bayesian networks.

A.1.1 Parameters

The parameters θ of any Bayesian network generated are divided into sets, one set

$$\theta_{i,\mathcal{X}_i|\mathbf{a}} = \left\{\theta_{i,x_i^j|\mathbf{a}}\right\}_{j=1}^{|\mathcal{X}_i|} \tag{A.1}$$

for each variable $\mathcal{X}_i$ and each value $\mathbf{a}$ of the parents $\boldsymbol{\mathcal{P}}a_{\mathcal{G}}^{\mathcal{X}_i}$ of $\mathcal{X}_i$ variables. Any set $\theta_{i,\mathcal{X}_i|\mathbf{a}}$ defines a (marginal) conditional probability density. The sum of the elements of this set must therefore equal to one.

The parameters of any Bayesian network generated are sampled from a product of uniform Dirichlet distributions, one for each variable and per configuration of the parent variables:

$$\mathbb{P}(\theta|\mathcal{G}) = \prod_{i=1}^{p} \prod_{\mathbf{a}\in Val(\boldsymbol{\mathcal{P}}a_{\mathcal{G}}^{\mathcal{X}_i})} Dirichlet(\frac{1}{|\mathcal{X}_i|}, \dots, \frac{1}{|\mathcal{X}_i|}) \ . \tag{A.2}$$

This generation process makes no assumption about a set of parameters, any configuration is equally likely. Moreover, each set of parameters is generated independently from the others. However, in a problematic situation, e.g.

$$\theta_{i,\mathcal{X}_i|\mathbf{a}} = \theta_{i,\mathcal{X}_i|\mathbf{b}} \qquad \forall \mathbf{a}, \mathbf{b} \in Val(\boldsymbol{\mathcal{P}}a_{\mathcal{G}}^{\mathcal{X}_i}) \ , \tag{A.3}$$

removing some or all of the edges incoming to $\mathcal{X}_i$ does not modify the probability distribution encoded by the Bayesian network, making learning the structure difficult. The probability to generate sets of parameters verifying Equation A.3 is null[1]. However, values of the parameters only slightly

[1]This is not true for the degenerate case where $|\mathcal{X}| = 1$, but a variable $\mathcal{X}$ whose value is certain can be removed from the network.

different can also be problematic for learning. This could be avoided by enforcing sufficient diversity in the parameter values.

A.1.2 Markov Trees

Structures of Markov trees were uniformly generated through the use of Prüfer lists. The procedure is described in Section 5.4.1.

A.1.3 DAG Structures

The constraint on the maximum number of parents was relaxed to generate other Bayesian networks, but it remained upper bounded. Without such a limit, the number of parameters would grow so much that storing them in memory would not be possible. Indeed, the number of parameters grows exponentially in the number of parents of a variable.

The structures of bounded in-degree k were generated by first arbitrarily ordering the p variables, and then by successively randomly drawing, for any i, first the number of parents of $\mathcal{X}_i$ in $[0, \min(i-1, k)]$, and then selecting at random these parents in $\{\mathcal{X}_1, \ldots, \mathcal{X}_{i-1}\}$ (without replacement). Considering only these variables obviously enforces the directed acyclicity constraint on the structure. Notice that the mean number of edges of such a structure converges to $pk/2$ as p tends to infinity.

These models are denoted by DAG-p-k. DAG-200-10 corresponds to a network of 200 variables, where the number of parents was bounded to 10.

A.2 Randomly generated mixtures of Markov trees

The weights and the terms of the mixtures were generated independently. In addition, each term was generated independently from the others.

The weights of a mixture of m terms were always set to a constant value $1/m$. This ensures that no weight is equal or close to zero. The estimation of such a mixture is easier, by avoiding (near) degenerate cases. However, the information about the weights was not provided to the algorithms estimating the weights.

The different terms of the mixtures were generated by the method of Section A.1.2.

Name	p	$\|\mathcal{X}_i\|$	$\|E(\mathcal{G})\|$	$\|\theta\|$	reference
Alarm10	370	2-4	570	5468	[BSCC89]
Child10	200	2-6	257	2323	[SDLC93]
Gene	801	3-5	977	8348	[TBA06]
Hailfinder10	560	2-11	1017	97448	[ABE+96]
Insurance10	270	2-5	556	14613	[BKRK97]
Link	724	2-4	1125	14211	[JK99]
Lung Cancer	800	2-3	1476	8452	[AST+10]
Munin	189	1-21	282	15622	[AJA+89]
Pigs	441	3-3	592	3675	[Jen97]

Table A.1: Distributions from the literature and their characteristics. $|\theta|$ corresponds to the number of independent parameters.

A.3 Reference Distributions

The distributions described in the previous sections are generated according to procedures I designed myself (except the Markov trees). This may have unconsciously influenced the design of the algorithms presented here. Therefore I also tested these algorithms on reference distribution taken from the literature and regularly used to evaluate algorithms for learning probabilistic graphical models.

These distributions (and associated learning and test sets) were downloaded from the online supplement of [AST+10]. Table A.1 lists their main characteristics: name, number and range of cardinality of variables, number of edges and of independent parameters, and reference.

Any distribution whose name ends by a number is an extended version of a smaller, reference model. The original model was extended because it had too few variables. The extension process is called “tiling” and was proposed in [TSBA06]. It consists in copying the network several times (the number at the end of the name corresponds to the number of copies), and joining the copies together with additional edges.

Alarm was originally a network for patient monitoring in a hospital. Child was meant as a decision system for medical diagnosis, to be used by telephone operators answering calls for a 'blue baby' problem. Gene is a network constructed based on gene expression data in yeast. Hailfinder predicts extreme weather conditions in Northeastern Colorado. Insurance is a network for risk assessment in the context of car insurance. Link is a pedigree network in human, used to compute the distance between two genes. Lung Cancer is a network constructed based on human lung cancer gene expression data. Munin is an expert system for the analysis of electromyography (EMG) measurements. Pigs is a genetic linkage network over a population of pigs.

Appendix B

Terminology of Graph Theory

Probabilistic graphical models rely on graph theory. Therefore, many concepts used to qualify graphs are also applied to PGMs. The goal of this appendix is to centralize the definitions of a few terms of graph theory used in this thesis.

A **graph** is a pair (V, E). V denotes a set of elements called the **nodes** or **vertices**. E is a subset of elements of $V \times V$. The elements of E are called the **edges** of the graph, and are said to link or join the nodes. In the present thesis, the two vertices of an edge will always be different.

A graph is called directed (respectively undirected) when it contains only directed (undirected) edges. A **directed edge** is an edge (i, j) such that there is no edge (j, i) in the graph, with $i, j \in V$. When $(i, j) \in E$ and $(j, i) \in E$, the graph contains an **undirected edge** between i and j (or the opposite). In other words, the ordering of the two nodes is important for a directed edge but not for an undirected one. A directed edge therefore has an origin and a target, and is often represented by an arrow.

A **hypergraph** is a pair (V, E) where the elements of E are allowed to contain any number of vertices. These elements are called **hyperedges**.

A **bipartite graph** is a graph where the nodes can be partitioned into two disjoint sets V_1 and V_2 and where an edge can only link a node of V_1 and a node of V_2.

A **path** is a sequence of vertices $i_1, \ldots, i_k$ of the graph (V, E), such that

$(i_q, i_{(q+1)}) \in E \; \forall q \in [1, k-1]$. A directed edge links the source to the target, but not the other way around.

A **cycle** is a path whose first and last vertices are identical.

An **acyclic graph** is a graph without cycle.

A **chord** is an edge between two non-successive vertices of a loop.

A **chordal graph** is a graph where every loop of more than three vertices has at least one chord.

A **subgraph** from a graph (V, E), induced by a subset of vertices $V' \in V$, is a graph (V', E'), where E' is the subset of E containing all the edges of E that link only vertices in V'.

A **completely connected graph** is a graph where there is an edge between every pair of variable.

A **clique** of an undirected graph is a completely connected subgraph of this graph. The size of a clique is its number of vertices. A **maximal clique** of a graph is a clique such that there is no greater clique in the graph that contains this clique. The **tree-width** of a graph is the size of its maximal cliques, minus 1.

In a directed graph, the **parents** of a node V_i are the subset of all the nodes V_j such that E contains an edge from V_j to V_i, i.e. all the nodes from which there is a path of length 1 ending at V_i. The **ancestors** of V_i are the nodes from which there is a path of any length to V_i. Likewise, the **children** (respectively **descendants**) of a node V_i are all the nodes for which there exist a path of length 1 (of any length) to V_i.

The **moral graph** of a directed graph (V, E) is an undirected graph (V, E'), constructed from (V, E) by

- for every vertex V, marrying, i.e. linking by an undirected edge, the parents of this variable in the graph;
- transforming each directed edge in an undirected edge.

A **topological ordering** of the vertices is an ordering such that, for every pair of vertices V_i,V_j $(i \neq j)$, an edge $V_i \rightarrow V_j$ can be present in the graph if and only if V_i precedes V_j in the ordering.

Appendix C

Complexity Theory and Probabilistic Graphical Models

This appendix is a compilation of different computational complexity results from the literature. A brief introduction to complexity theory is first given, before these results are presented for learning and then for inference in Bayesian networks and other probabilistic graphical models.

C.1 Complexity Theory

The goal of complexity theory is to classify different types of problems (a question to be solved) according to the computational complexity (and memory requirements) of the best algorithms that can be used to solve each type of problems. This complexity is studied as a function of the input of the problem, because it can be reasonably expected that an algorithm will take longer to complete on a larger problem. Inside a given complexity class, the algorithms scale more or less (depending on the class) similarly with the input size of the problem.

The goal of the current section is to intuitively present some notions of complexity theory. This introduction will hopefully help a reader unfamiliar with complexity theory to understand the results listed later in the present appendix. A formal definition of these notions is outside the scope of this appendix.

Two important complexity classes are P and NP (Non-deterministic polynomial time). Those two classes contain only decision problems, i.e.

problems where the answer is either "yes" or "no". The class **P** contains all decision problems that can be solved (by a deterministic algorithm) in a time that is polynomial in the size n of the input. **NP** contains all decision problems for which a solution can be verified in a time polynomial in the input. The class NP contains the class P, and the two classes are believed to be different. Establishing whether or not they are indeed different is one important question in computer science.

As an example, the subset-sum problem is a decision problem that belongs to NP but is (thought to be) not in P. The problem asks whether, in a set of integers (the input), a non-empty subset of them sum to zero. The associated function problem is to find such a non-empty subset. Verifying a solution to this function problem is easy (summing the subset and checking whether the result is indeed zero), but finding a solution is difficult and is (believed to be) impossible in a time that can be expressed as a polynomial function of the number of input integers. Solving the decision problem, i.e. determining whether the set of solutions of the associated function problem is empty or not, is also (believed to be) difficult.

The class NP contains easy or tractable problems (i.e. problems in P, that can be solved by an algorithmic of polynomial complexity in the input), but also problems that are thought to be untractable (i.e. problems that (if indeed P $\neq$ NP) would be unsolvable by an algorithm of polynomial complexity). To distinguish between tractable and (potentially) harder problems, the hardest problems in NP are said to be **NP-complete**. To establish that a given problem is NP-complete, it is sufficient to show that a (known) NP-complete problem can be transformed into the given problem by a polynomial algorithm. Expressed differently, it is sufficient to show that, if an algorithm can solve the given problem in polynomial time, than it can also solve a NP-complete problem (and any other problem in the class) in polynomial time.

There are problems that are even more complicated than NP-complete. The class **NP-hard** contains all problems that are at least as hard as NP-complete problems, because NP-hard problems can be used to solve NP-complete problems. NP-hard problems do not have to be decision problems. The class NP-complete is the intersection of NP and NP-hard.

#P contains problems that output a numerical value. #P problems correspond to a "counting" version of a decision problem that is in NP: rather than asking whether there is a solution, the #P problems asks how many solutions there are. As an illustration, the counting version of the subset-sum problem is "how many different non-null subsets of a given set of integers sum to zero?" Such a problem is at least as hard as the corresponding NP problem. An algorithm solving this counting problem can be used to solve the associated decision problem: if the number of solutions to the function problem is known, it is trivial to know whether there is at least one solution.

The two classes RP and PP are slightly different from the other classes considered so far. These classes classify decision problems based on the complexity of algorithms that are allowed to make random guesses. **RP** contains problems for which there exist a polynomial time algorithm that

- always outputs "no" where the true answer to the problem is "no"
- and that has at probability of 0.5 or greater to output "yes" when the true answer is "yes".

PP contains problems for which there exist a polynomial time algorithm that

- outputs "no" with a probability equal to or greater than 0.5 if the true answer is "no",
- and that output "yes" with a probability greater than 0.5 if the true answer is "yes".

In addition, complexity theory also construct complexity classes based on the notion of *oracle*. An oracle is a subroutine that is considered capable to answer a class of problems with a cost of 0. Calling the subroutine has a cost of 1. The notion of oracle allows the study of different sources of complexity in a given problem. For example, the class P^{PP} is a class of decision problems that can be solved by a polynomial time algorithm that have access to an oracle for problems in PP.

C.2 Complexity of Learning in Probabilistic Graphical Models

The number of possible DAGs grows superexponentially with the number of variables p [Rob77]. Learning the structure of such a model is therefore a complex task, even when imposing strong restrictions on the structure. This section contains various results regarding the complexity of learning probabilistic graphical models in general and Bayesian networks in particular.

Markov trees are one of the classes of models for which structure learning is easy. Indeed, it can be performed by the algorithm proposed by Chow and Liu [CL68], described in Section 3.3.4 and of complexity $\mathcal{O}(p^2 \log p)$.

Another positive result is that the structure of a Bayesian network can be recovered by a polynomial number of calls to an independence oracle (i.e. using an infinite amount of observations), provided the number of parents of any variable is bounded by a known constant. The SGS algorithm described in Section 3.3.5 can be used to do so.

For the structure of most other classes of probabilistic graphical models, the complexity of learning is much less scalable, and usually belongs to the NP-hard class of problems.

[Bou95] proved that exploiting an independence oracle to construct a Bayesian Network is NP-hard for binary variables.

[CGH94, Chi96] demonstrated that learning a Bayesian network structure from a finite learning set of observations based on a Bayesian score, such as BDe or BD, is NP-hard when each node is constrained to have at most 2 parents. This result also holds if the limit on the number of parents is higher.

[CHM04] extended the two previous results. They proved the conjecture of [Bou95] that exploiting an independence oracle to construct a Bayesian Network is NP-hard for variables of cardinality higher than 2. They also consider learning a structure based on a consistant score computed from an infinite (or very large) number of observations, and show this problem is NP-hard for structures in which each node has at most 3 parents. Again,this result also holds for a higher limit on the number of parents.

Structures close to trees are also difficult to learn: finding the optimal polytree [Das99], unless the target distribution factorises according to a polytree model, finding the optimal path (or chain model) [Mee01], or finding the best component-restricted forest (a forest where the number of nodes in each connected component is bounded) [LXHG$^+$11] are all NP-hard problems.

Hardness results have also been uncovered for other PGM. [KS01, Sre03] considered the problem of finding the maximum likelihood Markov random field structure of bounded tree-width t with respect to a set of observations, and found it to be NP-hard as well for $t \geq 2$ ($t = 1$ corresponds to the case of Markov trees).

Some Probably Approximately Correct learning (PAC learning [Val84]) procedures have also been developed for learning Markov Random fields. These algorithms do not always provide the best structure, but are guaranteed to construct a structure close to the optimal most of the time. [NB04] developed a polynomial time algorithm in p for PAC learning bounded tree-widths MRF structures. [AKN06] proposed an algorithm of similar complexity for PAC learning MRF structures of bounded connectivity (i.e. the number of neighboring variables of each factor is bounded). [CG07] designed another algorithm for PAC learning bounded tree-widths MRF structures in polynomial complexity in p. All three algorithms are exponential in the bound. Finally, [EG08] devised a PAC learning procedure for learning such structures in a time polynomial in p and k, the bound on the tree-width.

Because of these hardness results, learning algorithms usually rely on heuristics and/or limitations of the space of candidate structures to learn a model, by restricting the resolution of the search space [AW08] or by limiting its range (e.g. by constraining the number of candidate parents or the global structures searched [EG08]). In practice, learning of a probabilistic graphical model structure is considered not to be possible over a few thousands variables [Auv02, EG08].

C.3 Complexity of Inference in Probabilistic Graphical Models

This section reports theoretical results about the complexity of the inference operations defined in Section 3.2.1. These different queries belong to [widely believed] distinct complexity classes. The complexity on inference has also been investigated for approximate answers and/or under different constraints on the class of Bayesian networks. Complexity results are presented in the following order: unconstrained model, approximated inference, bounded probabilities and finally structural constraints (bounded tree-width or (poly)tree structures).

On general networks, computing $\mathbb{P}(\mathcal{X})$, the probability of a single variable, has been found to be NP-hard [Coo90] and later to be #P-complete [Rot96]. Note that computing $\mathbb{P}(\boldsymbol{\mathcal{X}} = \mathbf{x})$, the probability of an instantiation of all variables is however polynomial in the number of variables. [Shi94] proved that MEP inference is NP-hard. [PD04] studied the complexity of

MAP inference and demonstrated it to be NP^{PP}-complete.

Motivated by these early results for algorithmic complexity of inference, algorithms for approximated inference were developed. However, [DL93] showed that approximating probability inference is Np-hard as well: unless NP $\subseteq$ RP, there is no polynomial-time approximation algorithm for probabilistic inference, even with a probability of failure. [Rot96] achieves a similar result. [AH98] considered approximation algorithms for MEP, and found the problem to be NP-hard as well. In addition, they found out that computing the second (or l^{th}) best explanation given the best (the $(l-1)$ best) explanation(s) is NP-hard too. So is approximating MAP according to [PD04].

Some of these results however can be weakened when the probabilities of the network or the probabilities of interest are bounded away from 0 and 1, i.e. $1 > \mathbb{P}(\mathcal{X}_i = x_i|\boldsymbol{\mathcal{P}}a^{\mathcal{X}_i} = \mathbf{a}) > 0$. [DL97] provided a quasi-polynomial ($\mathcal{O}(2^{(\log n)^d})$, with d an integer depending on the longest path in the structure) time approximation of probabilistic inference. Computing MEP or MAP however remains respectively NP-hard [AHH00] and NP^{PP}-complete [PD04], even with bounded probabilities.

Constraining the structure of the underlying graph of the Bayesian network also simplifies the problem. Computing $\mathbb{P}(\mathcal{X})$ is feasible with polynomial-time algorithms on Markov trees [KP83], and algorithms such as the junction tree algorithm scales as $p\exp(k)$ for inference on Bayesian networks of bounded tree-width k, see e.g. [LS98]. Moreover, bounded tree-width is necessary to obtain tractable inference. For Markov Random Field (with pairwise potentials), [CSH08] showed that, if a series of hypothesis about complexity theory are true (see the reference for details), inference is superpolynomial in the tree-width k. [KBvdG10] derived a similar result for Bayesian networks: bounded tree-width is necessary for inference to be polynomial in n, based on a conjecture about complexity theory. MAP inference however remains NP-hard for polytrees (shown for polytrees with the maximum number of parents bounded by 2 and therefore of tree-width$=2$) [PD04].

Today, inference in PGMs remains an active research topic. Contests are regularly held to evaluate the best algorithms available. To the best of my knowledge, the last contest about this topic is the Probabilistic Inference Challenge (PIC2011)[1]. Many learning methods also specifically target low

[1]`http://www.cs.huji.ac.il/project/PASCAL/`

tree-width structures in order to limit inference complexity, see e.g. [BJ01, EG08, FNP99, SCG09].

Bibliography

[ABE$^+$96] B. Abramson, J. Brown, W. Edwards, A. Murphy, and R. L. Winkler. Hailfinder: A Bayesian system for forecasting severe weather. *International Journal of Forecasting*, 12(1):57 – 71, 1996. 250

[AdC95] S. Acid and L. de Campos. Approximations of causal networks by polytrees: an empirical study. In B. Bouchon-Meunier, R. Yager, and L. Zadeh, editors, *Advances in Intelligent Computing — IPMU '94*, volume 945, pages 149–158. Springer Berlin / Heidelberg, 1995. 113

[AH98] A. M. Abdelbar and S. M. Hedetniemi. Approximating MAPs for belief networks is NP-hard and other theorems. *Artificial Intelligence*, 102(1):21 – 38, 1998. 260

[AHH00] A. M. Abdelbar, S. T. Hedetniemi, and S. M. Hedetniemi. Complexity of approximating MAPs for belief networks with bounded probabilities. *Artificial Intelligence*, 124(2):283–288, 2000. 260

[AJA$^+$89] S. Andreassen, F. V. Jensen, S. K. Andersen, B. Falck, U. Kjærulff, M. Woldbye, A. R. Sørensen, A. Rosenfalck, and F. Jensen. MUNIN — an expert EMG assistant. In J. E. Desmedt, editor, *Computer-Aided Electromyography and Expert Systems*, chapter 21. Elsevier Science Publishers, Amsterdam, 1989. 250

[AKN06] P. Abbeel, D. Koller, and A. Y. Ng. Learning factor graphs in polynomial time and sample complexity. *The Journal of Machine Learning Research*, 7:1743–1788, 2006. 259

[ALDW08] S. Ammar, P. Leray, B. Defourny, and L. Wehenkel. High-dimensional probability density estimation with randomized ensembles of tree structured bayesian networks. In *Proceedings of the fourth European Workshop on Probabilistic Graphical Models (PGM08)*, pages 9–16, 2008. xxi, 17, 113, 115, 119, 155, 159, 163

[ALDW09a] S. Ammar, P. Leray, B. Defoumy, and L. Wehenkel. Pertubation et combinaison d'arbres de Markov pour l'estimation de densité. In *Proceedings of Conférence Francophone sur l'Apprentissage Automatique*, pages 65–79, 2009. 115

[ALDW09b] S. Ammar, P. Leray, B. Defourny, and L. Wehenkel. 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*, pages 156–167. Springer Berlin / Heidelberg, 2009. 17, 18, 115, 137, 158, 159, 163

[ALSW10] S. Ammar, P. Leray, F. Schnitzler, and L. Wehenkel. Sub-quadratic Markov tree mixture learning based on randomizations of the Chow-Liu algorithm. In *Proceedings of the Fifth European Workshop on Probabilistic Graphical Models (PGM10)*, pages 17–24, 2010. 18, 116, 163, 171

[ALW08] S. Ammar, P. Leray, and L. Wehenkel. Estimation de densité par ensembles aléatoires de poly-arbres. In *Proceedings of Journées Françaises des Réseaux Bayesiens*, 2008. 115

[ALW10a] S. Ammar, P. Leray, and L. Wehenkel. Mélanges sous-quadratiques d'arbres de Markov pour l'estimation de la densité de probabilité. In *Proceedings of 5èmes Journées Francophones sur les Réseaux Bayésiens*, 2010. 115

[ALW10b] S. Ammar, P. Leray, and L. Wehenkel. Sub-quadratic Markov tree mixture models for probability density estimation. In *Proceedings of the 19th International Conference on Computational Statistics (COMPSTAT)*, pages 673–680, 2010. xxi, 17, 18, 115, 163, 171, 172, 173

[Amm10] S. Ammar. *Modèles Graphiques Probabilistes pour l'Estimation de Densité en grande dimension : applications du principe Perturb & Combine pour les mélanges d'arbres.* PhD thesis, Université de Nantes, December 2010. xxi, 115, 119, 155, 159

[AST+10] C. F. Aliferis, A. Statnikov, I. Tsamardinos, S. Mani, and X. D. Koutsoukos. Local causal and Markov blanket induction for causal discovery and feature selection for classification part i: Algorithms and empirical evaluation. *The Journal of Machine Learning Research*, 11:171–234, 2010. 250

[Auv02] V. Auvray. On the construction of the inclusion boundary neighbourhood for Markov equivalent classes of Bayesian network structures. In *Proceedings of the 18th Conference on Uncertainty in Artificial Intelligence*, pages 26–35. Morgan Kaufmann, 2002. 6, 80, 259

[AW08] V. Auvray and L. Wehenkel. Learning inclusion-optimal chordal graphs. In *Proceedings of the 24th Conference on Uncertainty in Artificial Intelligence*, pages 18–25. AUAI Press, 2008. 80, 259

[Bac08] F. Bach. Bolasso: model consistent lasso estimation through the bootstrap. In *Proceedings of the 25th International Conference on Machine Learning*, pages 33–40. ACM, 2008. 243

[BG10] J. K. Bradley and C. Guestrin. Learning tree conditional random fields. In *Proceedings of the 27th International Conference on Machine Learning*, pages 127–134. Omnipress, 2010. 224, 227, 228, 229, 230, 232, 234

[BJ01] F. R. Bach and M. I. Jordan. Thin junction trees. In *Advances in Neural Information Processing Systems 14*, pages 569–576, 2001. 96, 113, 261

[BKRK97] J. Binder, D. Koller, S. Russell, and K. Kanazawa. Adaptive probabilistic networks with hidden variables. *Machine Learning*, 29(2):213–244, 1997. 250

[Bou95] R. Bouckaert. *Bayesian belief networks: from construction to inference.* PhD thesis, Utrecht university, 1995. 258

[Bre96] L. Breiman. Arcing classifiers. Technical report, Dept. of Statistics, University of California, 1996. 59, 60

[Bre99] L. Breiman. Using adaptive bagging to debias regressions. Technical report, Dept. of Statistics, University of California, 1999. 51, 202

[Bre01] L. Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001. 59, 62

[BSCC89] I. A. Beinlich, H. J. Suermondt, R. M. Chavez, and G. F. Cooper. The ALARM monitoring system: A case study with two probabilistic inference techniques for belief networks. In *Proceedings of the 2nd European Conference on Artificial Intelligence in Medicine*, pages 247–256. Springer-Verlag, Berlin, 1989. 250

[Bun91] W. Buntine. Theory refinement on Bayesian networks. In *Proceedings of the 7th Conference on Uncertainty in Artificial Intelligence*, pages 52–60. Morgan Kaufmann, 1991. 94

[Cay89] A. Cayley. A theorem on trees. *The Quarterly Journal of Pure and Applied Mathematics*, 23:376–378, 1889. 137, 160

[CG07] A. Chechetka and C. Guestrin. Efficient principled learning of thin junction trees. In *Advances in Neural Information Processing Systems 20*, pages 273–280, 2007. 113, 259

[CG10] A. Chechetka and C. Guestrin. Evidence-specific structures for rich tractable CRFs. In *Advances in Neural Information Processing Systems 23*, pages 352–360. 2010. 18, 228, 229, 233

[CGH94] D. Chickering, D. Geiger, and D. Heckerman. Learning Bayesian networks is NP-hard. *Microsoft Research*, pages 94–17, 1994. 258

[CH67] T. Cover and P. Hart. Nearest neighbor pattern classification. *Information Theory, IEEE Transactions on*, 13(1):21–27, 1967. 25

[CH92] G. F. Cooper and E. Herskovits. A Bayesian method for the induction of probabilistic networks from data. *Machine learning*, 9(4):309–347, 1992. 92, 114

[CH97] D. Chickering and D. Heckerman. Efficient approximations for the marginal likelihood of Bayesian networks with hidden variables. *Machine Learning*, 29:181–212, 1997. 114

[Cha00] B. Chazelle. A minimum spanning tree algorithm with inverse-Ackermann type complexity. *Journal of the ACM*, 47(6):1028–1047, 2000. 86, 127

[Chi96] D. Chickering. Learning Bayesian networks is NP-complete. *Learning from Data: Artificial Intelligence and Statistics V*, pages 121–130, 1996. 258

[CHM04] D. Chickering, D. Heckerman, and C. Meek. Large-sample learning of Bayesian networks is NP-hard. *The Journal of Machine Learning Research*, 5:1287–1330, 2004. 258

[CL68] C. Chow and C. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, 14:462–467, 1968. xxi, 17, 80, 83, 86, 125, 126, 127, 232, 258

[Coo90] G. Cooper. The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial Intelligence*, 42(2-3):393–405, 1990. 259

[CR06] R. Castelo and A. Roverato. A robust procedure for Gaussian graphical model search from microarray data with p larger than n. *The Journal of Machine Learning Research*, 7:2621–2650, 2006. 243

[CS96] P. Cheeseman and J. Stutz. Bayesian classification (autoclass): Theory and results. In *Advances in Knowledge Discovery and Data Mining*, pages 153–180. 1996. 110, 114

[CSH08] V. Chandrasekaran, N. Srebro, and P. Harsha. Complexity of inference in graphical models. In *Proceedings of the 24th Conference on Uncertainty in Artificial Intelligence*. AUAI Press, 2008. 260

[Dar09] A. Darwiche. *Modeling and Reasoning with Bayesian Networks*. Cambridge University Press, New York, NY, USA, 2009. 3, 67, 102, 228

[Das99] S. Dasgupta. Learning polytrees. In *Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence*, pages 134–141. Morgan Kaufmann, 1999. 112, 113, 258

[DBIV96] J. S. De Bonet, C. L. Isbell, and P. Viola. MIMIC: Finding optima by estimating probability densities. In *Advances in Neural Information Processing Systems 9*, 1996. 112

[DL93] P. Dagum and M. Luby. Approximating probabilistic inference in Bayesian belief networks is NP-hard. *Artificial intelligence*, 60(1):141–153, 1993. 260

[DL97] P. Dagum and M. Luby. An optimal approximation algorithm for Bayesian inference. *Artificial Intelligence*, 93(1-2):1 – 27, 1997. 260

[DL10] J. V. Dillon and G. Lebanon. Stochastic composite likelihood. *The Journal of Machine Learning Research*, 11:2597–2633, 2010. 228

[DLR77] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39:1–38, 1977. 53, 200

[DP97] P. Domingos and M. Pazzani. On the optimality of the simple Bayesian classifier under zero-one loss. *Machine learning*, 29(2):103–130, 1997. 110

[EG08] G. Elidan and S. Gould. Learning bounded treewidth Bayesian networks. *The Journal of Machine Learning Research*, 9:2699–2731, 2008. 6, 80, 96, 113, 259, 261

[EGWL05] D. Ernst, P. Geurts, L. Wehenkel, and L. Littman. Tree-based batch mode reinforcement learning. *The Journal of Machine Learning Research*, 6:503–556, 2005. 59

[Eli11] G. Elidan. Bagged structure learning of Bayesian network. In *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics*, volume 15, pages 251–259, 2011. 94

[FGW99] N. Friedman, M. Goldszmidt, and A. Wyner. Data analysis with Bayesian networks: A bootstrap approach. In *Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence*, pages 196–205. Morgan Kaufmann, 1999. 94

[FHT08] J. Friedman, T. Hastie, and R. Tibshirani. Sparse inverse covariance estimation with the graphical lasso. *Biostatistics*, 9(3):432–441, 2008. 243

[FK03] N. Friedman and D. Koller. Being Bayesian about network structure. A Bayesian approach to structure discovery in Bayesian networks. *Machine Learning*, 50(1):95–125, 2003. 94

[FNP99] N. Friedman, I. Nachman, and D. Peér. Learning Bayesian network structure from massive datasets: The "sparse candidate" algorithm. In *Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence*, pages 206–215. Morgan Kaufmann, 1999. 96, 113, 261

[Fre95] Y. Freund. Boosting a weak learning algorithm by majority. *Information and computation*, 121(2):256–285, 1995. 59

[FS95] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *Proceedings of the 2nd European Conference on Computational Learning Theory*, pages 23–37. Springer-Verlag, 1995. 115

[Gal07] F. Galton. Vox populi. *Nature*, 75:450–451, 1907. 57

[GEW06] P. Geurts, D. Ernst, and L. Wehenkel. Extremely randomized trees. *Machine Learning*, 63(1):3–42, 2006. 59

[GKL$^+$11] S. Gaspers, M. Koivisto, M. Liedloff, S. Ordyniak, and S. Szeider. Towards finding optimal polytrees. In *NIPS Workshop on Discrete Optimization in Machine Learning: Uncertainty, Generalization and Feedback*, 2011. 113

[Hal87] P. Hall. On Kullback-Leibler loss and density estimation. *The Annals of Statistics*, 15(4):1491–1519, 1987. 42

[Hes98] T. Heskes. Bias/variance decompositions for likelihood-based estimators. *Neural Computation*, 10(6):1425–1433, 1998. 43

[HL11] M. Hilbert and P. López. The world's technological capacity to store, communicate, and compute information. *science*, 332(6025):60–65, 2011. 5

[HTF01] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* Springer-Verlag, 2001. 37

[IF01a] S. Ioffe and D. Forsyth. Human tracking with mixtures of trees. In *Proceedings of the 8th IEEE International Conference on Computer Vision*, pages 690–695. IEEE, 2001. 116

[IF01b] S. Ioffe and D. Forsyth. Mixtures of trees for object recognition. In *Proceedings of the 2001 IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 180–185. IEEE, 2001. 116

[IM98] P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the 30th annual ACM symposium on Theory of computing*, pages 604–613, 1998. 139

[Jen97] C. S. Jensen. *Blocking Gibbs sampling for inference in large and complex Bayesian networks with applications in genetics.* PhD thesis, Aalborg University, 1997. 250

[JK99] C. S. Jensen and A. Kong. Blocking Gibbs sampling for linkage analysis in large pedigrees with many loops. *The American Journal of Human Genetics*, 65(3):885 – 901, 1999. 250

[KBvdG10] J. H. Kwisthout, H. L. Bodlaender, and L. van der Gaag. The necessity of bounded treewidth for efficient inference in Bayesian networks. In *Proceedings of the 19th European Conference on Artificial Intelligence*, pages 623–626, 2010. 260

[KF09] D. Koller and N. Friedman. *Probabilistic Graphical Models*. MIT Press, 2009. 3, 67, 90, 228

[KK06] J. Kollin and M. Koivisto. Bayesian learning with mixtures of trees. In *Machine Learning: ECML 2006*, pages 294–305. Springer-Verlag, 2006. 115, 117, 199, 200

[KK09] M. P. Kumar and D. Koller. Learning a small mixture of trees. In *Advances in Neural Information Processing Systems 22*, pages 1051–1059, 2009. 115, 116, 200

[KL51] S. Kullback and R. Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86, 1951. 31

[KMT96] P. Kontkanen, P. Myllymaki, and H. Tirri. Constructing Bayesian finite mixture models by the EM algorithm. Technical Report NC-TR-97-003, NeuroCOLT, 1996. 110

[KP83] J. Kim and J. Pearl. A computational model for causal and diagnostic reasoning in inference systems. In *Proceedings of the 8th International Joint Conference on Artificial Intelligence*, pages 190–193. Morgan Kaufmann, 1983. 96, 260

[Kru56] J. B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, 7(1):48–50, 1956. xxi, 86, 87, 127, 138

[KS01] D. Karger and N. Srebro. Learning Markov networks: Maximum bounded tree-width graphs. In *Proceedings of the 12th annual ACM-SIAM symposium on Discrete algorithms*, pages 392–401. Society for Industrial and Applied Mathematics, 2001. 113, 258

[KS04] M. Koivisto and K. Sood. Exact Bayesian structure discovery in Bayesian networks. *The Journal of Machine Learning Research*, 5:549–573, 2004. 94

[KS07] S. Kirshner and P. Smyth. Infinite mixtures of trees. In *Proceedings of the 24th International Conference on Machine Learning*, pages 417–423. ACM, 2007. 18, 115, 117, 199, 200

[KSAG05] A. Kraskov, H. Stögbauer, R. Andrzejak, and P. Grassberger. Hierarchical clustering using mutual information. *Europhysics Letters*, 70(2):278–284, 2005. 140

[Kuh55] H. Kuhn. The Hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955. 205

[LBBH98] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 25

[LD05] D. Lowd and P. Domingos. Naive Bayes models for probability estimation. In *Proceedings of the 22nd International Conference on Machine Learning*, pages 529–536. ACM, 2005. 110

[LMP01] J. D. Lafferty, A. McCallum, and F. C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the 18th International Conference on Machine Learning*, pages 282–289. Morgan Kaufmann, 2001. 224, 225, 228

[LS98] V. Lepar and P. Shenoy. A comparison of Lauritzen-Spiegelhalter, Hugin, and Shenoy-Shafer architectures for computing marginals of probability distributions. In *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence*, pages 328–337. Morgan Kaufmann, 1998. 96, 260

[LXHG$^+$11] H. Liu, M. Xu, A. G. Haijie Gu, J. Lafferty, and L. Wasserman. Forest density estimation. *The Journal of Machine Learning Research*, 12:907–951, 2011. 107, 127, 128, 129, 166, 258

[MB10] N. Meinshausen and P. Bühlmann. Stability selection. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 72(4):417–473, 2010. 243

[Mee01] C. Meek. Finding a path is harder than finding a tree. *Journal of Artificial Intelligence Research*, 15:383–389, 2001. 112, 258

[Mei99] M. Meila. An accelerated Chow and Liu algorithm: fitting tree distributions to high dimensional sparse data. In *Proceedings of the 16th International Conference on Machine Learning*, pages 249–57. Morgan Kaufmann, 1999. 17, 135

[MJ01] M. Meila and M. Jordan. Learning with mixtures of trees. *The Journal of Machine Learning Research*, 1:1–48, 2001. 115, 116, 199, 200

[MJ06] M. Meila and T. Jaakkola. Tractable Bayesian learning of tree belief networks. *Statistics and Computing*, 16:77–92, 2006. 117, 199

[MK08] G. McLachlan and T. Krishnan. *The EM algorithm and extensions*, volume 382. John Wiley and Sons, 2008. 53, 200

[MP99] M. Meila-Predoviciu. *Learning with mixtures of trees.* PhD thesis, MIT, 1999. 52, 115, 116

[MP00] G. McLachlan and D. Peel. *Finite mixture models*, volume 299. John Wiley and Sons, 2000. 52, 53, 56

[MR94] D. Madigan and A. Raftery. Model selection and accounting for model uncertainty in graphical models using Occam's window. *Journal of the American Statistical Association*, 89:1535–1546, 1994. 94, 114

[MVC12] T. Mensink, J. Verbeek, and G. Csurka. Tree-structured CRF models for interactive image labeling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2012. 228, 233, 235

[MYA95] D. Madigan, J. York, and D. Allard. Bayesian graphical models for discrete data. *International Statistical Review/Revue Internationale de Statistique*, 63:215–232, 1995. 94

[NB04] M. Narasimhan and J. Bilmes. PAC-learning bounded treewidth graphical models. In *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence*, pages 410–417. AUAI Press, 2004. 259

[NWL$^+$07] P. Naïm, P.-H. Wuillemin, P. Leray, O. Pourret, and A. Becker. *Réseaux bayésiens.* Algorithmes. Eyrolles, 2007. 3, 67, 90, 96

[ODW99] A. Onisko, M. Druzdzel, and H. Wasyluk. A Bayesian network model for diagnosis of liver disorders. In *Proceedings of* *the 11th Conference on Biocybernetics and Biomedical Engineering*, pages 842–846, 1999. 77

[OOM04] M. Ouerd, B. J. Oommen, and S. Matwin. A formal approach to using data distributions for building causal polytree structures. *Information Sciences*, 168:111–132, 2004. 113

[PD04] J. Park and A. Darwiche. Complexity results and approximation strategies for MAP explanations. *Journal of Artificial Intelligence Research*, 21(1):101–133, 2004. 259, 260

[Pea88] J. Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference.* Morgan Kaufmann, 1988. 3, 4, 67

[PM06] D. Pelleg and A. Moore. Dependency trees in sub-linear time and bounded memory. *The International Journal on Very Large Data Bases*, 15(3):250–262, 2006. xxi, 17, 135, 136

[POB09] P. Pletscher, C. S. Ong, and J. M. Buhmann. Spanning tree approximations for conditional random fields. In *Proceedings of the 12th International Conference on Artificial Intelligence and Statistics*, pages 408–415, 2009. 18, 233, 234, 236, 237

[Prü18] H. Prüfer. Neuer beweis eines satzes über permutationen. *Archiv der Mathematik und Physik*, 27:742–744, 1918. 137

[PT10] N. Payet and S. Todorovic. $(\mathrm{RF})^2$ – random forest random field. In *Advances in Neural Information Processing Systems 23*, pages 1885–1893, 2010. 233

[PW06] S. Parise and M. Welling. Bayesian model scoring in Markov random fields. In *Advances in Neural Information Processing Systems 19*, pages 1073–1080, 2006. 233, 234

[QSM05] Y. Qi, M. Szummer, and T. Minka. Bayesian conditional random fields. In *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics*, pages 269–276, 2005. 233, 234

[Qui89] A. Quiroz. Fast random generation of binary, t-ary and other types of trees. *Journal of classification*, 6(1):223–231, 1989. xxi, 137, 138

[Rob77] R. W. Robinson. Counting unlabeled acyclic digraphs. In *Combinatorial Mathematics V*, volume 622, pages 28–43. Springer Berlin / Heidelberg, 1977. 80, 257

[Rot96] D. Roth. On the hardness of approximate reasoning. *Artificial Intelligence*, 82(1-2):273–302, 1996. 259, 260

[RP87] G. Rebane and J. Pearl. The recovery of causal poly-trees from statistical data. In *Proceedings of the 3rd Conference on Uncertainty in Artificial Intelligence*, pages 175–182. Elsevier Science, 1987. 113

[RS02] S. Rosset and E. Segal. Boosting density estimation. In *Advances in Neural Information Processing Systems 15*, pages 641–648, 2002. 115

[SAL$^+$11] F. Schnitzler, S. Ammar, P. Leray, P. Geurts, and L. Wehenkel. Efficiently approximating Markov tree bagging for high-dimensional density estimation. In *Machine Learning and Knowledge Discovery in Databases, Part III*, pages 113–128. Springer Berlin / Heidelberg, 2011. xxi, 17, 18, 172, 174, 176, 177

[SAL$^+$12] F. Schnitzler, S. Ammar, P. Leray, P. Geurts, and L. Wehenkel. Approximation efficace de mélanges bootstrap d'arbres de Markov pour l'estimation de densité. In *Proceedings of Conférence Francophone sur l'Apprentissage Automatique*, 2012. 177

[SCG09] D. Shahaf, A. Chechetka, and C. Guestrin. Learning thin junction trees via graph cuts. *Proceedings of the 12th International Conference on Artificial Intelligence and Statistics*, pages 113–120, 2009. 96, 113, 224, 229, 230, 231, 261

[SDLC93] D. Spiegelhalter, A. Dawid, S. Lauritzen, and R. Cowell. Bayesian analysis in expert systems. *Statistical science*, 8(3):219–247, 1993. 250

[SG03] A. Strehl and J. Ghosh. Cluster ensembles - a knowledge reuse framework for combining multiple partitions. *The Journal of Machine Learning Research*, 3:583–617, 2003. 63

[SGS93] P. Spirtes, C. Glymour, and R. Scheines. *Causation, Prediction, and Search.* Springer-Verlag, 1993. xxi, 90, 91

[Shi94] S. E. Shimony. Finding MAPs for belief networks is NP-hard. *Artificial Intelligence*, 68(2):399–410, 1994. 259

[SLW10] F. Schnitzler, P. Leray, and L. Wehenkel. Towards subquadratic learning of probability density models in the form of mixtures of trees. In *Proceedings of the 18th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN)*, pages 219–224, 2010. xxi, 17, 18, 138, 144, 163

[SM05] C. Sutton and A. McCallum. Piecewise training of undirected models. In *Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence*, pages 568–575. AUAI Press, 2005. 228

[SM07a] C. Sutton and A. McCallum. An introduction to conditional random fields for relational learning. In *Introduction to Statistical Relational Learning.* MIT Press, 2007. 225

[SM07b] C. Sutton and A. McCallum. Piecewise pseudolikelihood for efficient training of conditional random fields. In *Proceedings of the 24th International Conference on Machine Learning*, pages 863–870. ACM, 2007. 228

[SMFR08] M. W. Schmidt, K. P. Murphy, G. Fung, and R. Rosales. Structure learning in random fields for heart motion abnormality detection. In *Proceedings of the 2008 IEEE Conference on Computer Vision and Pattern Recognition.* IEEE, 2008. 229

[SORS01] R. Santana, A. Ochoa-Rodriguez, and M. R. Soto. The mixture of trees factorized distribution algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 543–550. Morgan Kaufmann, 2001. 116

[Sre03] N. Srebro. Maximum likelihood bounded tree-width Markov networks. *Artificial intelligence*, 143(1):123–138, 2003. 113, 258

[SS05] J. Schäfer and K. Strimmer. An empirical Bayes approach to inferring large-scale gene association networks. *Bioinformatics*, 21(6):754–764, 2005. 243

[SW11] F. Schnitzler and L. Wehenkel. Two-level mixtures of Markov trees. In *PhD session, ECSQARU*, 2011. 18

[Tar83] R. E. Tarjan. *Data structures and network algorithms*. Society for Industrial and Applied Mathematics, 1983. 173

[TAW11] V. Y. Tan, A. Anandkumar, and A. S. Willsky. Learning high-dimensional Markov forest distributions: Analysis of error rates. *The Journal of Machine Learning Research*, 12:1617–1653, 2011. 107, 127, 128, 129

[TBA06] I. Tsamardinos, L. E. Brown, and C. F. Aliferis. The max-min hill-climbing Bayesian network structure learning algorithm. In *Machine Learning*, pages 31–78, 2006. 89, 250

[TJP04] A. Topchy, A. K. Jain, and W. Punch. A mixture model for clustering ensembles. In *Proceedings of the 4th SIAM International Conference on Data Mining*, pages 379–390, 2004. 63

[TMCH98] B. Thiesson, C. Meek, D. M. Chickering, and D. Heckerman. Learning mixtures of dag models. In *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence*, pages 504–513. Morgan Kaufmann, 1998. 114

[TSBA06] I. Tsamardinos, A. Statnikov, L. E. Brown, and C. F. Aliferis. Generating realistic large Bayesian networks by tiling. In *Proceedings of the 19th International FLAIRS Conference*, pages 592–597, 2006. 250

[Val84] L. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984. 259

[Web00] G. I. Webb. Multiboosting: A technique for combining boosting and wagging. *Machine Learning*, 40(2):159–196, 2000. 51

[Weh93] L. Wehenkel. Decision tree pruning using an additive information quality measure. In *Uncertainty in Intelligent Systems*, pages 397–411. Elsevier - North Holland, 1993. 129

[WNRZ11] S. Wang, B. Nan, S. Rosset, and J. Zhu. Random lasso. *The Annals of Applied Statistics*, 5(1):468–485, 2011. 243

[WP06] M. Welling and S. Parise. Bayesian random fields: the Bethe-Laplace approximation. In *Proceedings of the 22nd Conference on Uncertainty in Artificial Intelligence*, pages 512–519. AUAI Press, 2006. 233, 234

[WS97] D. H. Wolpert and J. Schmidhuber. On bias plus variance. *Neural Computation*, 9(6):1211–1243, 1997. 42

[ZLX10] J. Zhu, N. Lao, and E. Xing. Grafting-light: fast, incremental feature selection and structure learning of Markov random fields. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 303–312. ACM, 2010. 229