Aalborg Universitet

AALBORG UNIVERSITY
DENMARK

Learning with Hidden Variables

A Parameter Reusing Approach for Tree-Structured Bayesian Networks

Karciauskas, Gytis

Publication date:
2005

Document Version
Publisher's PDF, also known as Version of record

Link to publication from Aalborg University

Citation for published version (APA):
Karciauskas, G. (2005). *Learning with Hidden Variables: A Parameter Reusing Approach for Tree-Structured Bayesian Networks*. Department of Computer Science, Aalborg University.

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal -

Take down policy
If you believe that this document breaches copyright please contact us at vbn@aub.aau.dk providing details, and we will remove access to the work immediately and investigate your claim.

Downloaded from vbn.aau.dk on: April 18, 2024

Learning with Hidden Variables: A Parameter Reusing Approach for Tree-Structured Bayesian Networks

Gytis Karčiauskas
Department of Computer Science
Aalborg University

Ph.D. Thesis

2005

Læring af skjulte variable: Genbrug af parametre for træstrukturerede Bayesianske net

(title in Danish)

Summary

In this thesis we address the problem of learning the cardinalities of hidden variables and model parameters in tree-structured Bayesian networks with hidden variables. We work with latent class (LC) models and hierarchical latent class (HLC) models, which are among the simplest types of Bayesian networks with hidden variables for categorical data. The standard approaches for learning cardinalities and parameters in Bayesian networks with hidden variables often find only local maximum solutions or are too expensive computationally. We propose the so-called parameter reusing approach that uses parameters from a previously learned model for determining parameters of a next model. Modified parameters of the previous model are used as a starting configuration for the EM algorithm, which optimises the parameters of the next model. We reuse the parameters by splitting or merging components (i.e., states of a hidden variable). We discuss theoretical properties of such operations in the context of LC models. We propose algorithms that learn cardinalities and parameters in LC and HLC models by performing component splitting, merging, and an operation that combines these two. In experiments with synthetic and real data, these algorithms in a majority of cases performed better than the algorithms that use standard starting configurations for EM. In particular, the parameter reusing approach was better for bigger-sized training data. At the end, we propose some improvements of our algorithms and discuss how the parameter reusing approach could be extended to unrestricted Bayesian networks with hidden variables.

Acknowledgements

I would like to thank a number of people who helped me in this Ph.D. project. First, I would like to thank my supervisor Finn V. Jensen for all his guidance and support. Also I thank Tomáš Kočka for all the discussions, ideas, and for those days in Prague. I thank Nevin L. Zhang for his help concerning HLC models and especially for his source code. Also I thank Pedro Larrañaga and Jose A. Lozano for the support during my stay at the University of the Basque Country. And of course thanks to all the people I met in the Machine Intelligence (or the Decision Support Systems) group in Aalborg, in the Intelligent Systems Group in San Sebastián, and in the Institute of Information Theory and Automation in Prague.

Contents

List of Tables

List of Figures

List of Procedures

Chapter 1

Introduction

Bayesian networks (Pearl, 1988), also known as *probabilistic networks* or *Bayesian belief networks*, allow a representation of joint probability distributions in a compact way and have become popular in the field of Artificial Intelligence. As construction of Bayesian networks by using *human expert knowledge* is often a hard process, recently lots of research has been done on *automatic learning* of Bayesian networks from data (Neapolitan, 2003). For *complete data* (i.e., data where each variable from a Bayesian network is always observed), many algorithms have been proposed and strong theoretical results have been obtained (Chickering, 2002). For Bayesian networks with *hidden variables* (i.e., variables that are never observed in given data), the problem of learning a network from data becomes more challenging and the available algorithms provide only partial solutions. When learning with hidden variables, the problems of determining the *number* and *cardinalities* of hidden variables appear in addition to the usual problems of determining *links* between variables and model *parameters*. And even the problem of determining model parameters alone is much more difficult than in the case of complete data.

In this thesis, we address the problem of determining the cardinalities of hidden variables and model parameters assuming a fixed number of hidden variables and fixed links between variables in a model. The *standard approaches* for determining cardinalities and parameters often find only local maximum solutions or are too expensive computationally. We propose the so-called *parameter reusing approach* that uses parameters from a previously learned model for determining parameters of a next model. This often allows to find better solutions in the same time. We work with tree-structured Bayesian networks: *latent class* (LC) (Lazarsfeld and Henry, 1968; Goodman, 1974) and *hierarchical latent class* (HLC) (Zhang, 2004) models. They are among the simplest types of Bayesian networks with hidden variables for

categorical data. A nice feature of tree-structured Bayesian networks is that inference there is faster than in unrestricted Bayesian networks, because each variable can have no more than one parent.

This thesis is organised as follows. In next chapter we introduce the notation and definitions that will be used throughout the thesis. In Chapter 3 we overview the related work on learning with hidden variables. There we discuss learning of Bayesian networks with hidden variables, overview the related work on parameter reusing, and give an introduction to LC and HLC models. In Chapter 4 we discuss approaches for increasing the cardinality of a hidden variable while reusing the parameters in LC models. We prove the theoretical properties of the proposed approaches, discuss the implementation, and present the experiments performed. In Chapter 5 we discuss approaches for decreasing the cardinality of a hidden variable while reusing the parameters in LC models. We discuss the theoretical properties of the proposed approaches and the implementation. In Chapter 6 we discuss learning of LC models by combining the operations introduced in Chapters 4 and 5. We describe the algorithms for learning LC models and present the experiments performed. In Chapter 7 we extend our parameter reusing approach to HLC models. We describe the algorithms for learning HLC models and present the experiments performed. In Chapter 8 we conclude this thesis and discuss possible directions for further research.

Chapter 2

Notation and Definitions

In this chapter we give notation and definitions which will be used in this thesis.

A categorical or continuous *variable* is denoted by an upper-case letter (for example, D_i), a *state* of a variable by a lower-case letter (for example, d_i). A *vector of states* for a set of variables is denoted by a bold lower-case letter (for example, $\mathbf{d}$). *Data* (or *data set*) over variables $D_1, \ldots, D_k$ is $\mathbf{D} = (\langle \mathbf{d}_1, n_1 \rangle, \ldots, \langle \mathbf{d}_N, n_N \rangle)$, where each *instance* $\mathbf{d}_j$ is a k-dimensional vector, instance *weight* $n_j \geq 0$ is a real number, and $\mathbf{d}_j \neq \mathbf{d}_{j'}$ for $j \neq j'$.[1] The *size* of data $\mathbf{D}$ is $|\mathbf{D}| = \sum_{j=1}^{N} n_j$.

If a value of some D_i is not observed in some $\mathbf{d}_j$, we say that $\mathbf{d}_j$ contains *missing* data. If there exists such $\mathbf{d}_j$, we say that $\mathbf{D}$ contains *missing* data. Otherwise, $\mathbf{D}$ is called *complete*.

For a categorical variable X, $|X|$ denotes the *cardinality* (the number of possible states) and $dom(X)$ denotes the *domain* (the set of possible states) of X.

P denotes a *probability distribution* when it has variables as arguments. Otherwise, it denotes a single *probability*. $P_{\mathbf{D}}(D_1, \ldots, D_k)$ denotes the joint probability distribution obtained from $\mathbf{D}$.[2] That is,

$$P_{\mathbf{D}}(\mathbf{d}) = \begin{cases} \frac{n_j}{|\mathbf{D}|} & \text{if } \mathbf{d} = \mathbf{d}_j \text{ for some } 1 \leq j \leq N \\ 0 & \text{otherwise} \end{cases} \tag{2.1}$$

The *Kulbach-Leibler* distance between probability distributions

[1] Please note that we do not require weights n_j to be integers, but allow them to be real numbers. This will simplify a notation in some places.

[2] Here we assume that $\mathbf{D}$ over $D_1, \ldots, D_k$ is complete.

$P_1(D_1, \ldots, D_k)$ and $P_2(D_1, \ldots, D_k)$ is

$$KL(P_1, P_2) = \sum_{\mathbf{d} \in D_1 \times \ldots \times D_k} P_2(\mathbf{d}) \log \frac{P_2(\mathbf{d})}{P_1(\mathbf{d})} \tag{2.2}$$

A *model* is denoted by an upper-case letter (for example, M) and is used for representing a joint probability distribution $P_M(D_1, \ldots, D_k)$. Model M is specified by its *structure* $\mathbf{m}$ and *parameters* θ. That is, $M = (\mathbf{m}, \theta)$. The *marginal likelihood* of data $\mathbf{D}$ given model structure $\mathbf{m}$ is

$$P(\mathbf{D}|\mathbf{m}) = \int P(\mathbf{D}|\theta, \mathbf{m})\ P(\theta|\mathbf{m})\ d\theta \tag{2.3}$$

$P(\mathbf{D}|\theta, \mathbf{m}) = P(\mathbf{D}|M)$ is called the *likelihood* of $\mathbf{D}$ given M.

$$P(\mathbf{D}|M) = \prod_{j=1}^{N} P_M(\mathbf{d}_j)^{n_j} \tag{2.4}$$

where $P_M(\mathbf{d})$ is the probability of $\mathbf{d}$ given M. The *log-likelihood* of $\mathbf{D}$ given M is denoted as

$$LL(\mathbf{D}|M) = \ln P(\mathbf{D}|M) = \sum_{j=1}^{N} n_j \ln P_M(\mathbf{d}_j) \tag{2.5}$$

We say that $\mathbf{D}$ is *described perfectly* by model M if $P_M(D_1, \ldots, D_k) = P_{\mathbf{D}}(D_1, \ldots, D_k)$.

For fixed $\mathbf{D}$ and $\mathbf{m}$, the *maximum a posteriori* (MAP) parameters are

$$\begin{aligned} \theta_{MAP} = \arg\max_{\theta} P(\theta|\mathbf{D}, \mathbf{m}) &= \arg\max_{\theta} \frac{P(\mathbf{D}|\theta, \mathbf{m})\ P(\theta|\mathbf{m})}{P(\mathbf{D}|\mathbf{m})} \\ &= \arg\max_{\theta} P(\mathbf{D}|\theta, \mathbf{m})\ P(\theta|\mathbf{m}) \end{aligned} \tag{2.6}$$

and the *maximum likelihood* (ML) parameters are

$$\theta_{ML} = \arg\max_{\theta} P(\mathbf{D}|\theta, \mathbf{m}) \tag{2.7}$$

For a model M, $dim(M)$ denotes the number of independent parameters in M (that is, the *standard dimension* of M). For a model structure $\mathbf{m}$, $dim(\mathbf{m})$ is equal to $dim(M)$ of any model M having structure $\mathbf{m}$.

A variable from M that is never observed in $\mathbf{D}$ is called a *hidden* variable. Otherwise, it is called an *observed* variable. So, if M contains a hidden

variable, $\mathbf{D}$ over all the variables from M automatically contains missing data.

When using the term *structure* for models with hidden variables, we assume that cardinalities of hidden variables are also specified in a structure. We use the term *skeleton* for what is left when information about cardinalities of hidden variables is removed from a structure. So, structure $\mathbf{m} = (\mathbf{s}, \mathbf{c})$, where $\mathbf{s}$ is a skeleton of a model and $\mathbf{c}$ specifies cardinalities of hidden variables.

M is a *mixture* of m models if

$$P_M(\mathbf{d}) = \sum_{l=1}^{m} P_M(h_l)\ P_M(\mathbf{d}|h_l) \tag{2.8}$$

where $P_M(h_l)$ is the *weight* of the lth model, $P_M(\mathbf{d}|h_l)$ depends only on parameters of the lth model, and $\sum_{l=1}^{m} P_M(h_l) = 1$. So, a mixture can be seen as a model containing hidden variable H with states $h_1, \ldots, h_m$, where each state corresponds to one of m models. For mixtures, we will also use the term *component* to indicate one of m models (i.e., one of states $h_1, \ldots, h_m$).

$\mathbf{D}_l$ denotes the part of $\mathbf{D}$ that probabilistically belongs to component h_l in a mixture model. Formally,

$$\mathbf{D}_l = (\langle \mathbf{d}_1, n_{l1} \rangle, \ldots, \langle \mathbf{d}_N, n_{lN} \rangle) \tag{2.9}$$

where $n_{lj} = n_j P_M(h_l|\mathbf{d}_j)$ (so, $\sum_{l=1}^{m} n_{lj} = n_j, \forall j = 1, \ldots, N$). Here

$$P_M(h_l|\mathbf{d}_j) = \frac{P_M(h_l) P_M(\mathbf{d}_j|h_l)}{P_M(\mathbf{d}_j)} \tag{2.10}$$

A mixture model M is a *Gaussian mixture* if each $P_M(\mathbf{d}|h_l)$ is a k-dimensional Gaussian distribution. That is,

$$P_M(\mathbf{d}|h_l) = (2\pi)^{-\frac{k}{2}} |\mathbf{C}_l|^{-\frac{1}{2}} \exp\left(-\frac{1}{2} (\mathbf{d} - \mathbf{m}_l)^T \mathbf{C}_l^{-1} (\mathbf{d} - \mathbf{m}_l) \right) \tag{2.11}$$

where mean vector $\mathbf{m}_l$ and covariance matrix $\mathbf{C}_l$ constitute the parameters of component h_l.

Model $M = (\mathbf{m}, \theta)$ is a *Bayesian network* with variables $X_1, \ldots, X_n$ if (1) structure $\mathbf{m}$ corresponds to a directed acyclic graph with nodes $X_1, \ldots, X_n$ [3] and (2) parameters θ consist of probability distributions $P_M(X_i|pa(X_i))$, $i = 1, \ldots, n$, where $pa(X)$ is a set of parents of X in $\mathbf{m}$. Then

$$P_M(X_1, \ldots, X_n) = \prod_{i=1}^{n} P_M(X_i|pa(X_i)) \tag{2.12}$$

[3] We use the terms *node* and *variable* interchangeably.

In this thesis, we consider only Bayesian networks with categorical variables. x_i^v denotes the vth state of variable X_i and $\mathbf{pa}_i^u$ denotes the uth configuration of parents of X_i. q_i denotes the number of possible configurations of parents of X_i (i.e., $q_i = \prod_{Y \in pa(X_i)} |Y|$).

Note that for a Bayesian network M

$$dim(M) = \sum_{i=1}^{n} (|X_i| - 1)\ q_i \tag{2.13}$$

Chapter 3

Related Work

In this chapter we overview the related work on learning with hidden variables. First, we discuss learning the structure of Bayesian networks with hidden variables, then estimating their parameters. Then we overview the related work on parameter reusing. Finally, we give an introduction to latent class and hierarchical latent class models.

3.1 Learning the Structure with Hidden Variables

Much research has been done on learning Bayesian network structure from complete data (Heckerman, 1995), and strong theoretical results have been obtained for tree-structured networks in particular (Chow and Liu, 1968) and for unrestricted networks in general (Chickering, 2002). When learning from complete data $\mathbf{D}$, it is not difficult to evaluate a Bayesian network structure $\mathbf{m}$, because the marginal likelihood $P(\mathbf{D}|\mathbf{m})$ has a closed-form solution:

$$P(\mathbf{D}|\mathbf{m}) = \prod_{i=1}^{n} \prod_{u=1}^{q_i} \frac{\Gamma(\alpha_{iu})}{\Gamma(\alpha_{iu} + N_{iu})} \cdot \prod_{v=1}^{|X_i|} \frac{\Gamma(\alpha_{iuv} + N_{iuv})}{\Gamma(\alpha_{iuv})} \tag{3.1}$$

where N_{iuv} is the sum of weights of instances where X_i is in state x_i^v and $pa(X_i)$ are in configuration $\mathbf{pa}_i^u$ (i.e., $N_{iuv} = \sum_{j:\ X_i=x_i^v, pa(X_i)=\mathbf{pa}_i^u \text{ in } \mathbf{d}_j} n_j$), α_{iuv} is the corresponding parameter of the Dirichlet prior, $N_{iu} = \sum_{v=1}^{|X_i|} N_{iuv}$, and $\alpha_{iu} = \sum_{v=1}^{|X_i|} \alpha_{iuv}$.

The problem of learning Bayesian network structure becomes more challenging when some data is missing, especially when there are hidden variables. As shown by Cooper and Herskovits (1992) and Cooper (1995), in

theory one could consider all the possible assignments of hidden variable states to the instances in the data set and then use Formula 3.1 for complete data to evaluate a Bayesian network structure. However, in practice to consider all the possible assignments of hidden variable states is extremely expensive computationally.

Probably the best known practical contribution is the *Structural EM* algorithm (Friedman, 1997; Friedman, 1998), where the EM algorithm (described in Section 3.2) is extended to learning structures with hidden variables. The main idea is to complete data as in the E-step of the standard EM, and then to use the completed data to evaluate adjustments of a structure. These two steps are repeated iteratively. The algorithm has been proved to converge and experiments show good performance. The disadvantage is that the number of hidden variables and the cardinality of each hidden variable must be fixed.

3.1.1 Detecting Hidden Variables

The problem of automatically detecting hidden variables has been addressed by many researchers.

Pearl (1986) proposes an algorithm for learning tree-structured Bayesian networks where non-leafs are hidden variables. All the variables are assumed to be binary. The network structure is learned by computing *correlations* for pairs of observed variables. The algorithm is fast and it discovers the true Bayesian network. However, it assumes that the observed variables can be described by such a tree-structured network and that precise correlation coefficients are known. Liu et al. (1990) extend the algorithm of Pearl (1986) by allowing errors in correlation data occur. Their algorithm performs greedy search for network structure. This algorithm does not have theoretical guarantees any longer, but is applicable to real data.

The FCI algorithm of Spirtes et al. (1993); Spirtes et al. (2000) learns the Bayesian network structure by performing multiple *tests of conditional independence*. Under some assumptions, the algorithm sometimes indicates the existence of a hidden common cause for observed variables. However, the algorithm does not perform well on data of small size.

The algorithm of Connolly (1993) learns tree-structured Bayesian networks where non-leafs are hidden variables. Differently from Pearl (1986) and Liu et al. (1990), the variables are not required to be binary. Observed variables are grouped by computing *mutual information*. The cardinalities of hidden variables are determined by *conceptual clustering* (Fisher, 1987).

The algorithm of Martin and VanLehn (1995) learns Bayesian networks that contain edges only from hidden variables to observed ones. The al-

gorithm computes *correlation* for each pair of observed variables and then introduces hidden variables to explain dependencies between observed ones. To determine the cardinalities of hidden variables and model parameters, the algorithm uses a fast approximation of the normative approach of Cooper and Herskovits (1992); Cooper (1995). Instead of considering assignments of hidden variable states for all the instances in the data set at once, it considers these assignments for one instance at a time. In the beginning, the cardinality of each hidden variable is set to 1. Then the algorithm goes through all the instances, one at a time. For the jth instance, it finds the most probable state of each hidden variable (or introduces the new state of a hidden variable, if this is the best for describing the jth instance) given the current model, which is based on previous $j-1$ instances. Then the algorithm updates model parameters according to these most probable assignments of hidden variable states.

Kwoh and Gillies (1996) propose an algorithm that creates a hidden node when the observed variables having the same parent are not conditionally independent. The hidden node is then introduced between those variables and their parent. The algorithm searches for conditional dependencies by computing *conditional correlation* and/or *mutual information* between observed variables in the already available network.

Ramachandran and Mooney (1998) propose an algorithm that can add hidden nodes in order to improve a classification accuracy of *Bayesian network classifiers* with noisy-or and noisy-and nodes. For *Dynamic Bayesian networks*, Boyen et al. (1999) introduce hidden variables by searching for violations of the Markov property.

Elidan et al. (2000) propose a heuristic for detecting hidden variables in Bayesian networks. First, a network over the observed variables is learned. Next, a search for "structural signatures" of hidden variables in the learned network is performed. The algorithm searches for *semi-cliques* – sets of variables where each variable is linked to at least half of the variables in that set. Candidate hidden variables are introduced as parents of variables in semi-cliques. Finally, each candidate is evaluated and the best scoring network is selected.

Tian and Pearl (2002) propose a systematic procedure of identifying *functional constraints* induced by Bayesian networks with hidden variables. Their procedure can be used for inferring structures with hidden variables from data.

3.1.2 Learning the Cardinalities

The simplest and probably the most popular approach for learning the cardinality of a hidden variable is to learn independently a different model for different cardinalities of a hidden variable and then select the best model (Chickering and Heckerman, 1997; Uebersax, 2001; NorsysSoftwareCorp., 2005). However, this requires a separate estimation of parameters for each model, which can be very time consuming.

The algorithms of Connolly (1993) and Martin and VanLehn (1995), described in the previous section, try to determine the cardinalities of hidden variables.

Another fast heuristic approach has been proposed by Elidan and Friedman (2001). For each instance from training data, an assignment to a particular state of a hidden variable is maintained. Because of this, complete data scoring function can be used. The algorithm starts with a maximal possible number of states of a hidden variable and merges states pairwise in a greedy way until all the states are merged into one state. Two states are merged into a new one by assigning to the new state all the instances that have previously been assigned to one of these two states. When merging in a greedy way, the algorithm selects to merge such two states that the resulting model has the highest complete data score. After a separate model for each cardinality has been obtained, the algorithm selects the cardinality that corresponds to the highest scoring model. The cardinalities of two or more interacting hidden variables are learned by repeatedly selecting one variable and learning its cardinality while keeping the cardinalities of other variables fixed.

3.1.3 Scoring the Structures

When selecting among structures of Bayesian networks with hidden variables, some problems with scoring methods also appear. For missing data, the closed-form solution (Formula 3.1) for determining the marginal likelihood of complete data can no longer be used. As Chickering and Heckerman (1997) point out, one must use *large sample approximations* (such as Laplace, BIC, Cheeseman-Stutz scores) or *Monte-Carlo approaches*, which are more accurate but also more expensive computationally. When computing a large sample approximation for a structure $\mathbf{m}$, ML or MAP parameters θ' for $\mathbf{m}$ have to be computed. That is why we will also say that a large sample approximation evaluates a *parameterised* model $M = (\mathbf{m}, \theta)$, where θ is an estimate of θ'.

Mostly we will use a well-known BIC score (Schwarz, 1978), defined as

$$BIC(M) = LL(\mathbf{D}|M) - \frac{dim(M)}{2} \ln |\mathbf{D}| \tag{3.2}$$

Assuming that the parameters of model M are those of maximum likelihood, BIC score is asymptotically correct for Bayesian networks without hidden variables (Schwarz, 1978; Haughton, 1988; Geiger et al., 2001).

Sometimes we will use Cheeseman-Stutz (CS) score (Cheeseman and Stutz, 1995), defined as

$$CS(M) = LL(\mathbf{D}|M) - LL(\mathbf{D}'|M) + \log P(\mathbf{D}'|\mathbf{m}) \tag{3.3}$$

where $\mathbf{m}$ is the structure of model M and $\mathbf{D}'$ is obtained by completing $\mathbf{D}$ using model M. Here $P(\mathbf{D}'|\mathbf{m})$ has a closed-form solution (Formula 3.1), because data $\mathbf{D}'$ is complete. In the experiments of Chickering and Heckerman (1997), CS was more accurate than the BIC score.

When a Bayesian network contains hidden variables, BIC, CS and other large sample approximations can fail because of the following reasons. First, the same model can be obtained by relabeling states of a hidden variable (Chickering and Heckerman, 1997). This means that for a model with a hidden variable that has m states there exist not one but $m!$ maximum likelihood parameterisations, while in a derivation of the BIC and CS scores it is assumed that the likelihood function has a single maximum.

Second, the model dimension in the BIC score sometimes should be lower than the standard dimension $dim(M)$. Geiger et al. (1996) argue that the dimension of a Bayesian network with hidden variables is the rank of the Jacobian matrix of the transformation between the parameters of the network and the parameters of the joint probability distribution over all the observed variables. This rank is called the *effective dimension* of a Bayesian network $M = (\mathbf{m}, \theta)$. For any fixed structure $\mathbf{m}$, the effective dimension was shown to be constant for almost any parameterisation θ. This constant is called the effective dimension of structure $\mathbf{m}$.

Third, for some singular data $\mathbf{D}$ the Laplace approximation, which is a base for the BIC and CS scores, is not correct (Rusakov and Geiger, 2003). However, in spite of these properties, in practice the standard BIC score is often used for models with hidden variables (Barash and Friedman, 2001; Beal and Ghahramani, 2003; Zhang, 2004; Zhang and Kočka, 2004).

Recently, Beal and Ghahramani (2003) proposed to use the *variational Bayesian* method for scoring structures of Bayesian networks with hidden variables. The method computes a lower bound on the marginal likelihood.

The main idea is to approximate the joint probability distribution over hidden variables and parameters by a product of the probability distribution over hidden variables only and the probability distribution over parameters only. The algorithm optimises the probability distribution over hidden variables while holding the probability distribution over parameters fixed and after that optimises the probability distribution over parameters while holding the probability distribution over hidden variables fixed. These two steps are iterated until convergence. So, this method can be seen as a modification of the standard EM algorithm for Bayesian networks.

3.2 Estimating the Parameters

Computing maximum a posteriori (MAP) or maximum likelihood (ML) parameters for a given structure of a Bayesian network with hidden variables often is also a difficult problem. For a variable that is hidden or has a hidden parent, the MAP or ML parameters can not be computed in a closed-form, as it is done in the case of complete data (Heckerman, 1995). Iterative methods, such as the EM algorithm (Dempster et al., 1977), Gibbs sampling (Geman and Geman, 1984; Madigan and York, 1995), or the gradient ascent (Binder et al., 1997) have to be used.

Probably the most popular choice is the *EM algorithm*. Generally, the EM algorithm estimates parameters θ of a model M when training data for M is not complete. This is done by iteratively alternating between the expectation (E) and maximisation (M) steps. In the E-step, the current value of θ and the observed data are used to estimate values for missing data. This way, the complete data is obtained. In the M-step, this complete data is used to compute the new value of θ. Lauritzen (1995) described how to apply the EM algorithm to Bayesian networks. In the E-step, the following expected sufficient statistics are computed:

$$N_{iuv} = \sum_{j=1}^{N} n_j \cdot P_M(x_i^v, \mathbf{pa}_i^u | \mathbf{d}_j) \tag{3.4}$$

where $P_M(x_i^v, \mathbf{pa}_i^u | \mathbf{d}_j)$ is the probability of X_i being in state x_i^v and $pa(X_i)$ being in configuration $\mathbf{pa}_i^u$ given current parameterisation of a Bayesian network M and data instance $\mathbf{d}_j$. If X_i or a variable from $pa(X_i)$ are not observed in $\mathbf{d}_j$, this probability can be computed by entering $\mathbf{d}_j$ as an evidence and performing inference in a Bayesian network (for example, using the algorithm of Jensen et al. (1990)). Then, in the M-step, these expected sufficient statistics are used to compute new parameters of a Bayesian network in the same way as computing ML or MAP parameters for complete

data. That is, for ML parameters

$$P_M(x_i^v|\mathbf{pa}_i^u) = \frac{N_{iuv}}{\sum_{v'=1}^{|X_i|} N_{iuv'}} \tag{3.5}$$

and for MAP parameters with Dirichlet prior parameters α_{iuv}

$$P_M(x_i^v|\mathbf{pa}_i^u) = \frac{\alpha_{iuv} + N_{iuv}}{\sum_{v'=1}^{|X_i|}(\alpha_{iuv'} + N_{iuv'})} \tag{3.6}$$

Dempster et al. (1977) showed that the EM algorithm converges to a local maximum (in terms of $LL(\mathbf{D}|M)$). The initial parameters, that is the starting point for EM, are usually taken randomly. Meila and Heckerman (1998) performed the experiments with starting points obtained from training data or from a clustering algorithm and found that they give very similar performance to the random starting points.

The problem with using the EM algorithm is that Bayesian networks with hidden variables usually have many local maxima. So, the EM algorithm often finds local rather than global maximum parameters. It seems that this problem becomes more serious as the number of states of a hidden variable increases (Uebersax, 2000). The standard way of dealing with this problem is to run the EM algorithm many times from random starting points. The more starting points are used, the closer to the global maximum the best final parameters should be. However, often a high number of starting points is required, thus making the algorithm computationally very expensive. A more feasible computationally is the *multiple restart EM* algorithm (Chickering and Heckerman, 1997), where many different random starting points are used, but repeatedly, after a specified number of EM iterations, only parameterisations giving the highest likelihood are retained. Even though this algorithm is faster, often it still requires much time. Other proposals for escaping local maxima include simulated annealing (Kirkpatrick et al., 1983), tabu search (Glover and Laguna, 1993), using a subsample of the whole training data to create a starting point (Fayyad et al., 1998), reweighting of instances in training data (Elidan et al., 2002), using the Information Bottleneck principle (Elidan and Friedman, 2003), and reusing parameters from already available models (Section 3.3).

Another problem is that the EM algorithm can converge slowly. Much research has been done trying to speed up the EM algorithm for Bayesian networks in particular (Thiesson, 1995; Zhang, 1996; Bauer et al., 1997; Fischer and Kersting, 2003) and in other settings (Jamshidian and Jennrich, 1997; McLachlan and Krishnan, 1997; Meng and van Dyk, 1997; Bradley et al., 1998; Neal and Hinton, 1998; Moore, 1999; Ortiz and Kaelbling, 1999; Thiesson et al., 1999; Celeux et al., 2001; Salakhutdinov and Roweis, 2003).

3.3 Parameter Reusing

In this section we overview the approaches that iteratively use parameters from a previously learned model for estimating parameters of a next model. Often, by estimating parameters, at the same time the cardinality of a hidden variable is learned.

Several authors propose to learn mixture models by starting with a single component and then repeatedly *adding mixture components*. Vlassis and Likas (2002) use this approach for learning Gaussian mixtures. They add the mth component while keeping the relative mixture weights and parameters of the first $m-1$ components. For the mean of the mth component, each point in training data is considered as a candidate. The best candidate is taken as a starting point for EM. This EM learns the weight and parameters of the mth component only. This is called the *partial EM*. After that the weights and parameters of all the mixture components are adjusted by the standard EM. Verbeek et al. (2003) modify the above algorithm by initialising the mean of the mth component in such a way that this new component would correspond to a part of one of the $m-1$ components. That is, one of the $m-1$ components is split into two parts, and one of these parts is taken as the mth component. Such parameters of the mth component are taken as a starting point for the partial EM. Verbeek et al. (2003) report an improvement over the algorithm of Vlassis and Likas (2002). Blekas and Likas (2004) adapt the method of Vlassis and Likas (2002) for categorical data. Their algorithm learns what we call latent class models. The parameters of the mth component are initialised in the following way. The whole training data is partitioned according to some method into a prespecified number of parts. Then each part is considered as a candidate mth component. As in algorithm of Vlassis and Likas (2002), the best candidate is taken as a starting point for the partial EM. The usage of the standard EM afterwards is not indicated by the authors. Meek et al. (2002) propose a method for learning mixtures of density, regression, and classification models. They also add a new component while keeping the relative mixture weights and parameters of the previous components. Once a component has been added, its parameters can not be changed later, when adding more components.

Figueiredo and Jain (2002) propose to learn mixture models by starting with too many components and then *removing unnecessary components*. Their algorithm incorporates automatic component removal into parameter estimation. This is achieved by introducing Dirichlet priors with negative parameters for the weights of components and then computing MAP estimates of these weights. This way, in the M-step of the EM algorithm, the

weights of components having weak support in data are set to zero. That is, such components are removed. Also, when several components have similar parameters, Dirichlet priors with negative parameters promote competition among such components, eventually leaving only one of them. In the experiments with Gaussian mixtures, a large number of initial components are parameterised randomly, and then the algorithm iteratively estimates the parameters (which includes the removal of components). Additionally, after the algorithm has converged, the component having the lowest positive weight is removed and again the parameters are estimated in the same way. All this is repeated until a prespecified minimum number of components is reached. The model having the highest score is returned. Brand (1999) uses a similar approach for learning hidden Markov models. His algorithm also starts with many components and uses MAP estimates for removing weakly supported components. This algorithm uses the minimum-entropy prior, which contrary to the Dirichlet prior, does not lead to the closed-form solution of the M-step.

Ueda et al. (2000) propose an algorithm for improving parameters of a mixture model with a fixed number of components. Their algorithm tries to overcome the local maxima problem by repeatedly performing simultaneous *splitting* of a component and *merging* of two other components. A component is split into two new components by setting the weights of the new components to be half of the initial component's weight and the parameters to be small random perturbations of the initial component's parameters. Two components are merged into a new component by setting the weight of the new component to be the sum of the initial components' weights and the parameters to be a weighted average of the initial components' parameters. Parameterisation obtained by simultaneous splitting and merging is taken as a starting point for the EM algorithm. The algorithm uses heuristic criteria for selecting the most promising candidates for splitting and for merging. For splitting, a component h_s is considered to be promising if the distance between the following probability distributions is large: the empirical distribution given by $\mathbf{D}_s$ and the distribution specified by the parameters of the component. For merging, a pair of components is considered to be promising if many instances from training data belong to these two components with similar probabilities. The algorithm repeatedly tries to improve the parameters by splitting-merging and then running the EM algorithm. The algorithm stops when none of the several most promising split-merge operations with EM after them improve the parameters. The algorithm has been applied to the training of Gaussian mixtures and mixtures of factor analysers. Ueda and Ghahramani (2002) extend the algorithm of Ueda et al. (2000) so that it is used together with the variational Bayesian method. Here, their al-

gorithm learns both the number of components and the parameters of a model by considering separately merge, split-merge, and split operations. During each iteration, operations of each of the three types are tried. At the end of each iteration, the algorithm applies an operation which gives the highest increase in the scoring function. The algorithm stops when none of the several most promising operations of each type increase the score. For selecting the most promising candidates, the same criteria as in Ueda et al. (2000) are used. The algorithm has been applied to continuous data.

For clustering of continuous data, splitting and merging of clusters together with the *k-means* procedure has been used already by Ball and Hall (1967). The algorithm of Richardson and Green (1997) for learning Gaussian mixtures by using *Markov chain Monte Carlo* method, tries both splitting/merging of components and adding/removing them.

For learning *hidden Markov models*, successive state splitting (Takami and Sagayama, 1992; Montacié et al., 1996; Ostendorf and Singer, 1997; Stenger et al., 2001) and merging (Stolcke and Omohundro, 1994) have been used.

3.4 LC and HLC Models

LC and HLC models are among the simplest types of Bayesian networks with hidden variables. Here we give a brief introduction to these models and overview the approaches for learning them.

3.4.1 Latent Class Models

Latent class analysis (Lazarsfeld and Henry, 1968; Goodman, 1974) is a method for finding classes of similar instances from multivariate categorical data. Data **D** is assumed to be generated by a *latent class* (LC) model, shown in Figure 3.1. An LC model consists of a hidden *class variable* (H) and

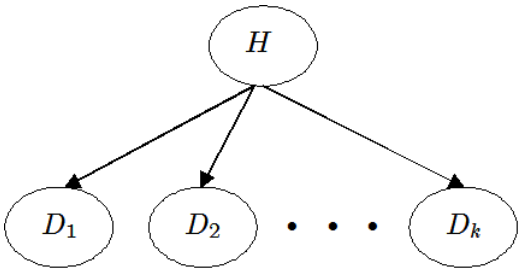

Figure 3.1: A latent class model.

observed *manifest variables* $(D_1, \ldots, D_k)$. Manifest variables are assumed to be conditionally independent given the class variable. Seeing an LC model as a mixture model, each state of the class variable corresponds to a different component (class) and a sub-model for each component consists of variables $D_1, \ldots, D_k$ with no edges. In the context of Bayesian networks, an LC model is also known as a *naive Bayes* model with a hidden class variable. The *parameters* of an LC model L consist of a marginal probability distribution $P_L(H)$ and conditional probability distributions $P_L(D_i|H),\ i = 1, \ldots, k$. If the states of H are $h_1, \ldots, h_m$, then the probability of $\mathbf{d} = (d_1, \ldots, d_k)$ given L is

$$P_L(\mathbf{d}) = \sum_{l=1}^{m} P_L(h_l) P_L(\mathbf{d}|h_l) = \sum_{l=1}^{m} P_L(h_l) \prod_{i=1}^{k} P_L(D_i = d_i|h_l) \tag{3.7}$$

If variable D_j is not observed in $\mathbf{d}$, then terms $P(D_j = d_j|h_l)$ are not present in the equation above.

An LC model gives a "soft" classification of $\mathbf{d}$. That is, $\mathbf{d}$ belongs to each class h_l with probability

$$P_L(h_l|\mathbf{d}) = \frac{P_L(h_l) P_L(\mathbf{d}|h_l)}{P_L(\mathbf{d})} \tag{3.8}$$

The goal in latent class analysis is for given data $\mathbf{D}$ to determine the optimal number of components ($|H|$) and model parameters. This is done by learning parameters for different $|H|$ and then selecting $|H|$ that gives the best model according to some criterion (for example, according to the BIC score).

To denote the structure of an LC model we will write out the cardinalities of all the variables. The structure of a model from Figure 3.1 will be denoted by $|H| : |D_1|, \ldots, |D_k|$. If $|D_1| = |D_2| = \cdots = |D_k|$, the structure will be often denoted by $|H| : k \times |D_1|$.

3.4.2 Hierarchical Latent Class Models

The assumption of an LC model that manifest variables are independent within each class is often unrealistic. To deal with this problem in a systematic way, *hierarchical latent class* (HLC) models have been introduced (Zhang, 2002; Zhang, 2004). Here, for variables that are not conditionally independent given their parent, a new parent (hidden variable) is introduced, which is made a child of an old parent. This can be done repeatedly, resulting in a tree-structured model. So, an HLC model is a Bayesian network which has a rooted tree structure, with the leaves being observed variables

and internal nodes being hidden variables. An example of an HLC model is shown in Figure 3.2. Variables D_i are observed and variables H_i are hidden.

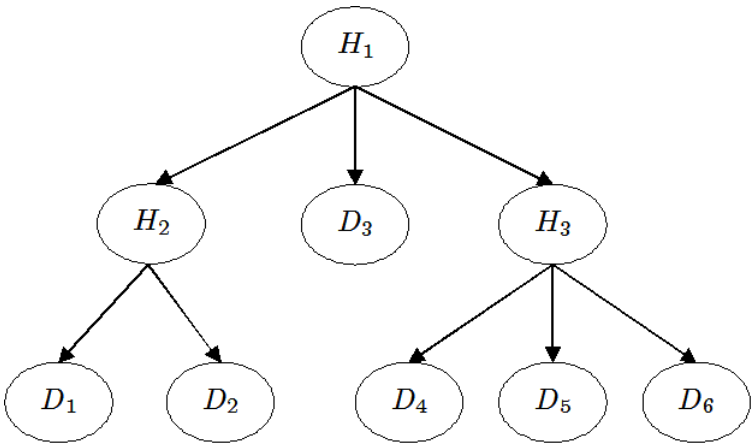

Figure 3.2: An example of a hierarchical latent class model.

Now we will define some concepts regarding HLC models. For more details, see Zhang (2002); Zhang (2004).[1] Two HLC models M and M' are *marginally equivalent* if they share the same observed variables $D_1, \ldots, D_k$ and

$$P_M(D_1, \ldots, D_k) = P_{M'}(D_1, \ldots, D_k).$$

Two models are *equivalent* if they are marginally equivalent and have the same number of independent parameters. *Root walking* in model M is an operation of reversing the arrow going from the root X to its hidden child Y. This way, a new model M' with Y as a root is obtained. Parameters $P_{M'}(Y)$ and $P_{M'}(X|Y)$ are obtained from $P_M(X)$ and $P_M(Y|X)$, while all the other parameters are kept the same. For example, a model in Figure 3.3 is obtained from a model in Figure 3.2 by root walking from H_1 to H_2. Root walking leads to an equivalent model. Because of this, the root of a model can not be determined when learning from data. Instead, one learns *unrooted HLC models*, which are HLC models with undirected edges. So, each unrooted HLC model corresponds to n usual HLC models, where n is the number of hidden variables in the model.

Regular HLC model structures have been introduced by setting upper bounds on cardinalities of hidden variables based on cardinalities of observed variables. An HLC model structure is regular if for each hidden variable H with neighbors[2] $X_1, \ldots, X_l$ the following is true:

[1] In these papers, model *structure* means the same what we call model *skeleton*.

[2] Neighbors of a variable are its parent and children.

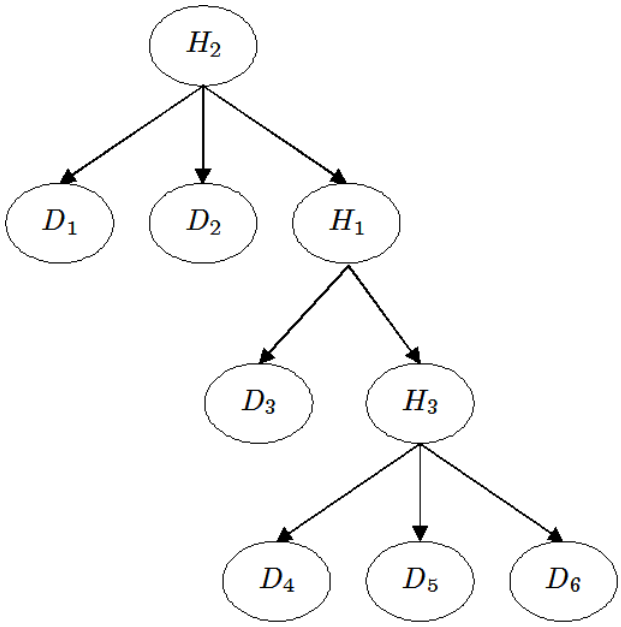

Figure 3.3: A model obtained by root walking.

- if $l = 2$, then $|H| < \frac{|X_1||X_2|}{\max\{|X_1|,|X_2|\}}$,
- if $l > 2$, then $|H| \leq \frac{\prod_{i=1}^{l}|X_i|}{\max_{i=1}^{l}|X_i|}$.

We say that an HLC model is regular if it has a regular structure. It has been proved that for each non-regular HLC model a marginally equivalent regular model can be obtained. This means that it is enough to consider only regular models when learning. Also, it has been proved that the set of regular HLC model structures is finite for a given set of observed variables.

Now we overview methods for learning HLC models. The algorithm of Connolly (1993), mentioned in Section 3.1.1, is the first attempt to learning what we call HLC models. A more systematic *double hill-climbing (DHC) algorithm* (Zhang, 2002; Zhang, 2004) performs a two level search: (1) a search for the best skeleton and (2) a search for the best structure given a fixed skeleton. When searching for the best skeleton, the algorithm starts with an LC model, and in each step it generates candidate skeletons. For each candidate, level two search is performed where the algorithm learns the cardinalities of the hidden variables. This is done by starting with a structure where cardinalities of all the hidden variables are minimum (in most cases, 2). In each step, candidate structures are generated and the best structure is selected. Each candidate structure is obtained by incrementing the car-

dinality of one hidden variable. The best structure is selected by learning parameters using the EM algorithm and then computing the score for each candidate. When none of the candidate structures increase the score, the algorithm returns to level one search, taking the score of the best structure found as the score of the candidate skeleton. In the experiments, the DHC algorithm has found good models, but its drawback is high complexity. That is why Zhang and Kočka (2004) introduced a *single hill-climbing (SHC) algorithm*, which optimises skeleton and cardinalities at the same time. The algorithm starts with a simple model (an LC model) and repeatedly increases the log-likelihood of the data by increasing the model complexity. After that, it repeatedly decreases the model complexity while keeping similar log-likelihood. These two phases of increasing and decreasing the model complexity are repeated iteratively. When increasing the model complexity, the algorithm considers candidate structures obtained by introducing a hidden variable, introducing a state in a hidden variable, and changing the parent of a variable. When decreasing the model complexity, the algorithm considers candidate structures obtained by deleting a hidden variable and deleting a state of a hidden variable. Even though being faster than the DHC algorithm, this algorithm is still inefficient because in order to estimate parameters it runs the EM algorithm for each candidate structure. That is why Zhang and Kočka (2004) also propose a *heuristic SHC (HSHC) algorithm* where, similarly to the structural EM, completed data is used for evaluating candidate structures. Only for the candidate structures that are the best according to completed data, a modified EM algorithm is run. This modified EM algorithm optimises only the parameters in that part of a model where the structure has been changed, while keeping all the other parameters fixed. That is why it is called a *local EM*. These modifications make the learning of HLC models much faster.

Chapter 4

Increasing the Cardinality

In this chapter we discuss approaches for increasing the cardinality of a hidden variable while reusing the parameters. We do this for categorical data, in the context of latent class (LC) models. Part of this chapter appeared in Karčiauskas et al. (2004a).

First we give an overview of why and how the parameters can be reused. Then we define formally how to do it. After that, we prove the theoretical properties of the proposed approaches. Then we discuss the implementation of cardinality increase. Finally, we present the experiments performed.

4.1 Overview

As mentioned in Section 3.1.2, models with different cardinalities of a hidden variable are usually learned independently of each other. However, we could try to take advantage of the fact that the states of a hidden variable H in models with different $|H|$ are not completely independent. For example, it is natural to expect that some components from an $m+1$–component LC model are similar to some components from an m–component LC model when these models are learned from the same data. So, if parameters of an m–component model have already been estimated, it may be useful to somehow use them when learning an $m+1$–component model. This may allow us to learn a better $m+1$–component model than in the case of estimating its parameters from scratch. One can think about two approaches for reusing the parameters from an m–component model.

In the first approach, which we call *component introduction*, the $m+1$–component model is initialised by adding a new component to the m–component model. All m "old" components are kept unchanged, only their weights are scaled down. The new component could be initialised ran-

domly or according to some method. Then the parameters of the $m+1$–component model can be optimised by running for example the EM algorithm. Vlassis and Likas (2002) use this approach for continuous data (learning Gaussian mixtures) and Blekas and Likas (2004) for categorical data (learning LC models), as described in Section 3.3.

In the second approach, which we call *component splitting*, one component from the m–component model is split into two new components. The $m+1$–component model is initialised to contain these two new components and the other $m-1$ "old" components. The parameters of the model can then be optimised as in the first approach. As described in Section 3.3, Ueda et al. (2000) and Ueda and Ghahramani (2002) use this approach for continuous data. Verbeek et al. (2003) use a similar approach for continuous data as well.

Next we will investigate these two approaches in more detail.

4.2 Definitions

Here we give formal definitions of component introduction and component splitting operations for LC models.

Definition 4.1 *We say that model L^* is obtained from model L by* introducing *a component if L^* contains all the components from L with their marginal probabilities scaled down and one new component. More formally, if L contains components $h_1, \ldots, h_m$, then L^* contains components $h_1, \ldots, h_{m+1}$ and the following is true for some $p \in (0;1)$:*

- $P_{L^*}(h_l) = (1-p)P_L(h_l),\ P_{L^*}(D_i|h_l) = P_L(D_i|h_l),$
  $l = 1, \ldots, m,\ i = 1, \ldots, k,$
- $P_{L^*}(h_{m+1}) = p.$

Please note that here we do not impose any restrictions for $P_{L^*}(D_i|h_{m+1})$.

Definition 4.2 *We say that model L^* is obtained from model L by* splitting *a component h_s if L^* instead of h_s contains components h_s^1 and h_s^2 that are both similar to h_s, and all the other components are identical in both models. More formally, if L contains components $h_1, \ldots, h_m$, then L^* contains components $h_1, \ldots, h_{s-1}, h_s^1, h_s^2, h_{s+1}, \ldots, h_m$ and the following is true:*

- $P_{L^*}(h_l) = P_L(h_l),\ P_{L^*}(D_i|h_l) = P_L(D_i|h_l),$
  $l = 1, \ldots, m,\ l \neq s,\ i = 1, \ldots, k,$

- $P_{L^*}(h_s^1) = P_{L^*}(h_s^2) = \frac{1}{2}P_L(h_s)$,
- $||P_{L^*}(D_i|h_s^1) - P_L(D_i|h_s)|| < \epsilon$,
  $||P_{L^*}(D_i|h_s^2) - P_L(D_i|h_s)|| < \epsilon$,
  $i = 1, \ldots, k$, *where* $||\cdot||$ *is an* L^2*-norm of a vector and* $\epsilon \in \mathbb{R}$ *is chosen in advance and is close to 0.*

4.3 Theoretical Properties

Having defined the component introduction and component splitting operations, we now look at their theoretical properties. For convenience, we assume that training data $\mathbf{D}$ over the observed variables is complete.

4.3.1 Component Introduction

We will show that if an LC model L does not describe perfectly data $\mathbf{D}$, it is possible to increase the log-likelihood of $\mathbf{D}$ by introducing a component in L. The idea is to consider a configuration $\mathbf{d}'$ from $\mathbf{D}$ for which $P_L(\mathbf{d}')$ is too small. A new component in L^* would have a very small marginal probability and would correspond to $\mathbf{d}'$ only. Below we give a formal proof [1].

Theorem 4.1 *If for data* $\mathbf{D}$ *and LC model* L *we have* $P_L(D_1, \ldots, D_k) \neq P_{\mathbf{D}}(D_1, \ldots, D_k)$*, then it is possible to obtain an LC model* L^* *from model* L *by introducing a component, such that* $LL(\mathbf{D}|L^*) > LL(\mathbf{D}|L)$.

Proof. Since $P_L(D_1, \ldots, D_k) \neq P_{\mathbf{D}}(D_1, \ldots, D_k)$, $\exists \mathbf{d}' \in D_1 \times \ldots \times D_k$ such that $P_L(\mathbf{d}') < P_{\mathbf{D}}(\mathbf{d}')$. Let L^* be an LC model obtained from model L by introducing a component (see Definition 4.1). Set

$$P_{L^*}(D_i = d|h_{m+1}) = \begin{cases} 1 & \text{if } D_i = d \text{ in } \mathbf{d}' \\ 0 & \text{otherwise} \end{cases} \tag{4.1}$$

and take p for $P_{L^*}(h_{m+1})$ so close to 0 that $P_{L^*}(\mathbf{d}') < P_{\mathbf{D}}(\mathbf{d}')$. We will show that $KL(P_{L^*}, P_{\mathbf{D}}) < KL(P_L, P_{\mathbf{D}})$.

First note that

$$P_{L^*}(\mathbf{d}) = \begin{cases} p + (1-p)P_L(\mathbf{d}) & \text{if } \mathbf{d} = \mathbf{d}' \\ (1-p)P_L(\mathbf{d}) & \text{otherwise} \end{cases} \tag{4.2}$$

[1] Contributed by Tomáš Kočka.

Then we have

$$
\begin{aligned}
KL(P_{L^*}, P_{\mathbf{D}}) &= \sum_{j=1}^{N} P_{\mathbf{D}}(\mathbf{d}_j) \log \frac{P_{\mathbf{D}}(\mathbf{d}_j)}{P_{L^*}(\mathbf{d}_j)} = -\sum_{j=1}^{N} P_{\mathbf{D}}(\mathbf{d}_j) \log \frac{P_{L^*}(\mathbf{d}_j)}{P_{\mathbf{D}}(\mathbf{d}_j)} \\
&= -\sum_{\substack{j=1 \\ \mathbf{d}_j \neq \mathbf{d}'}}^{N} P_{\mathbf{D}}(\mathbf{d}_j) \log \frac{P_{L^*}(\mathbf{d}_j)}{P_{\mathbf{D}}(\mathbf{d}_j)} - P_{\mathbf{D}}(\mathbf{d}') \log \frac{P_{L^*}(\mathbf{d}')}{P_{\mathbf{D}}(\mathbf{d}')} \\
&= -\sum_{\substack{j=1 \\ \mathbf{d}_j \neq \mathbf{d}'}}^{N} P_{\mathbf{D}}(\mathbf{d}_j) \left(\log \frac{P_L(\mathbf{d}_j)}{P_{\mathbf{D}}(\mathbf{d}_j)} + \log(1-p) \right) \\
&\quad -P_{\mathbf{D}}(\mathbf{d}') \log \frac{p + (1-p) P_L(\mathbf{d}')}{P_{\mathbf{D}}(\mathbf{d}')} \\
&= -\sum_{\substack{j=1 \\ \mathbf{d}_j \neq \mathbf{d}'}}^{N} P_{\mathbf{D}}(\mathbf{d}_j) \log \frac{P_L(\mathbf{d}_j)}{P_{\mathbf{D}}(\mathbf{d}_j)} - \sum_{\substack{j=1 \\ \mathbf{d}_j \neq \mathbf{d}'}}^{N} P_{\mathbf{D}}(\mathbf{d}_j) \log(1-p) \\
&\quad -P_{\mathbf{D}}(\mathbf{d}') \log \left(P_L(\mathbf{d}') \frac{\frac{p}{P_L(\mathbf{d}')} + 1 - p}{P_{\mathbf{D}}(\mathbf{d}')} \right) \\
&= -\sum_{\substack{j=1 \\ \mathbf{d}_j \neq \mathbf{d}'}}^{N} P_{\mathbf{D}}(\mathbf{d}_j) \log \frac{P_L(\mathbf{d}_j)}{P_{\mathbf{D}}(\mathbf{d}_j)} - (1 - P_{\mathbf{D}}(\mathbf{d}')) \log(1-p) \\
&\quad -P_{\mathbf{D}}(\mathbf{d}') \log \frac{P_L(\mathbf{d}')}{P_{\mathbf{D}}(\mathbf{d}')} - P_{\mathbf{D}}(\mathbf{d}') \log \left(\frac{p}{P_L(\mathbf{d}')} + 1 - p \right) \\
&= KL(P_L, P_{\mathbf{D}}) - (1 - P_{\mathbf{D}}(\mathbf{d}')) \log(1-p) \\
&\quad -P_{\mathbf{D}}(\mathbf{d}') \log \left(\frac{p}{P_L(\mathbf{d}')} + 1 - p \right) \\
&= KL(P_L, P_{\mathbf{D}}) + \log \frac{1}{1-p} - P_{\mathbf{D}}(\mathbf{d}') \log \left(\frac{p}{(1-p) P_L(\mathbf{d}')} + 1 \right) \quad (4.3)
\end{aligned}
$$

From Equation 4.2 we have that $P_L(\mathbf{d}') = \frac{P_{L^*}(\mathbf{d}') - p}{1-p}$. This together with $P_{L^*}(\mathbf{d}') < P_{\mathbf{D}}(\mathbf{d}')$ gives that

$$
\begin{aligned}
P_{\mathbf{D}}(\mathbf{d}') \log \left(\frac{p}{(1-p) P_L(\mathbf{d}')} + 1 \right) &= P_{\mathbf{D}}(\mathbf{d}') \log \left(\frac{p}{P_{L^*}(\mathbf{d}') - p} + 1 \right) \\
&> P_{\mathbf{D}}(\mathbf{d}') \log \left(\frac{p}{P_{\mathbf{D}}(\mathbf{d}') - p} + 1 \right) = P_{\mathbf{D}}(\mathbf{d}') \log \left(\frac{P_{\mathbf{D}}(\mathbf{d}')}{P_{\mathbf{D}}(\mathbf{d}') - p} \right) \quad (4.4)
\end{aligned}
$$

Since in general $\left(\frac{\alpha}{\alpha-p}\right)^{\alpha} \geq \frac{1}{1-p}$ for $p < \alpha \leq 1$, we have that

$$P_{\mathbf{D}}(\mathbf{d}') \log \left(\frac{P_{\mathbf{D}}(\mathbf{d}')}{P_{\mathbf{D}}(\mathbf{d}') - p} \right) = \log \left(\frac{P_{\mathbf{D}}(\mathbf{d}')}{P_{\mathbf{D}}(\mathbf{d}') - p} \right)^{P_{\mathbf{D}}(\mathbf{d}')} \geq \log \frac{1}{1-p} \tag{4.5}$$

Combining (4.3), (4.4), and (4.5) gives that

$$KL(P_{L^*}, P_{\mathbf{D}}) < KL(P_L, P_{\mathbf{D}}) \tag{4.6}$$

From this we get

$$\sum_{j=1}^{N} P_{\mathbf{D}}(\mathbf{d}_j) \log \frac{P_{\mathbf{D}}(\mathbf{d}_j)}{P_{L^*}(\mathbf{d}_j)} < \sum_{j=1}^{N} P_{\mathbf{D}}(\mathbf{d}_j) \log \frac{P_{\mathbf{D}}(\mathbf{d}_j)}{P_L(\mathbf{d}_j)} \tag{4.7}$$

$$\sum_{j=1}^{N} P_{\mathbf{D}}(\mathbf{d}_j) \log P_{L^*}(\mathbf{d}_j) > \sum_{j=1}^{N} P_{\mathbf{D}}(\mathbf{d}_j) \log P_L(\mathbf{d}_j) \tag{4.8}$$

$$\sum_{j=1}^{N} \frac{P_{\mathbf{D}}(\mathbf{d}_j)}{|\mathbf{D}|} \log P_{L^*}(\mathbf{d}_j) > \sum_{j=1}^{N} \frac{P_{\mathbf{D}}(\mathbf{d}_j)}{|\mathbf{D}|} \log P_L(\mathbf{d}_j) \tag{4.9}$$

$$LL(\mathbf{D}|L^*) > LL(\mathbf{D}|L) \tag{4.10}$$

■

4.3.2 Component Splitting

We will show that component splitting has the same theoretical property. That is, if an LC model L does not describe perfectly data $\mathbf{D}$, it is possible to increase the log-likelihood of $\mathbf{D}$ by splitting a component in L. At first glance, this statement may seem obvious, because we increase the number of parameters in a model when splitting. However, in Definition 4.2 we require that the two components obtained by splitting be very similar to the component that has been split. In other words, we have to prove that $\forall \epsilon > 0$ we can increase the log-likelihood by splitting. The general idea of the proof is to split a component by changing slightly the conditional probabilities of variables for which the probability distributions obtained from the data and from the model differ. Below we give a formal proof.

Lemma 4.1 *If for data* $\mathbf{D}$ *and an* m*–component LC model* L *we have*

$$P_{\mathbf{D}_l}(D_1, \ldots, D_k) = P_L(D_1|h_l) \cdot \ldots \cdot P_L(D_k|h_l), \; l = 1, \ldots, m \tag{4.11}$$

then $P_{\mathbf{D}}(D_1, \ldots, D_k) = P_L(D_1, \ldots, D_k)$. [2]

[2]For definition of $P_{\mathbf{D}_l}$, see (2.9) and (2.1).

Proof.

$$P_{\mathbf{D}}(D_1,\ldots,D_k) = \sum_{l=1}^{m} P(h_l) P_{\mathbf{D}_l}(D_1,\ldots,D_k)$$

$$= \sum_{l=1}^{m} P(h_l) P_L(D_1|h_l)\cdot\ldots\cdot P_L(D_k|h_l) = P_L(D_1,\ldots,D_k) \quad (4.12)$$

■

Lemma 4.2 *Let $P(X_1,\ldots,X_k)$ be a probability distribution over categorical variables $X_1,\ldots,X_k$. For any $\mathcal{X} \subset \{X_1,\ldots,X_k\}$ let $P(\mathcal{X})$ denote the distribution obtained by marginalising $\{X_1,\ldots,X_k\} \setminus \mathcal{X}$ out of $P(X_1,\ldots,X_k)$. If*

$$P(X_1,\ldots,X_k) \neq P(X_1)\cdot\ldots\cdot P(X_k) \quad (4.13)$$

then with probability 1 (in respect of parameters of $P(X_1,\ldots,X_k)$) there exist two variables $A, B \in \{X_1,\ldots,X_k\}$ such that

$$P(A,B) \neq P(A)P(B) \quad (4.14)$$

Proof. $P(X_1,\ldots,X_k)$ is determined by $|X_1|\cdot\ldots\cdot|X_k|-1$ free parameters. Let $\mathcal{Q}$ denote the space of all the possible values of these parameters. Let $\mathcal{R}$ denote the subspace of $\mathcal{Q}$ where the parameters satisfy $P(X_1,\ldots,X_k) = P(X_1)\cdot\ldots\cdot P(X_k)$. Since this puts constraints on $\mathcal{Q}$, $\mathcal{R}$ has measure 0 in $\mathcal{Q}$. So, $\overline{\mathcal{R}}$ has measure 1 in $\mathcal{Q}$.[3]

Let $\mathcal{S}$ denote the subspace of $\mathcal{Q}$ where for some variables $A, B \in \{X_1,\ldots,X_k\}$ we have $P(A,B) = P(A)P(B)$. Since this puts constraints on $\mathcal{Q}$, $\mathcal{S}$ has also measure 0 in $\mathcal{Q}$. So, $\overline{\mathcal{S}}$ has measure 1 in $\mathcal{Q}$.

So, $\overline{\mathcal{S}}$ has measure 1 in $\overline{\mathcal{R}}$. This means that with probability 1 there exist variables A, B for which Inequality 4.14 holds. ■

One may wonder whether a stronger statement, saying that there always (not just with probability 1) exist such variables A and B, is true. Below we provide an example where such A and B do not exist.

Example. Let X_1, X_2, and X_3 be variables with states $\{0,1\}$ and let $P(X_1,X_2,X_3)$ be as shown in Table 4.1.

It is easy to check that $P(X_1,X_2,X_3) \neq P(X_1)P(X_2)P(X_3)$, but $P(X_1,X_2) = P(X_1)P(X_2)$, $P(X_1,X_3) = P(X_1)P(X_3)$, $P(X_2,X_3) = P(X_2)P(X_3)$.

[3]We use $\overline{\mathcal{U}}$ to denote the complement of a set $\mathcal{U}$.

	$X_1 = 0$		$X_1 = 1$	
	$X_2 = 0$	$X_2 = 1$	$X_2 = 0$	$X_2 = 1$
$X_3 = 0$	0.05	0.10	0.15	0.20
$X_3 = 1$	0.07	0.08	0.13	0.22

Table 4.1: An example of $P(X_1, X_2, X_3)$ distribution.

Lemma 4.3 *Let $P(A, B)$ be a probability distribution over categorical variables A and B. Let $P(A)$ and $P(B)$ be the distributions obtained by marginalising respectively B and A out of $P(A, B)$. If*

$$P(A, B) \neq P(A)P(B) \tag{4.15}$$

then there exist states a_1, a_2 of A and states b_1, b_2 of B such that

$$\frac{P(a_1, b_1)}{P(a_1)P(b_1)} + \frac{P(a_2, b_2)}{P(a_2)P(b_2)} \neq \frac{P(a_1, b_2)}{P(a_1)P(b_2)} + \frac{P(a_2, b_1)}{P(a_2)P(b_1)} \tag{4.16}$$

Proof. Let $a_1, \ldots, a_u$ be all the states of A, and $b_1, \ldots, b_v$ be all the states of B. Denote

$$s_{ij} = \text{sgn}(P(a_i, b_j) - P(a_i)P(b_j)) \tag{4.17}$$

$1 \leq i \leq u, 1 \leq j \leq v$. We will show that there exist i_1, i_2, j_1, j_2 such that

$$s_{i_1 j_1} \geq 0, s_{i_2 j_2} \geq 0, s_{i_1 j_2} \leq 0, s_{i_2 j_1} < 0 \tag{4.18}$$

This will mean that by reordering the states of A and B (taking $a_1 = a_{i_1}, a_2 = a_{i_2}, b_1 = b_{i_1}, b_2 = b_{i_2}$) we can make Inequality 4.16 true, because then its left side becomes higher than or equal to 2 and its right side strictly lower than 2.

Assume that i_1, i_2, j_1, j_2 satisfying Condition 4.18 do not exist.

Let S be the matrix of s_{ij}, where i is a row index and j is a column index. Since

$$\sum_{i=1}^{u} P(a_i, b_j) = P(b_j) = \sum_{i=1}^{u} P(a_i)P(b_j) \tag{4.19}$$

then each column j of S containing $s \neq 0$ must also contain $-s$. Similarly, each row i of S containing $s \neq 0$ must also contain $-s$.

Inequality 4.15 means that there exists $s_{ij} \neq 0$. Let $i' = \min\{i : \exists j \text{ such that } s_{ij} \neq 0\}$. Let $J_{+1} = \{j : s_{i'j} = 1\}, J_{-1} = \{j : s_{i'j} = -1\}, J_0 = \{j : s_{i'j} = 0\}$. Define function $r : J_{+1} \to \{i' + 1, \ldots, u\}$ in the following way. $r(j) = \min\{i : s_{ij} = -1\}$. This means that

$$\forall j \in J_{+1} : s_{ij} \geq 0, \ i = i', \ldots, r(j) - 1 \tag{4.20}$$

Let $i'' = \max_{j \in J_{+1}} r(j)$. Let $j' \in R_{+1}$ be such that $r(j') = i''$. Together with (4.20) this means that

$$s_{ij'} \geq 0,\ i = i', \ldots, i'' - 1 \tag{4.21}$$

Then:

- $\forall j \in J_{+1} : s_{i''j} = -1$. Otherwise, if $\exists j'' \in J_{+1} : s_{i''j''} \geq 0$ then for $i_1 = r(j''), i_2 = i'', j_1 = j', j_2 = j''$ we have $s_{i_1 j_1} \geq 0$ (follows from $r(j'') < i''$ together with (4.21)), $s_{i_2 j_2} \geq 0$, $s_{i_1 j_2} = -1$ (follows from the definition of function r), and $s_{i_2 j_1} = -1$ (follows from the definition of j'). So i_1, i_2, j_1, j_2 would satisfy Condition 4.18.
- $\forall j \in J_{-1} \cup J_0 : s_{i''j} = -1$. Otherwise, if $\exists j'' \in J_{-1} \cup J_0 : s_{i''j''} \geq 0$ then for $i_1 = i', i_2 = i'', j_1 = j', j_2 = j''$ we have $s_{i_1 j_1} \geq 0$ (follows from (4.21)), $s_{i_2 j_2} \geq 0$, $s_{i_1 j_2} \leq 0$ (because $j'' \in J_{-1} \cup J_0$), and $s_{i_2 j_1} = -1$ (follows from the definition of j'). So i_1, i_2, j_1, j_2 would satisfy Condition 4.18.

So, $\forall j : s_{i''j} = -1$, which can not be true. So, the assumption that i_1, i_2, j_1, j_2 satisfying Condition 4.18 do not exist was wrong. ■

Theorem 4.2 *If for data* $\mathbf{D}$ *and LC model* L *we have* $P_L(D_1, \ldots, D_k) \neq P_{\mathbf{D}}(D_1, \ldots, D_k)$*, then with probability 1 (in respect of* $\mathbf{D}$ *and parameters of* L*) it is possible to obtain an LC model* L^* *from model* L *by splitting a component, such that* $LL(\mathbf{D}|L^*) > LL(\mathbf{D}|L)$*.*

Proof. According to Lemma 4.1, exists component h_s such that

$$P_{\mathbf{D}_s}(D_1, \ldots, D_k) \neq P_L(D_1|h_s) \cdot \ldots \cdot P_L(D_k|h_s) \tag{4.22}$$

Assume that $\forall i : P_{\mathbf{D}_s}(D_i) = P_L(D_i|h_s)$ (the case when this is not true will be considered at the end). Then, according to Lemma 4.2, with probability 1 there exist variables $A, B \in \{D_1, \ldots, D_k\}$ such that

$$P_{\mathbf{D}_s}(A, B) \neq P_L(A|h_s) P_L(B|h_s) \tag{4.23}$$

For convenience, we will assume that $A = D_1$ and $B = D_2$. According to Lemma 4.3, there exist states a_1, a_2 of A and states b_1, b_2 of B such that

$$\begin{aligned} &\frac{P_{\mathbf{D}_s}(a_1, b_1)}{P_L(a_1|h_s) P_L(b_1|h_s)} + \frac{P_{\mathbf{D}_s}(a_2, b_2)}{P_L(a_2|h_s) P_L(b_2|h_s)} \neq \\ &\frac{P_{\mathbf{D}_s}(a_1, b_2)}{P_L(a_1|h_s) P_L(b_2|h_s)} + \frac{P_{\mathbf{D}_s}(a_2, b_1)}{P_L(a_2|h_s) P_L(b_1|h_s)} \end{aligned} \tag{4.24}$$

Let us produce model L^* by splitting component h_s from L into components h_s and h_{m+1} from L^* (the same way as in Definition 4.2 h_s is split into h_s^1 and h_s^2). Set $P_{L^*}(D_i|h_s)$ and $P_{L^*}(D_i|h_{m+1})$ in the following way:

$$
\begin{aligned}
P_{L^*}(D_i|h_s) = P_{L^*}(D_i|h_{m+1}) &= P_L(D_i|h_s), \forall i > 2 && (4.25)\\
P_{L^*}(a|h_s) = P_{L^*}(a|h_{m+1}) &= P_L(a|h_s), \forall a \in dom(A) \setminus \{a_1, a_2\} && (4.26)\\
P_{L^*}(b|h_s) = P_{L^*}(b|h_{m+1}) &= P_L(b|h_s), \forall b \in dom(B) \setminus \{b_1, b_2\} && (4.27)\\
P_{L^*}(a_1|h_s) &= P_L(a_1|h_s) + \sqrt{\epsilon} && (4.28)\\
P_{L^*}(a_2|h_s) &= P_L(a_2|h_s) - \sqrt{\epsilon} && (4.29)\\
P_{L^*}(b_1|h_s) &= P_L(b_1|h_s) + \sqrt{\epsilon} && (4.30)\\
P_{L^*}(b_2|h_s) &= P_L(b_2|h_s) - \sqrt{\epsilon} && (4.31)\\
P_{L^*}(a_1|h_{m+1}) &= P_L(a_1|h_s) - \sqrt{\epsilon} && (4.32)\\
P_{L^*}(a_2|h_{m+1}) &= P_L(a_2|h_s) + \sqrt{\epsilon} && (4.33)\\
P_{L^*}(b_1|h_{m+1}) &= P_L(b_1|h_s) - \sqrt{\epsilon} && (4.34)\\
P_{L^*}(b_2|h_{m+1}) &= P_L(b_2|h_s) + \sqrt{\epsilon} && (4.35)
\end{aligned}
$$

where $\epsilon \in \mathbb{R}$.

Then for $\mathbf{d} = (d_1, \ldots, d_k)$

$$
\begin{aligned}
P_{L^*}(\mathbf{d}) &= \sum_{\substack{l=1\\ l\neq s}}^{m} P_{L^*}(h_l)P_{L^*}(\mathbf{d}|h_l) + P_{L^*}(h_s)P_{L^*}(\mathbf{d}|h_s)\\
&\quad + P_{L^*}(h_{m+1})P_{L^*}(\mathbf{d}|h_{m+1})\\
&= \sum_{\substack{l=1\\ l\neq s}}^{m} P_L(h_l)P_L(\mathbf{d}|h_l) + \frac{1}{2}P_L(h_s)\Big(P_{L^*}(\mathbf{d}|h_s) + P_{L^*}(\mathbf{d}|h_{m+1})\Big)\\
&= \sum_{\substack{l=1\\ l\neq s}}^{m} P_L(h_l)P_L(\mathbf{d}|h_l)\\
&\quad + \frac{1}{2}P_L(h_s)\Big(P_{L^*}(A = d_1|h_s)P_{L^*}(B = d_2|h_s)\\
&\qquad + P_{L^*}(A = d_1|h_{m+1})P_{L^*}(B = d_2|h_{m+1})\Big)\prod_{i=3}^{k} P_L(D_i = d_i|h_s)
\end{aligned}
\tag{4.36}
$$

Let $\mathbf{D}' = \{\langle \mathbf{d}, n\rangle \in \mathbf{D} : d_1 \in \{a_1, a_2\}$ and $d_2 \in \{b_1, b_2\}$ in $\mathbf{d}\}$. If $\langle \mathbf{d}, n\rangle \notin \mathbf{D}'$, then (4.36) together with (4.26) and (4.27) gives $P_{L^*}(\mathbf{d}) = P_L(\mathbf{d})$.

Now we will consider the case where $\langle \mathbf{d}, n\rangle \in \mathbf{D}'$. For $a \in \{a_1, a_2\}$ and $b \in \{b_1, b_2\}$ let us denote

$$P^*(a,b) = \frac{1}{2}\Big((P_{L^*}(a|h_s)P_{L^*}(b|h_s) + P_{L^*}(a|h_{m+1})P_{L^*}(b|h_{m+1})\Big) \tag{4.37}$$

By applying (4.28)–(4.35) we get

$$P^*(a_1,b_1) = P_L(a_1|h_s)P_L(b_1|h_s) + \epsilon \tag{4.38}$$
$$P^*(a_1,b_2) = P_L(a_1|h_s)P_L(b_2|h_s) - \epsilon \tag{4.39}$$
$$P^*(a_2,b_1) = P_L(a_2|h_s)P_L(b_1|h_s) - \epsilon \tag{4.40}$$
$$P^*(a_2,b_2) = P_L(a_2|h_s)P_L(b_2|h_s) + \epsilon \tag{4.41}$$

That is,

$$P^*(a,b) = P_L(a|h_s)P_L(b|h_s) + \sigma(a,b)\ \epsilon \tag{4.42}$$

where $\sigma(a,b) = \begin{cases} 1, & \text{if } a = a_1, b = b_1 \text{ or } a = a_2, b = b_2 \\ -1, & \text{otherwise} \end{cases}$.

Putting this into (4.36) gives that for $\mathbf{d} = (d_1, \ldots, d_k)$ from $\mathbf{D}'$

$$\begin{aligned} P_{L^*}(\mathbf{d}) = & \sum_{\substack{l=1 \\ l\neq s}}^{m} P_L(h_l)P_L(\mathbf{d}|h_l) \\ & + P_L(h_s)\Big(P_L(A = d_1|h_s)P_L(B = d_2|h_s) + \sigma(d_1,d_2)\ \epsilon\Big) \\ & \prod_{i=3}^{k} P_L(D_i = d_i|h_s) \\ = \ & P_L(\mathbf{d}) + P_L(h_s)\prod_{i=3}^{k} P_L(D_i = d_i|h_s)\ \sigma(d_1,d_2)\ \epsilon \end{aligned} \tag{4.43}$$

So, $LL(\mathbf{D}|L^*) = LL(\mathbf{D}|L)$ for $\epsilon = 0$, and

$$\begin{aligned} \frac{\partial LL(\mathbf{D}|L^*)}{\partial \epsilon} &= \sum_{\langle \mathbf{d}, n\rangle \in \mathbf{D}'} \frac{\partial(n \ln P_{L^*}(\mathbf{d}))}{\partial \epsilon} \\ &= \sum_{\langle \mathbf{d}=(d_1,\ldots,d_k), n\rangle \in \mathbf{D}'} \frac{n}{P_{L^*}(\mathbf{d})} P_L(h_s)\prod_{i=3}^{k} P_L(D_i = d_i|h_s)\ \sigma(d_1,d_2) \end{aligned} \tag{4.44}$$

Since $P_{L^*}(\mathbf{d}) = P_L(\mathbf{d})$ for $\epsilon = 0$, we get

$$\begin{aligned}
&\frac{\partial LL(\mathbf{D}|L^*)}{\partial \epsilon}(0) \\
&= \sum_{\langle \mathbf{d}=(d_1,\ldots,d_k),n\rangle \in \mathbf{D}'} \frac{n}{P_L(\mathbf{d})} P_L(h_s) \prod_{i=3}^{k} P_L(D_i = d_i|h_s)\ \sigma(d_1,d_2) \\
&= \sum_{\langle \mathbf{d}=(d_1,\ldots,d_k),n\rangle \in \mathbf{D}'} \frac{P_L(h_s)\prod_{i=1}^{k} P_L(D_i = d_i|h_s)}{P_L(\mathbf{d})} \\
&\qquad \frac{1}{P_L(A = d_1|h_s)P_L(B = d_2|h_s)}\ n\ \sigma(d_1,d_2) \\
&= \sum_{\langle \mathbf{d}=(d_1,\ldots,d_k),n\rangle \in \mathbf{D}'} \frac{P_L(h_s|\mathbf{d})}{P_L(A = d_1|h_s)P_L(B = d_2|h_s)}\ n\ \sigma(d_1,d_2) \\
&= |\mathbf{D}_s| \sum_{\langle \mathbf{d}=(d_1,\ldots,d_k),n\rangle \in \mathbf{D}'} \frac{\sigma(d_1,d_2)}{P_L(A = d_1|h_s)P_L(B = d_2|h_s)} \frac{n\ P_L(h_s|\mathbf{d})}{|\mathbf{D}_s|} \\
&= |\mathbf{D}_s| \sum_{a\in\{a_1,a_2\},b\in\{b_1,b_2\}} \left(\frac{\sigma(a,b)}{P_L(a|h_s)P_L(b|h_s)} \right. \\
&\qquad \left. \sum_{\langle \mathbf{d}=(a,b,d_3,\ldots,d_k),n\rangle \in \mathbf{D}} \frac{n\ P_L(h_s|\mathbf{d})}{|\mathbf{D}_s|} \right) \\
&= |\mathbf{D}_s| \sum_{a\in\{a_1,a_2\},b\in\{b_1,b_2\}} \sigma(a,b) \frac{P_{\mathbf{D}_s}(a,b)}{P_L(a|h_s)P_L(b|h_s)} \qquad (4.45)
\end{aligned}$$

Because of Inequality 4.24, $\frac{\partial LL(\mathbf{D}|L^*)}{\partial \epsilon}(0) \neq 0$. If $\frac{\partial LL(\mathbf{D}|L^*)}{\partial \epsilon}(0) > 0$, we can make $LL(\mathbf{D}|L^*) > LL(\mathbf{D}|L)$ by taking ϵ small enough. Otherwise, if $\frac{\partial LL(\mathbf{D}|L^*)}{\partial \epsilon}(0) < 0$, we can make $LL(\mathbf{D}|L^*) > LL(\mathbf{D}|L)$ by changing the sign before $\sqrt{\epsilon}$ in (4.30), (4.31), (4.34), (4.35) (then $\sigma(a,b)$ changes to $-\sigma(a,b)$), and taking ϵ small enough.

So, we have proved the theorem under the assumption (made in the beginning) that $\forall i : P_{\mathbf{D}_s}(D_i) = P_L(D_i|h_s)$. Now assume that $\exists i' : P_{\mathbf{D}_s}(D_{i'}) \neq P_L(D_{i'}|h_s)$. For convenience, we will assume that $i' = 1$ and denote $A = D_1$. Then there exist states a_1, a_2 of A such that

$$P_{\mathbf{D}_s}(a_1) > P_L(a_1|h_s), \quad P_{\mathbf{D}_s}(a_2) < P_L(a_2|h_s) \qquad (4.46)$$

(for these states to be exactly a_1 and a_2, we may need to relabel the states of A).

Let us produce model L^* by splitting component h_s from L into components h_s and h_{m+1} from L^*. Set $P_{L^*}(D_i|h_s)$ and $P_{L^*}(D_i|h_{m+1})$ in the following way:

$$\begin{aligned} P_{L^*}(D_i|h_s) = P_{L^*}(D_i|h_{m+1}) &= P_L(D_i|h_s), \forall i > 1 && (4.47)\\ P_{L^*}(a|h_s) = P_{L^*}(a|h_{m+1}) &= P_L(a|h_s), \forall a \in dom(A) \setminus \{a_1, a_2\} && (4.48)\\ P_{L^*}(a_1|h_s) = P_{L^*}(a_1|h_{m+1}) &= P_L(a_1|h_s) + \epsilon && (4.49)\\ P_{L^*}(a_2|h_s) = P_{L^*}(a_2|h_{m+1}) &= P_L(a_2|h_s) - \epsilon && (4.50) \end{aligned}$$

where $\epsilon \in \mathbb{R}$.

Now the proof proceeds as in the previous pages. If we denote $\mathbf{D}' = \{\langle \mathbf{d}, n\rangle \in \mathbf{D} : d_1 \in \{a_1, a_2\} \text{ in } \mathbf{d}\}$, then for $\langle \mathbf{d}, n\rangle \notin \mathbf{D}'$ we get $P_{L^*}(\mathbf{d}) = P_L(\mathbf{d})$ and for $\mathbf{d} = (d_1, \ldots, d_k)$ from $\mathbf{D}'$ we get

$$P_{L^*}(\mathbf{d}) = P_L(\mathbf{d}) + P_L(h_s) \prod_{i=2}^{k} P_L(D_i = d_i|h_s)\ \sigma(d_1)\ \epsilon \qquad (4.51)$$

where $\sigma(a) = \begin{cases} 1, & \text{if } a = a_1 \\ -1, & \text{if } a = a_2 \end{cases}$. So, we have $LL(\mathbf{D}|L^*) = LL(\mathbf{D}|L)$ for $\epsilon = 0$, and similarly as in (4.45)

$$\begin{aligned} \frac{\partial LL(\mathbf{D}|L^*)}{\partial \epsilon}(0) &= |\mathbf{D}_s| \sum_{a \in \{a_1, a_2\}} \sigma(a) \frac{P_{\mathbf{D}_s}(a)}{P_L(a|h_s)} \\ &= |\mathbf{D}_s| \left(\frac{P_{\mathbf{D}_s}(a_1)}{P_L(a_1|h_s)} - \frac{P_{\mathbf{D}_s}(a_2)}{P_L(a_2|h_s)} \right) \qquad (4.52) \end{aligned}$$

Because of (4.46), $\frac{\partial LL(\mathbf{D}|L^*)}{\partial \epsilon}(0) > 0$, and we can make $LL(\mathbf{D}|L^*) > LL(\mathbf{D}|L)$ by taking ϵ small enough. ■

In our proof, the parameter ϵ approaches 0. That is, it is possible to change the conditional probabilities $P_L(D_i|h_s)$ only slightly when splitting and still increase the log-likelihood of $\mathbf{D}$.

We have proved that with probability 1 it is possible to increase the log-likelihood of $\mathbf{D}$ by splitting a component in L. However, we believe that a stronger statement, saying that it is always possible to increase the log-likelihood by splitting (if the log-likelihood is not already the maximum possible, of course), is also true. Probability 1 has been introduced because in our proof we consider only pairs of variables when splitting. To prove a stronger statement, we would need to consider all the k observed variables, which makes formulas much more complicated.

4.3.3 Implications for Model Selection

We have shown that both component introduction and component splitting allow to increase the log-likelihood of data $\mathbf{D}$ (provided that the log-likelihood is not already maximum). However, for model selection the log-likelihood alone is not used because that would lead to overfitting. Instead, a *penalised likelihood* score is often used. Generally, it evaluates model M by computing

$$score(M) = LL(\mathbf{D}|M) - penalty(M, \mathbf{D}) \tag{4.53}$$

where $penalty(M, \mathbf{D})$ is a penalty for the complexity of M. Laplace, BIC, and Cheeseman-Stutz approximations, mentioned in Section 3.1.3, are the examples of a penalised likelihood score. So, for the BIC score

$$penalty(M, \mathbf{D}) = \frac{dim(M)}{2} \ln |\mathbf{D}| \tag{4.54}$$

and for the Cheeseman-Stutz score

$$penalty(M, \mathbf{D}) = LL(\mathbf{D}'|M) - \log P(\mathbf{D}'|\mathbf{m}) \tag{4.55}$$

where $\mathbf{m}$ is the structure of model M and $\mathbf{D}'$ is obtained by completing $\mathbf{D}$ using model M.

How do then component introduction and component splitting operations look in respect of a penalised likelihood score? Even when we increase $LL(\mathbf{D}|M)$ by performing component introduction or component splitting in M, we may lose more in the penalty term $penalty(M, \mathbf{D})$ than gain in the log-likelihood term $LL(\mathbf{D}|M)$. So, $score(M)$ may decrease. However, the log-likelihood term dominates over the penalty term as the size $|\mathbf{D}|$ increases. For example, in the BIC score with fixed model M and fixed probability distribution $P_{\mathbf{D}}$, the log-likelihood term decreases linearly while the minus penalty term decreases only logarithmically as $|\mathbf{D}|$ increases. So, for $\mathbf{D}$ having size big enough both component introduction and component splitting allow to increase a penalised likelihood score.

4.4 Implementation

In this section we discuss our implementation of cardinality increase for LC models.

4.4.1 Component Introduction vs. Component Splitting

First we have to choose whether to use the component introduction or component splitting approach. As we have just shown, both of them have good

theoretical properties. Another concern is their practical performance. For continuous data, the algorithm of Verbeek et al. (2003), based on component splitting, performed better than the algorithm of Vlassis and Likas (2002), based on component introduction. Also, in our initial experiments of LC model learning, component splitting performed better than component introduction (with $p = 0.01$ and uniform $P_{L^*}(D_i|h_{m+1})$ – see Definition 4.1).

The usefulness of each approach may also depend on particular data. Assume that the two-dimensional training data $\mathbf{D}$ looks as in Figure 4.1.[4] Assume that the two component model, shown in Figure 4.2, has been

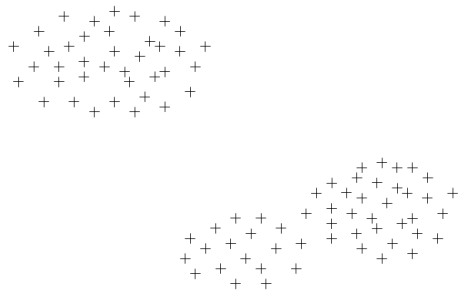

Figure 4.1: Two-dimensional continuous training data.

learned. Here components are shown as ellipses. Each ellipse indicates a two-dimensional Gaussian distribution, with the large part (let us say, 95%) of the probability mass being inside the ellipse.[5] The best three-component model should be similar to the one depicted in Figure 4.3. In this case, when moving from the two–component to a three–component model, component splitting seems to be more appropriate than component introduction. Splitting the lower component, as shown in Figure 4.4, and running the EM algorithm afterwards should lead to the model from Figure 4.3.

On the other hand, the training data may look as in Figure 4.5 and the two–component model as in Figure 4.6. The best three-component model should be similar to the one depicted in Figure 4.7. In this case, when moving from the two–component to a three–component model, component introduc-

[4]Even though we work with categorical data, for the sake of visualisation we use continuous data here.

[5]So, instances which are inside of one of the ellipses belong to the corresponding component with much higher probability than to another component. In Figure 4.6, three instances which are outside of the ellipses belong to the two components with similar probabilities.

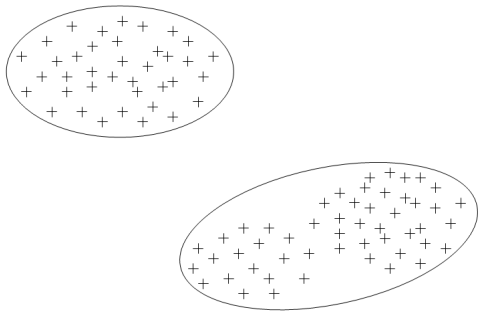

Figure 4.2: The two–component model for the training data.

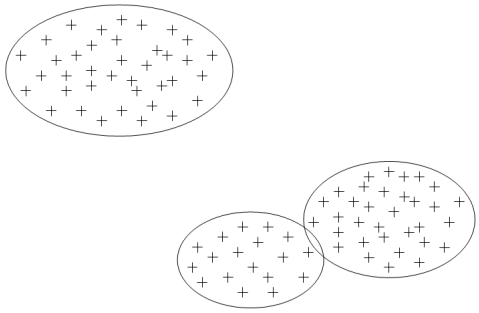

Figure 4.3: The three–component model for the training data.

tion seems to be more appropriate than component splitting. It is very unlikely that splitting one of the components from Figure 4.6 and running the EM algorithm afterwards would lead to the model from Figure 4.7.

Even though we have considered only small artificial examples here, it seems that neither component introduction nor splitting is always to prefer. Because of the experimental results, mentioned in the beginning of this section, we choose use component splitting.

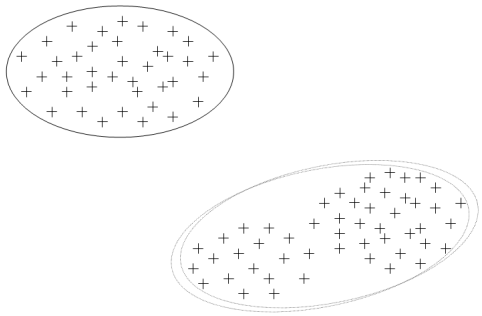

Figure 4.4: Splitting of a component.

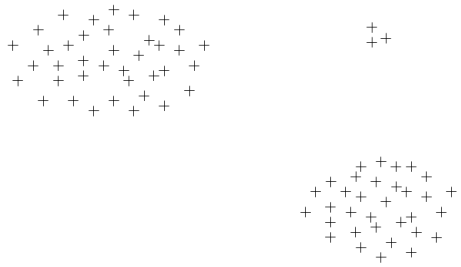

Figure 4.5: Another two-dimensional continuous training data.

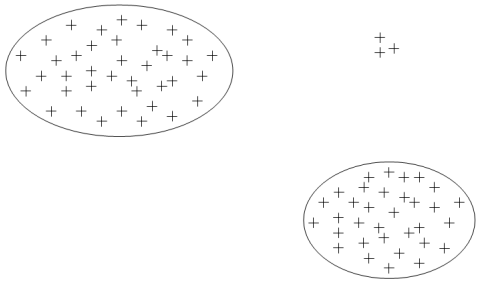

Figure 4.6: The two–component model for the training data.

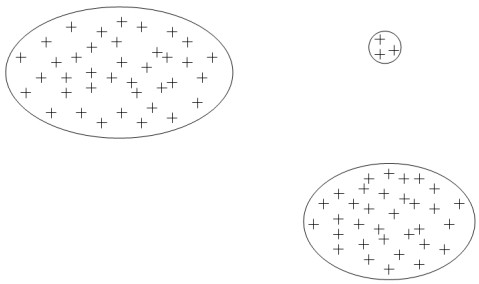

Figure 4.7: The three–component model for the training data.

4.4.2 Component Splitting in Detail

In this section we give a detailed description of our implementation of component splitting for LC models.

In Definition 4.2, where we described component splitting formally, $P_{L^*}(D_i|h_s^1)$ and $P_{L^*}(D_i|h_s^2)$ $(i = 1, \ldots, k)$ are not specified exactly, but only said to be close to $P_L(D_i|h_s)$. One way of constructing $P_{L^*}(D_i|h_s^1)$ and $P_{L^*}(D_i|h_s^2)$ could be making them equal to $P_{L^*}(D_i|h_s)$ and $P_{L^*}(D_i|h_{m+1})$ from the proof of Theorem 4.2. However, this approach makes changes only for two observed variables when performing a split, while in practice we would like to increase the log-likelihood as much as possible with a single split. So, more reasonable is to try changing $P_L(D_i|h_s)$ for all variables D_i. If changing $P_L(D_i|h_s)$ is not necessary for some D_i, the EM algorithm should move it to the original value. So, we make small random perturbations in all $P_L(D_i|h_s)$.[6] More precisely, component h_s is split by making $P_{L^*}(D_i|h_s^1) - P_L(D_i|h_s) = \vec{r}_i$ and $P_{L^*}(D_i|h_s^2) - P_L(D_i|h_s) = -\vec{r}_i$. Each element of $\vec{r}_i$ is a random number from $[-p; p]$, where $p \in \mathbb{R}$ is a small positive parameter. Random numbers from $[-p; p]$ are sampled from the uniform distribution. If according to vectors $\vec{r}_i$ some conditional probabilities from L^* would be outside of $(0; 1)$, those probabilities are set to be near the edge of this interval. Finally, $P_{L^*}(D_i|h_s^1)$ and $P_{L^*}(D_i|h_s^2)$ are normalised.

When trying to increase the number of components in model L, we split each component of L in the way just described. This way we obtain a set of models $\mathcal{M}$ that contains as many models as there are components in L and each model from $\mathcal{M}$ has one component more than L. Then we optimise the parameters of models from $\mathcal{M}$ and select the best model among them by running the multiple restart EM, mentioned in Section 3.2. All the models from $\mathcal{M}$ are used as the initial starting points for the multiple restart EM, which at the end obtains the final model having one more component than L.

As an alternative to random splitting, we have also tried the following deterministic approaches of splitting. Here we assume that all the observed variables are binary.

1. Having $\vec{r}_i = \begin{pmatrix} -p \\ p \end{pmatrix}$ for odd i and $\vec{r}_i = \begin{pmatrix} p \\ -p \end{pmatrix}$ for even i.

2. Searching for the optimal (i.e., giving the highest increase in the log-likelihood) splitting where each $\vec{r}_i$ is either $\begin{pmatrix} -p \\ p \end{pmatrix}$ or $\begin{pmatrix} p \\ -p \end{pmatrix}$. Since

[6] This way, we follow the component splitting approach for continuous data of Ueda et al. (2000), described in Section 3.3.

there are 2 ways of choosing each $\vec{r}_i$, there are 2^k possible ways of splitting for k observed variables. Considering all the 2^k possibilities can be too expensive computationally, so we use a greedy search that considers only $O(k)$ possibilities. First, order variables D_i according to how far $P_L(D_i|h_s)$ is from $\begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix}$. Then go through all the variables D_i by starting from those for which $P_L(D_i|h_s)$ is closest to $\begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix}$ and setting $\vec{r}_i$ in the following way. For the first D_i, set $\vec{r}_i = \begin{pmatrix} -p \\ p \end{pmatrix}$. For each next D_i, choose between $\vec{r}_i = \begin{pmatrix} -p \\ p \end{pmatrix}$ and $\vec{r}_i = \begin{pmatrix} p \\ -p \end{pmatrix}$ by performing the following computations:

- Produce two-component models L_1 and L_2, both having hidden states h_s^1 and h_s^2, and the following parameters:
  - $P_{L_u}(h_s^1) = P_{L_u}(h_s^2) = 0.5$,
  - $P_{L_u}(D_i|h_s^1) = P_L(D_i|h_s) + \vec{r}_i$ and $P_{L_u}(D_i|h_s^2) = P_L(D_i|h_s) - \vec{r}_i$, for those D_i where $\vec{r}_i$ has already been chosen,
  - $P_{L_1}(D_i|h_s^1) = P_L(D_i|h_s) + \begin{pmatrix} -p \\ p \end{pmatrix}$,
    $P_{L_1}(D_i|h_s^2) = P_L(D_i|h_s) - \begin{pmatrix} -p \\ p \end{pmatrix}$,
    $P_{L_2}(D_i|h_s^1) = P_L(D_i|h_s) + \begin{pmatrix} p \\ -p \end{pmatrix}$,
    $P_{L_2}(D_i|h_s^2) = P_L(D_i|h_s) - \begin{pmatrix} p \\ -p \end{pmatrix}$,
    for the current D_i,
  - $P_{L_u}(D_i|h_s^1) = P_{L_u}(D_i|h_s^2) = P_L(D_i|h_s)$, for all the rest D_i.
- If $LL(\mathbf{D}_s|L_1) > LL(\mathbf{D}_s|L_2)$, set $\vec{r}_i = \begin{pmatrix} -p \\ p \end{pmatrix}$. Otherwise, set $\vec{r}_i = \begin{pmatrix} p \\ -p \end{pmatrix}$.

3. Splitting component h_s in the direction of highest variance of data set $\mathbf{D}_s$. Since all the observed variables are binary, we treat them as having states 0 and 1. We compute the covariance matrix for data set $\mathbf{D}_s$ and the leading eigenvector $\vec{e}$ of this matrix. Then we set $\vec{r}_i = \begin{pmatrix} -\vec{e}_i p \\ \vec{e}_i p \end{pmatrix}$, $i = 1, \ldots, k$.

For all these approaches, in the same way as for random splitting, we make sure that conditional probabilities from the new model are inside $(0; 1)$ and normalise them, if necessary. In our initial tests, none of these three approaches was clearly better than random splitting. Since random splitting is simple and works for non-binary observed variables as well, we will use random splitting further on.

4.5 Experiments

In this section we present the experiments where we compare the approach based on component splitting and the standard approach for learning LC models.

4.5.1 Algorithms

In this section we describe the algorithms that are tested in the experiments. These algorithms are formulated as anytime algorithms – that is, their quality of results improves gradually as computation time increases (Zilberstein, 1996). The algorithms take training data $\mathbf{D}$ as an input. They try to find an LC model that is the best according to the function *score*, which can be any penalised likelihood scoring function. The algorithms can run arbitrarily long, and they always maintain the best LC model found so far.

The first algorithm is called *Splitting* and its pseudocode is given below. It starts with the one–component model (which can be seen as a model containing only observed variables and having no edges), for which the maximum likelihood parameters are deterministically computed from $\mathbf{D}$. Then the algorithm repeatedly tries to increase the score of a model by splitting a component. In step 3 the algorithm produces a set of models which in step 4 are used as starting points for EM, as described in Section 4.4.2. If model L' obtained in step 4 has a higher score than the current model L, L' is taken as the new current model. This way the cardinality of a hidden variable is increased. Otherwise, the current model L remains the same. Our algorithm allows executing steps 3–7 many times for the same model L, because in each iteration different set of models $\mathcal{M}$ is obtained by splitting components randomly. Please also note that our algorithm only allows to increase the number of components.

The second algorithm is called *Standard*, because it uses standard starting points for EM. Its pseudocode is similar to that of *Splitting* and is given below. The difference from *Splitting* is that in step 3 only one model having one component more than model L is produced. The parameters of this new

Procedure 4.1 *Splitting*(**D**)

1: Let L be the LC model with one component and with maximum likelihood parameters computed from **D**.
2: **loop**
3: Produce a set of models $\mathcal{M}$ by randomly splitting each component of L.
4: Obtain model L' by running the multiple restart EM with $\mathcal{M}$ as starting points.
5: **if** $score(L') > score(L)$ **then**
6: $L \leftarrow L'$.
7: **end if**
8: **end loop**

model L' are sampled from the uniform distribution. In step 4 the parameters are optimised by running the standard EM algorithm.

Procedure 4.2 *Standard*(**D**)

1: Let L be the LC model with one component and with maximum likelihood parameters computed from **D**.
2: **loop**
3: Let L' be a randomly parameterised model having one component more than L.
4: Optimise the parameters of L' by running the EM algorithm.
5: **if** $score(L') > score(L)$ **then**
6: $L \leftarrow L'$.
7: **end if**
8: **end loop**

Both *Splitting* and *Standard* try to determine the optimal number of components of an LC model. They always try to increase the number of components by one. However, this restriction of increasing by one is not necessary for the approach that uses standard starting points for EM. Since the parameters of each new model are learned from scratch, there may be more efficient ways of determining the number of components than it is done in *Standard*. That is why we also compare the parameter reusing and standard approaches when the latter has an unfair advantage of knowing the number of components from the start. The algorithm called *StandardFixed* uses standard starting points for EM and estimates model parameters when the number of components is fixed. Its pseudocode is given below. It repeatedly tries to improve model parameters by running the multiple restart EM from

random starting points. Constant N_0 is the initial number of starting points for EM. In step 4 the parameters of models from $\mathcal{M}$ are sampled from the uniform distribution. In each iteration the number of starting points for EM is doubled.

Procedure 4.3 *StandardFixed*$(\mathbf{D}, m)$

1: Let L be an LC model with m components and random parameters.
2: $N \leftarrow N_0$.
3: **loop**
4: Produce a set $\mathcal{M}$ of N models, where each model has the same number of components as L and random parameters.
5: Obtain model L' by running the multiple restart EM with $\mathcal{M}$ as starting points.
6: **if** $score(L') > score(L)$ **then**
7: $L \leftarrow L'$.
8: **end if**
9: $N \leftarrow 2N$.
10: **end loop**

4.5.2 Setup of Experiments

The setup of experiments is the following. As a model scoring function, we use the Cheeseman-Stutz score. The EM algorithm searches for MAP parameters of a model. Prior distributions of model parameters are Dirichlet with its parameters equal to 1 (that is, uniform in the model parameter space), as described in Chickering and Heckerman (1997), page 196. We have chosen this setup because it has been reported by Chickering and Heckerman (1997) to be more suitable than other scores for an LC model selection. We set the algorithm parameter p from Section 4.4.2 to 0.1. N_0 in *StandardFixed* is 1. Everywhere in the multiple restart EM, $\frac{1}{q}$th of the models giving the highest log-likelihood are retained after 1, 3, 7, 15, ... iterations of EM, until only one best model is left. After that the EM algorithm is run until either the difference between successive values of log-likelihood is less than 0.01 or 200 iterations (including earlier iterations) are reached.[7] In the experiments from this chapter $q = 2$. In case of the standard EM, it is simply run until either the difference between successive values of log-likelihood is less than 0.01 or 200 iterations are reached. We run the experiments on a Java^{TM} 2 platform, 2.8 GHz Intel(R) processor.

[7] If the number of starting points is very large, more than 200 iterations may be performed before one best model is left.

First, we produced data sets from the Reuters-21578 text categorisation test collection (Lewis, 1997). The data was preprocessed as described in Appendix A. All the observed variables are binary and data sets over the observed variables are complete.

Next, from the UCI Machine Learning Repository (Blake and Merz, 1998) we obtained *Solar Flare* (refered to as *Solar*), *Large Soybean* (refered to as *Soybean*), and *Mushroom* classification data sets and discarded the class information. Here the cardinality of the observed variables ranges from 2 to 10. In *Soybean* and *Mushroom*, data sets over the observed variables contain missing data.

These preprocessed text categorisation and UCI repository data sets are used as an input data $\mathbf{D}$ for the algorithms described in Section 4.5.1. Since all three tested algorithms are stochastic, a single algorithm is run 5 times on a single data set. First, we run the algorithm *Standard* until the mean score of model L stops to increase fast. More precisely, for each data set we had that at stopping time t $\frac{score(t)-score(2t/3)}{score(2t/3)-score(t/3)} < 0.1$, where $score(u)$ is the mean score for *Standard* at time u. After that we run the algorithm *Splitting* for the same time as *Standard* on each data set. For data sets where the mean score of the final model found by *Standard* and by *Splitting* are significantly different we run the algorithm *StandardFixed*. The parameter m of *StandardFixed* is the number of components in the best scoring model found by *Splitting* for a given data set. This way, *StandardFixed* is given the initial advantage over *Splitting*. We run *StandardFixed* for the same time as *Splitting* on a given data set.

4.5.3 Results

The results for *Standard* and *Splitting* are summarised in Table 4.2. The first 9 are text categorisation data sets, and the last 3 are from the UCI repository. For each data set, the following is shown: the data set size, the number of observed variables, the running time of an algorithm (in minutes), the mean score (with a 95% confidence interval for the mean) and the average number of components of the final model for both algorithms, and the time ratio indicating how many times *Splitting* is faster than *Standard*.[8] We say that one algorithm finds *better* scoring models than another algorithm if the difference in the mean score values is higher than 1. The names of data sets where *Splitting* found better scoring models than *Standard* are written

[8]Time ratio is computed as $\frac{time_{Standard}}{time_{Splitting}}$, where $time_{Algorithm}$ is the time when the mean score of the model found by *Algorithm* is equal to the mean score of the final model found by *Standard* (for Crude20 data set, the final model found by *Splitting*).

in bold. We say that one algorithm finds *significantly better* scoring models than another algorithm if intervals for the mean do not overlap and the difference in the mean score values is higher than 1. For the score values, bold text indicates a significantly better (and underlined – a significantly worse) score when comparing *Splitting* with *Standard*.

Data set	Size	Vari-ables	Time	*Standard*		*Splitting*		Time ratio
				score	$\|H\|$	score	$\|H\|$	
Corn10	10787	10	4	-10340.1 ± 1.7	8	-10339.1 ± 0.3	8	5.7
Crude10	10787	10	5	-10491.8 ± 0.7	6	-10491.3 ± 0.7	6	10.0
Earn10	10787	10	5	-26387.0 ± 7.1	11	-26380.7 ± 0.0	10	5.3
Corn20	10787	20	21	-23527.4 ± 2.4	10	-23527.2 ± 0.7	10	2.9
Crude20	10787	20	50	-23799.3 ± 1.0	9	-23799.4 ± 0.5	9	9.7
Earn20	10787	20	167	-46784.1 ± 13.5	15	-46767.2 ± 5.8	17	3.1
Corn30	10787	30	67	-40421.8 ± 8.2	11	**-40409.9 ± 0.6**	11	3.2
Crude30	10787	30	48	-28949.3 ± 10.3	9	-28943.4 ± 2.2	10	3.5
Earn30	10787	30	186	-74112.5 ± 71.4	14	**-73922.3 ± 0.3**	18	4.1
Solar	1389	10	1	-8343.2 ± 1.0	5	-8342.3 ± 0.5	5	2.3
Soybean	683	35	95	-11180.7 ± 29.2	14	**-11072.4 ± 35.8**	17	12.6
Mushroom	8124	22	280	-83858.7 ± 1400.2	15	**-80935.0 ± 24.8**	21	3.0

Table 4.2: Results for *Standard* and *Splitting*.

As mentioned in the previous section, we run algorithm *StandardFixed* for data sets where the mean scores for *Standard* and for *Splitting* are significantly different. The results from these tests are summarised in Table 4.3. For convenience, we also include score values for *Splitting* from Table 4.2. As indicated in the previous section, $|H|$ for *StandardFixed* for a given data set is always the same and is equal to the number of components in the best scoring model found by *Splitting*. The same way as in Table 4.2, the time ratio field indicates how many times *Splitting* is faster than *StandardFixed*.

Data set	*StandardFixed*		*Splitting*	Time ratio
	score	$\|H\|$	score	
Corn30	-40431.2 ± 10.3	11	**-40409.9 ± 0.6**	5.4
Earn30	-73994.6 ± 48.9	18	**-73922.3 ± 0.3**	1.4
Soybean	-11113.5 ± 11.1	15	-11072.4 ± 35.8	5.0
Mushroom	-81796.7 ± 241.0	21	**-80935.0 ± 24.8**	2.4

Table 4.3: Results for *StandardFixed* and *Splitting*.

Figures 4.8 and 4.9 show how the mean score changes during time for two data sets (bars at the end indicate 95% confidence intervals for the mean). Graphs for the other data sets look similar.

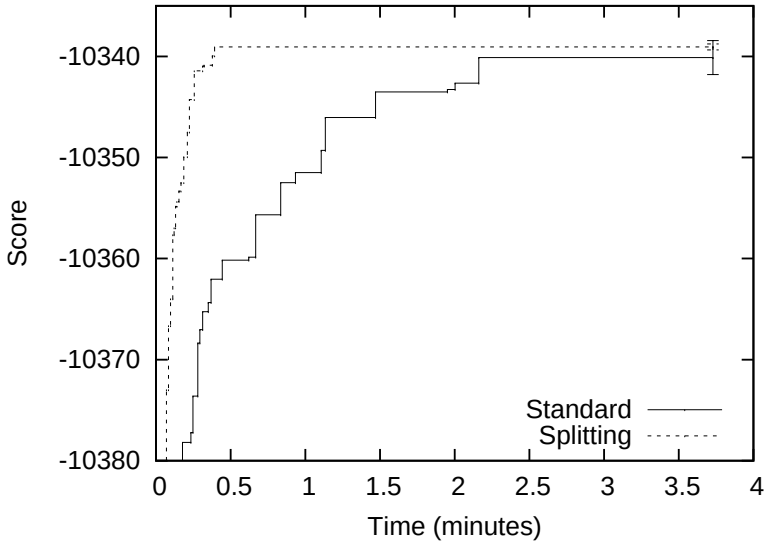

Figure 4.8: Score change during time for Corn10 data set.

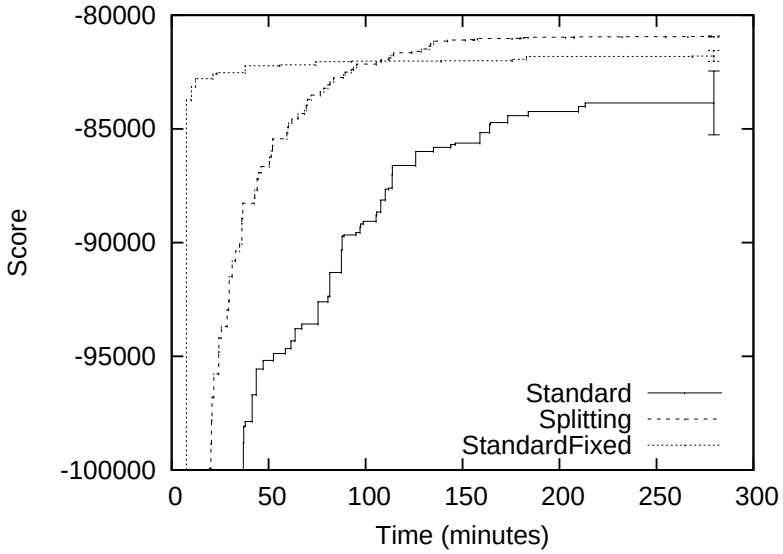

Figure 4.9: Score change during time for Mushroom data set.

4.5.4 Discussion

As seen from Table 4.2, *Splitting* found better scoring models than *Standard* for 7 out of 12 data sets. For other 5 data sets, the scores of the models found by *Standard* and by *Splitting* are very similar. For 4 out of those 7 data sets, *Splitting* found significantly better scoring models than *Standard*. The general pattern is that *Splitting* outperforms *Standard* more as the number of components ($|H|$) increases. The results from Table 4.3 show that even when the standard approach was given the advantage of knowing the number of components, *Splitting* was better on those many-component data sets. The most likely reason for such results is that the more components an LC model has, the lower the chances that a random starting point for the EM algorithm will be a good starting point. Uebersax (2000) also points out that the main factor contributing to local maximum solutions seems to be the number of components. The time ratio field in both tables indicates that *Splitting* is faster than standard approaches.

As seen from Figures 4.8 and 4.9, models found by *Splitting* at any moment have higher score than models found by *Standard*. Obviously, *StandardFixed* finds higher scoring models than *Splitting* in the beginning, but later, when *Splitting* reaches a high number of components, it becomes better than *StandardFixed* for those 4 data sets where *StandardFixed* has been tested.

From Tables 4.2 and 4.3 we can also see that the variance of the score for *Splitting* is lower than for the algorithms which use standard starting points for EM (except for "Soybean" and "Crude10" data sets). A possible explanation is that there is less randomness in *Splitting*: only the parameters of two new components obtained by splitting are random (and they are still close to the parameters of the component that has been split), while in the other two algorithms all the initial parameters of a model are taken randomly.

So, by using the fact that LC models with different number of components for the same data have similar parameters, we are able to construct better than standard starting points for the EM algorithm. However, our algorithm *Splitting* may still be improved. Since *Splitting* only allows to increase the number of components, problems can appear when non-optimal parameters for some number of components m are found. If m is the optimal number of components, a model with m components should have a higher score than a model with any other number of components. However, if parameters of an m–component model are not optimal, *Splitting* may find an $m + 1$–component model with a higher score. This model would be taken as the new best model, and there is no way of going back to m components. This way, *Splitting* overestimates the number of components. This has happened

in our experiments: for some data sets, some final models found by *Splitting* have more components but lower score than final models found by *Splitting* in other runs for the same data set. That is why we believe that an operation for decreasing the number of components (for example, component merging) should be introduced.

Chapter 5

Decreasing the Cardinality

In this chapter we discuss approaches for decreasing the cardinality of a hidden variable while reusing the parameters. Similarly as with the cardinality increase, we do this for categorical data, in the context of latent class (LC) models. Part of this chapter appeared in Karčiauskas et al. (2004b).

First we give an overview of how the cardinality can be decreased by reusing the parameters. Then we define formally how to do it. After that, we discuss the theoretical properties of the proposed approaches. Then we discuss the implementation of the cardinality decrease.

5.1 Overview

In Section 4.5.4 we have noticed the need of an operation for decreasing the number of components while reusing the parameters. By making analogies to the cardinality increase, one can think about two approaches for obtaining an m-1–component model from an m–component model.

The first approach is opposite to the component introduction and we call it *component removal*. Here an m-1–component model is initialised by removing one component from an m–component model. All the other components are kept unchanged, only their weights are scaled up. Then the parameters of the m-1–component model can be optimised by running for example the EM algorithm. The algorithm of Figueiredo and Jain (2002) repeatedly removes components, as described in Section 3.3.

The second approach is opposite to the component splitting and we call it *component merging*. Here an m-1–component model is initialised by merging two components from an m–component model into one component. The other $m - 2$ "old" components are kept unchanged. The parameters of the m-1–component model can then be optimised as in

the first approach. As described in Section 3.3, Ueda et al. (2000) and Ueda and Ghahramani (2002) use this approach for continuous data. The algorithm of Elidan and Friedman (2001), described in Section 3.1.2, has similarities to this approach and works with categorical data. The biggest difference is that their algorithm works with "hard" assignments of each instance to a particular state of a hidden variable while this approach works with "soft" assignments (i.e., an instance belongs to each state of a hidden variable with some probability).

Next we will investigate these two approaches in more detail.

5.2 Definitions

Here we give formal definitions of component removal and component merging operations for LC models.

Definition 5.1 *We say that model L^* is obtained from model L by* removing *a component h_r if L^* contains all the components from L except h_r with their marginal probabilities scaled up. More formally, if L contains components $h_1, \ldots, h_m$, then L^* contains components $h_1, \ldots, h_{r-1}, h_{r+1}, \ldots, h_m$ and the following is true:*

- $P_{L^*}(h_l) = \frac{1}{1-P_L(h_r)} P_L(h_l)$, $l = 1, \ldots, m,\ l \neq r$,
- $P_{L^*}(D_i|h_l) = P_L(D_i|h_l)$, $l = 1, \ldots, m,\ l \neq r,\ i = 1, \ldots, k$.

Definition 5.2 *We say that model L^* is obtained from model L by* merging *components h_s^1 and h_s^2 into h_s if h_s in L^* is a weighted average of h_s^1 and h_s^2 in L, and all the other components are identical in both models. More formally, if L contains components $h_1, \ldots, h_{m-2}, h_s^1, h_s^2$, then L^* contains components $h_1, \ldots, h_{m-2}, h_s$ and the following is true:*

- $P_{L^*}(h_l) = P_L(h_l),\ P_{L^*}(D_i|h_l) = P_L(D_i|h_l)$,
  $l = 1, \ldots, m-2,\ i = 1, \ldots, k$,
- $P_{L^*}(h_s) = P_L(h_s^1) + P_L(h_s^2)$,
- $P_{L^*}(D_i|h_s) = \frac{P_L(h_s^1)}{P_{L^*}(h_s)} P_L(D_i|h_s^1) + \frac{P_L(h_s^2)}{P_{L^*}(h_s)} P_L(D_i|h_s^2),\ i = 1, \ldots, k$.

5.3 Theoretical Properties

Having defined the component removal and component merging operations, we now look at their theoretical properties. For convenience, we assume that training data $\mathbf{D}$ over the observed variables is complete.

First, let us see what properties we would like these cardinality decrease operations to have. We would use these operations when a model contains too many components. That is, when decreasing the cardinality, we expect that training data $\mathbf{D}$ can be described equally well (or almost equally well) by a model with one component less. If $\mathbf{D}$ is described perfectly by our current m–component model and it is possible to describe $\mathbf{D}$ perfectly by an m-1–component model as well, then we would obviously like a cardinality decrease operation to be able to obtain this m-1–component model from our current model. Immediately, we see that the component removal does not have this property, except for trivial cases when the component being removed has a marginal probability 0 or it is the combination of all the other components. For the component merging, the situation is more interesting. We analyse it below.

5.3.1 Identifiability

In our discussion of the theoretical properties of the component merging, we will use the concept of LC model identifiability (Goodman, 1974).

Definition 5.3 *The model structure* $\mathbf{m}$ *is* identifiable *if for almost every data* $\mathbf{D}$ *the maximum likelihood parameters* θ_{ML} *for* $\mathbf{m}$ *are unique (here it is assumed that any two parameterisations that differ only in the ordering of components are the same).*[1]

So, if $\mathcal{D}$ denotes the set of all the possible $\mathbf{D}$ over the observed variables from an identifiable structure $\mathbf{m}$ and $\mathcal{D}_1$ denotes the set of those $\mathbf{D}$ from $\mathcal{D}$ for which θ_{ML} for $\mathbf{m}$ are unique, then $\mathcal{D}_1$ has measure 1 in $\mathcal{D}$. We will denote the measure zero set $\mathcal{D} \setminus \mathcal{D}_1$ by $\mathcal{D}_0$.

Let $dc = |D_1| \cdot \ldots \cdot |D_k| - 1$ denote the complete dimension of $D_1 \times \ldots \times D_k$. If the parameters of a model $L = (\mathbf{m}, \theta_{ML})$ are those of maximum likelihood, then $P_L(D_1, \ldots, D_k) = P_{\mathbf{D}}(D_1, \ldots, D_k)$. So, estimation of θ_{ML} can be seen as solving dc equations with $dim(\mathbf{m})$ unknowns. If $dim(\mathbf{m}) > dc$, then θ_{ML} can not be uniquely determined from $\mathbf{D}$, which means that structure $\mathbf{m}$ is not identifiable. For example (Uebersax, 2001), the structure 2:2,2,2 is

[1] We would like to remind that the weights of instances in data $\mathbf{D}$ are real numbers, as defined in Chapter 2. That is, each $\mathbf{D}$ over $D_1, \ldots, D_k$ defines some $P_{\mathbf{D}}(D_1, \ldots, D_k)$.

identifiable: there are 7 independent parameters in an LC model that has this structure, and the dimension of $D_1 \times D_2 \times D_3$ is also 7. However, the structure 3:2,2,2 is not identifiable: for any given $\mathbf{D}$, there exist infinitely many different maximum likelihood parameterisations.

Even though $dim(\mathbf{m}) \leq dc$ very often indicates that $\mathbf{m}$ is identifiable, in general this is not a sufficient condition. For example, for the structure $\mathbf{m}$ = 3:2,2,2,2, $dim(\mathbf{m}) = 14$ and $dc = 15$. However, this $\mathbf{m}$ is not identifiable (Kočka and Zhang, 2002).

It turns out (Goodman, 1974; Geiger et al., 1996) that $\mathbf{m}$ is identifiable if and only if the effective dimension of $\mathbf{m}$, described in Section 3.1.3, is equal to its standard dimension $dim(\mathbf{m})$.

We will also use the concept of identifiability of a parameterised model.

Definition 5.4 *The model $M = (\mathbf{m}, \theta)$ is* identifiable *if for each $\theta' \neq \theta$ the models M and $M' = (\mathbf{m}, \theta')$ define different joint probability distributions over the observed variables, i.e. $P_M(D_1, \ldots, D_k) \neq P_{M'}(D_1, \ldots, D_k)$ (again, it is assumed that $\theta = \theta'$ if they differ only in the ordering of components).*

That is, a model is identifiable if its parameters are unique.

5.3.2 Component Merging for Identifiable Structures

First, we performed the initial experiments to check the properties of component merging. We parameterised randomly an m–component model L and generated perfectly described by L data $\mathbf{D}$ (that is, $P_L(D_1, \ldots, D_k) = P_{\mathbf{D}}(D_1, \ldots, D_k)$). Then we estimated the maximum likelihood (ML) parameters of an m+1–component model L' given $\mathbf{D}$. That is, $\mathbf{D}$ is described perfectly by L' as well. We did this for small models L, because when L gets larger it becomes computationally difficult to find ML parameters for L'.[2] Each time, we checked if it is possible to obtain model L from model L' by merging two components of L'. In some cases it was possible, and in some not. This lead us to making in Karčiauskas et al. (2004b) a conjecture about the component merging, which we give here a little reformulated.

Conjecture 5.1 *Assume that data $\mathbf{D}$ is described perfectly by an identifiable LC model L. Assume that the structure of an LC model L', having one*

[2]Even for small models, we have to use iterative methods for finding ML parameters with some precision. We have tried to find exact ML parameters with Maple^{TM}, but already for the 2:2,2,2,2 structure the number of equations and unknowns becomes too high.

component more than L, is identifiable. Then, if $\mathbf{D}$ *is described perfectly by L', model L can be obtained by merging two components of L'.*[3]

However, later we have discovered that for some models this conjecture does not hold. For example, for a model L having the 2:3,3,2 structure we obtained a model L' having the 3:3,3,2 structure and describing $\mathbf{D}$ perfectly, where L can not be obtained by merging any two states of L'. Since the structure 3:3,3,2 is identifiable, this is a counter-example to the conjecture. We obtained such a result for several different parameterisations of the 2:3,3,2 structure.

5.3.3 Component Merging when Variables are Binary

Having discovered that Conjecture 5.1 is not true in general, we restrict ourselves to investigating the case when all the observed variables are binary. First, we performed experiments similar to those of Section 5.3.2. For each randomly parameterised model L, we estimated the ML parameters of L' by running the standard EM from a random starting point (for larger L', this required running EM more than once, until the ML parameters with some precision have been found). Then we checked if it is possible to obtain model L from model L' by merging two components of L'. The results of these experiments are summarised in Table 5.1. Here a-b:kx2 indicates all the structures with k binary variables where the number of components ranges from a to b. For the models above the middle line, it is possible to obtain model L from model L' by merging some two components of L'. For the models below the middle line, it is not possible. In all these experiments, model L is identifiable. Here we did not include the trivial case when L has only one component, because then merging the two components of L' always produces L. Also, we did not test models with 7 or more observed variables, because then it becomes computationally too difficult to find the ML parameters for L'.

Based on these results, now we make the following conjecture.

Conjecture 5.2 *Assume that* $\mathbf{D}$ *is described perfectly by an identifiable LC model L with binary observed variables. Assume that for an LC model L', having one component more than L, we have $dim(L') \leq dc$. Then, if* $\mathbf{D}$ *is*

[3]Please note that even though the structure $\mathbf{m}$ of L' is identifiable, the model $L' = (\mathbf{m}, \theta)$ itself is not identifiable, because θ is just one of infinitely many ML parameterisations of $\mathbf{m}$ given $\mathbf{D}$. This is because L' has one component more than L, and so L' has too many parameters for describing $\mathbf{D}$. In other words, when considering the structure $\mathbf{m}$, the data $\mathbf{D}$ belongs to a measure zero set $\mathcal{D}_0$, introduced after Definition 5.3.

Structure of L	Structure of L'	Structure of L' identifiable	$dim(L') \leq dc$
2:4x2	3:4x2	no	yes
2-4:5x2	3-5:5x2	yes	yes
2-8:6x2	3-9:6x2	yes	yes
2:3x2	3:3x2	no	no
3:4x2	4:4x2	no	no
5:5x2	6:5x2	no	no
9:6x2	10:6x2	no	no

Table 5.1: Results of experiments about properties of component merging.

described perfectly by L', model L can be obtained by merging two components of L'.

Since for any identifiable structure $\mathbf{m}$, $dim(\mathbf{m}) \leq dc$ (see Section 5.3.1), this conjecture is a little stronger then Conjecture 5.1 for LC models with binary observed variables (in our counter-example to Conjecture 5.1, not all the observed variables were binary).

5.3.4 Summary

We have studied the properties of the component merging by experimenting with LC models of small size. We have tried to characterise model structures for which component merging works, assuming that the structure contains one component too much. First, we conjectured that it is enough for this structure to be identifiable. However, we have found an example where this condition is not sufficient. Then, for LC models with binary observed variables, we conjectured that it is enough for this structure to satisfy a condition slightly weaker than identifiability. No counter-examples to this conjecture have been found. In general, it seems that component merging works for structures where the number of observed variables k is high enough or, alternatively, the number of components m is low enough. Our conjectures are the attempts of specifying conditions for m and k under which the component merging works.

So far, we have assumed that the current model contains one component too much. In practice, it can of course differ from the true model by more than one component. Can we expect that under some conditions it is possible to arrive to the true model by repeatedly performing a pairwise component merging? We have made the following test. The same way as in Section 5.3.3,

for a randomly parameterised model L having structure 2:5x2, we estimated the ML parameters of model L' having structure 4:5x2. We repeated this several times for different parameterisations of L. In some cases, it was possible to arrive to model L by performing a pairwise component merging in L' and thus obtaining a 3–component model L'' and then performing a pairwise component merging in L''. However, in other cases model L could be obtained only by merging three components at once in L' (when more than two components are merged, they are combined into one new component the same way as h_s^1 and h_s^2 are combined in Definition 5.2). So, it seems that when the number of components is too high by more than one, a pairwise merging is not enough.

Finally, we should remind that all these results about the component merging are based on estimation of the ML parameters with some precision (and the larger the model, the lower the precision). Our judgements on whether the parameters are those of ML and whether merging some particular components leads to some particular model are based on those estimates of parameters.

5.4 Implementation

In this section we discuss our implementation of cardinality decrease for LC models. First we have to choose whether to use the component removal or component merging approach. We choose component merging because it corresponds to component splitting, which we are already using, and because it seems to have better (even though not strong) theoretical properties than component removal. As discussed in Section 5.3.4, merging more than two components at once may have better theoretical properties than a pairwise merging. However, we use a pairwise merging because it has a lower complexity and because we believe that it does not make a big difference for practical data.

When trying to decrease the number of components in model L, we merge each pair of components in the way specified in Definition 5.2. So, for an m–component model L, we obtain a set $\mathcal{M}$ that contains $\binom{m}{2}$ models, each of them having one component less than L. Then we optimise the parameters of models from $\mathcal{M}$ and select the best model among them by running the multiple restart EM with $\mathcal{M}$ as starting points. Our initial tests showed that the pair determined by the multiple restart EM is usually among the best pairs determined by a separate standard EM for each merge candidate (which is computationally much more expensive than the multiple restart EM).

Chapter 6

Learning of Latent Class Models

In this chapter we discuss learning of latent class (LC) models by combining the component splitting and component merging operations, introduced in the previous two chapters. Part of this chapter appeared in Karčiauskas et al. (2004b).

First, we introduce an operation for improving model parameters while keeping the number of components the same. Then we describe the algorithms that learn LC models by both increasing and decreasing the number of components. Finally, we present experiments comparing the performance of these algorithms.

6.1 Parameter Adjustment

In this section, we introduce an operation that tries to improve model parameters for a fixed number of components $|H|$. First, we give a motivation for such an operation.

It is possible to learn an LC model by repeatedly incrementing and decrementing $|H|$, where starting points for EM are obtained only by splitting and merging components, and it is required that any single increment or decrement of $|H|$ increases the model score. The algorithm would stop when neither incrementing nor decrementing $|H|$ increases the model score. However, the following situation can occur. The best model G has m components, and by incrementing and decrementing $|H|$ we have arrived at model L with m components that has however a lower score than G because of not optimal parameters. It can be that neither incrementing nor decrementing $|H|$ increases the score of L, and so L would be the final model. However, it can be

that by *both* incrementing and decrementing $|H|$ we obtain model L' (with m components) that has a higher score than L.

Even if the number of components in our current model L is not optimal, it may still be worth trying to *both* increment and decrement $|H|$. For example, it can happen that model L has $m-1$ components, and neither incrementing nor decrementing $|H|$ increases the score of L. It can be that by both incrementing and decrementing $|H|$ we obtain a higher scoring model L', from which model G can be obtained by a single increment of $|H|$. A similar situation in respect of a decrement of $|H|$ can occur when L has $m+1$ components.

So, we introduce an operation that we call the *parameter adjustment*. It consists of the successive component splitting and merging (or component merging and splitting) with no requirement that a single increment or decrement of $|H|$ would increase the model score. Below we give a formal definition.

Definition 6.1 *We say that model L' is obtained from model L by* parameter adjustment *if there exists model L^* such that:*

- *L^* is obtained from L by splitting a component and then running EM, and L' is obtained from L^* by merging two components and then running EM,*
  or
- *L^* is obtained from L by merging two components and then running EM, and L' is obtained from L^* by splitting a component and then running EM.*

For an m-component model L, there are $O(m^3)$ possible ways to perform the parameter adjustment, because for each of m ways to split there are $\binom{m+1}{2}$ ways to merge, or alternatively for each of $\binom{m}{2}$ ways to merge there are $m-1$ ways to split. However, in our implementation we will consider only $O(m^2)$ ways to perform the parameter adjustment, as described in the next section.

Even if all the $O(m^3)$ ways to perform the parameter adjustment are tried, there are no guarantees that such an operation will improve the parameters of L when they are not optimal. Also, the only reason for choosing a single splitting and a single merging rather than changing $|H|$ by more than 1 is that this approach is the fastest. However, as it will be seen from the experiments, this operation often helps to find higher scoring models.

6.2 Algorithms

In this section, we discuss the algorithms that will be used in our experiments on learning of LC models.

6.2.1 Algorithm Based on Splitting and Merging

In this section, we provide our algorithm that learns LC models by using the component splitting, component merging, and parameter adjustment operations. Algorithm *SplitMerge* takes training data $\mathbf{D}$ as input and returns an LC model.[1] It tries to find an LC model that is the best according to function the *score*, which can be any penalised likelihood scoring function. The pseudocode of *SplitMerge* is given in page 59. The algorithm starts with the one–component model L, for which the maximum likelihood parameters are deterministically computed from $\mathbf{D}$. The algorithm repeatedly tries to increase the score of L by either incrementing $|H|$ (procedure *Split*), decrementing $|H|$ (procedure *Merge*), or adjusting parameters (procedure *Adjust*). In phase 1, $|H|$ is repeatedly incremented until the score of L does not increase. Similarly, in phase 2, $|H|$ is repeatedly decremented until the score of L does not increase. Phases 1 and 2 are performed repeatedly one after another until none of them increases the score of L. Then one attempt to adjust the parameters of L is made. If adjusting the parameters increases the score of L, the algorithm goes back to running phases 1 and 2. Otherwise, model L is returned.

So, algorithm *SplitMerge* follows the idea of the greedy equivalence search (GES) algorithm for learning Bayesian network structures from complete data (Meek, 1997; Chickering, 2002). GES starts with a network containing no edges and in phase 1 repeatedly increases the model score by increasing the model complexity. In phase 2 it repeatedly increases the model score by decreasing the model complexity and then it terminates. Algorithm *SplitMerge* also repeatedly increases the model complexity in phase 1 and repeatedly decreases it in phase 2. The difference is that in *SplitMerge* we allow phases 1 and 2 to be executed more than once.[2] Also, we have introduced the third phase of improving model parameters while keeping the complexity the same, because in the presence of hidden variables the optimal parameters can not be computed in a closed-form.

[1] Contrary to the anytime algorithms from Section 4.5.1, *SplitMerge* has a stopping criterion.

[2] In this aspect, *SplitMerge* follows the approach of SHC algorithm for HLC models, described in Section 3.4.2.

Procedure 6.1 *SplitMerge*(**D**)

Let L be the LC model with one component.
doPhase1 $\leftarrow$ *true*, *doPhase2* $\leftarrow$ *true*.
loop
  if *doPhase1* **then**
    $L_0 \leftarrow L$.
    repeat
      $L' \leftarrow Split(L, \mathbf{D})$.
      if $score(L') > score(L)$ **then**
        $L \leftarrow L'$.
      end if
    until $L \neq L'$.
    if $score(L) > score(L_0)$ **then**
      doPhase2 $\leftarrow$ *true*.
    end if
    doPhase1 $\leftarrow$ *false*.
  end if
  if *doPhase2* **then**
    $L_0 \leftarrow L$.
    repeat
      $L' \leftarrow Merge(L, \mathbf{D})$.
      if $score(L') > score(L)$ **then**
        $L \leftarrow L'$.
      end if
    until $L \neq L'$.
    if $score(L) > score(L_0)$ **then**
      doPhase1 $\leftarrow$ *true*.
    end if
    doPhase2 $\leftarrow$ *false*.
  end if
  if not *doPhase1* and not *doPhase2* **then**
    $L' \leftarrow Adjust(L, \mathbf{D})$.
    if $score(L') > score(L)$ **then**
      $L \leftarrow L'$, *doPhase1* $\leftarrow$ *true*, *doPhase2* $\leftarrow$ *true*.
    else
      return L.
    end if
  end if
end loop

Procedure *Split* increments $|H|$ by splitting a component in L. The pseudocode is given below. Here we introduce some improvements over the implementation of component splitting from Section 4.4.2. One improvement is that for each component h_s, a number (constant N_0 from step 2) of independent random splits is performed (in Section 4.4.2, each component was split only once). As in Section 4.4.2, component h_s is split randomly by making $P_{L^*}(D_i|h_s^1) - P_L(D_i|h_s) = \vec{r}_i$ and $P_{L^*}(D_i|h_s^2) - P_L(D_i|h_s) = -\vec{r}_i$ (see Definition 4.2 on page 22). Each element of $\vec{r}_i$ is a random number from $[-p; p]$ (and at least one of them is exactly p or $-p$), where $p \in \mathbb{R}$ is a small positive parameter.[3] And it is ensured that $\forall i : \sum_{j=1}^{|D_i|} r_{i,j} = 0$.

Another major improvement is that after splitting h_s a partial EM rather than a normal EM is run. That is, only the parameters for the two new components are updated. This can be understood as running a normal EM for the model containing only two new components and after that substituting h_s in L with those two components.[4] A model that has the highest score after a partial EM is selected and parameters of that model are updated by running a normal EM. We have introduced a partial EM, because then the algorithm time is used for learning those parameters that are most likely to be different in a new model.

Procedure 6.2 *Split*$(L, \mathbf{D})$

1: **for each** component h_s of L **do**
2: Produce set $\mathcal{L}_0$ of two-component models by performing N_0 random independent splits of h_s and for each split producing a model that contains only new components h_s^1 and h_s^2 from L^* (see Definition 4.2 on page 22).
3: Obtain model L_0 by running the multiple restart EM with $\mathcal{L}_0$ as starting points and data $\mathbf{D}_s$.
4: Obtain model L_s from L by substituting h_s with the two components from L_0.
5: **end for**
6: $L' \leftarrow \arg\max_{L_s} score(L_s)$.
7: Optimise the parameters of L' by running the standard EM with data $\mathbf{D}$.
8: **return** L'.

[3] If $\vec{r}_i$ causes any parameter from $P_{L^*}(D_i|h_s^1)$ or $P_{L^*}(D_i|h_s^2)$ to be outside of $[0.000001; 0.999999]$, $\vec{r}_i$ is scaled down so that all the parameters are inside this interval. If already $P_L(D_i|h_s)$ contains parameters outside of this interval, $\vec{r}_i$ is set to $\vec{0}$.

[4] Both in producing the two-component model and in substituting h_s is L, marginal probabilities of those two components are scaled so that the sum of marginal probabilities of all the components in a model is equal to 1.

Procedure *Merge* decrements $|H|$ by merging two components in L. The pseudocode is given below. The procedure is implemented as described in Section 5.4.

Procedure 6.3 *Merge*$(L, \mathbf{D})$

1: $\mathcal{L} \leftarrow \emptyset$.
2: **for each** pair of components $\{h_a, h_b\}$ of L **do**
3: Obtain model L^* by merging h_a and h_b.
4: Add L^* to $\mathcal{L}$.
5: **end for**
6: Obtain model L' by running the multiple restart EM with $\mathcal{L}$ as starting points and data $\mathbf{D}$.
7: **return** L'.

Procedure *Adjust* tries to increase the score of L by performing the parameter adjustment operation. The pseudocode is given below. For an adjustment to be accepted, we require the increase in score to be higher than a positive parameter δ. Otherwise, the algorithm may spend lots of time making insignificant improvements in the parameters of L.

First, we try to increase the score of L by decrementing and then incrementing $|H|$. Here, when incrementing $|H|$, we consider only components from the best model obtained by component merging (which is model L' in steps 1 and 2). If this does not succeed, then we try to increase the score of L by incrementing and then decrementing $|H|$. Here, when decrementing $|H|$, we consider only those pairs of components where one component has just been obtained by splitting and the other is an "old" component inherited from L. So, we try to save the computation time by considering only the most promising candidates for splitting and merging.

Our operation of parameter adjustment is similar to the simultaneous splitting and merging of components used by Ueda et al. (2000) (see Section 3.3) to improve parameters of a mixture model with a fixed number of components. There, the components involved in splitting are required to be different from those involved in merging, and criteria for selecting the most promising candidates for splitting and merging are used. We, on the other hand, allow situations where the same component is involved in both splitting and merging. The best split or merge operation is determined by the EM algorithm alone.

In the implementation of this procedure, model L' in step 1 and models L_s in step 8 are not computed directly but taken from the last run of *Merge* and the last run of *Split*. This can be done because in *SplitMerge* the procedure

Adjust is called only when model L remains unchanged both in phase 1 and phase 2.

Procedure 6.4 *Adjust*$(L, \mathbf{D})$

1: $L' \leftarrow Merge(L, \mathbf{D})$.
2: $L'' \leftarrow Split(L', \mathbf{D})$.
3: **if** $score(L'') > score(L) + \delta$ **then**
4: **return** L''.
5: **else**
6: $\mathcal{L} \leftarrow \emptyset$.
7: **for each** component h_s of L **do**
8: Obtain model L_s as in $Split(L, \mathbf{D})$.
9: **for each** pair of components $\{h_a, h_b\}$ where h_a is a new component and h_b is an old component of L_s **do**
10: Obtain model L^* by merging h_a and h_b in L_s.
11: Add L^* to $\mathcal{L}$.
12: **end for**
13: **end for**
14: Obtain model L'' by running the multiple restart EM with $\mathcal{L}$ as starting points and data $\mathbf{D}$.
15: **if** $score(L'') > score(L) + \delta$ **then**
16: $L \leftarrow L''$.
17: **end if**
18: **return** L.
19: **end if**

6.2.2 Algorithms Based on Standard Starting Points

In the experiments, the algorithm *SplitMerge* will be compared with the algorithms that use standard starting points for EM. One of them, called *Standard3* has almost the same main procedure as *SplitMerge* (see page 59). It also has 3 phases: increasing $|H|$, decreasing $|H|$, and improving parameters. The difference from *SplitMerge* is that instead of calling *Split*, *Merge*, or *Adjust* it obtains L' by running the multiple restart EM on a model that has correspondingly one component more than L, one component less than L, or the same number of components as L (in the latter case, the score increase by more than δ is also required). Starting points for EM are sampled from the uniform distribution.

Similarly to Section 4.5.1, we also test the *StandardFixed* algorithm, which is described in page 42.

6.3 Experiments

In this section we describe the experiments where we compare the approach that uses the component splitting and merging (algorithm *SplitMerge*) with the approach that uses standard starting points (algorithms *Standard3* and *StandardFixed*) for learning LC models. First, we describe the setup of experiments. Then we give the results for synthetic and for real data. Finally, we discuss the results.

6.3.1 Setup of Experiments

The setup of experiments is the following. As a model scoring function, we use the BIC score. We have used the Cheeseman-Stutz (CS) score in our earlier experiments from Section 4.5. However, we have noticed the following undesirable property of the CS score. It can happen that for models M_1 and M_2 having the same structure but different parameterisations $CS(M_1) < CS(M_2)$ while $LL(\mathbf{D}|M_1) > LL(\mathbf{D}|M_2)$. [5] To understand why does this happen, recall that for $M = (\mathbf{m}, \theta)$

$$CS(M) = LL(\mathbf{D}|M) - \big(LL(\mathbf{D}'|M) - \log P(\mathbf{D}'|\mathbf{m})\big) \tag{6.1}$$

where $\mathbf{D}'$ is obtained by completing $\mathbf{D}$ using model M. So, $\mathbf{D}'$ is a function of $\mathbf{D}$ and θ, and thus the penalty term $\big(LL(\mathbf{D}'|M) - \log P(\mathbf{D}'|\mathbf{m})\big)$ depends on θ. Since the CS score can be misleading when comparing different models having the same structure, we have decided to use the BIC score, where the penalty term does not depend on model parameters.

The EM algorithm searches for the ML parameters. N_0 in *Split* and N_0 in *StandardFixed* are set to 16. Otherwise, the setup of EM is the same as described in Section 4.5.2. The parameter of the multiple restart EM $q = 2$ in *Split*, *Standard3*, and *StandardFixed*. $q = 10$ in *Merge* and *Adjust*, because the number of possible pairs to merge grows fast with the number of components. The number of starting points for EM in *Standard3* is set to 64 in order to make a running time of *SplitMerge* and *Standard3* similar. Parameter p from *Split* is 0.001. Parameter δ from *Adjust* and *Standard3* is 1. The experiments are run on a Java^{TM} 2 platform, 2.8 GHz Intel(R) processor.

[5] To check how often does this happen, we have performed the following test. We sampled $\mathbf{D}$ of size 10000 from a randomly parameterised 5:5x2 model. Then we created 5:5x2 models M_1 and M_2 and learned their parameters by using the multiple restart EM with $\mathbf{D}$ as the training data. We repeated this whole procedure 20 times. In 7 out of those 20 runs, the model from M_1 and M_2 having higher CS score had lower log-likelihood.

In the same way as in Section 4.5, a single algorithm is run 5 times on a single data set. Again, the number of components m in *StandardFixed* is the number of components in the best scoring model found by *SplitMerge* for a given data set. For *SplitMerge* and *Standard3*, the algorithm is stopped if it does not terminate in 5 hours (for synthetic data, in 10 hours), and the current model L is returned. The running time of *StandardFixed* is equal to the mean running time of *SplitMerge* for a given data set.

When displaying the results of the experiments, the same notation as in Section 4.5.3 is used.[6] That is, for each row in a table, the field in the first column is written in bold if *SplitMerge* found better scoring models than both *Standard3* and *StandardFixed*. This field is underlined if both *Standard3* and *StandardFixed* found better scoring models than *SplitMerge*. For the score values, bold text indicates a significantly better (and underlined – a significantly worse) score when comparing pairwise *SplitMerge* with *Standard3* and *SplitMerge* with *StandardFixed*. The field "t" in the tables indicates the rounded mean running time in minutes. The field "$|H|$" indicates the number of components in the best scoring model.

6.3.2 Results for Synthetic Data

In these experiments, training data $\mathbf{D}$ is always sampled from some LC model. Forward sampling is used and $\mathbf{D}$ over observed variables is always complete. The generative LC model is always parameterised randomly, with its parameters sampled from the uniform distribution.

In the first series of tests, we sampled $\mathbf{D}$ of different sizes from the same LC model. We did this for several different LC model structures. The results are shown in Tables 6.1-6.6.

Then we have selected one $\mathbf{D}$ where t=600 (that is, the algorithms have been stopped because of the time limit) and let *Standard3* and *SplitMerge* run until they terminate themselves. We did this for $\mathbf{D}$ of size 10^5 generated from the 10:15x2 model (from Table 6.4). The results are shown in Table 6.7.

[6] Just to remind, we say that one algorithm finds *better* scoring models than another algorithm if the difference in the mean score values is higher than 1. We say that one algorithm finds *significantly better* scoring models than another algorithm if intervals for the mean do not overlap and the difference in the mean score values is higher than 1.

$\|\mathbf{D}\|$	*Standard3*			*SplitMerge*			*StandardFixed*
	score	t	$\|H\|$	score	t	$\|H\|$	score
10^2	-323.0 ± 0.0	0	1	-323.0 ± 0.0	0	1	-323.0 ± 0.0
10^3	-3030.9 ± 0.2	0	2	-3031.4 ± 0.0	0	2	-3030.8 ± 0.0
10^4	-29856.7 ± 0.3	0	2	-29857.4 ± 0.0	0	2	-29856.8 ± 0.5
10^5	-298224.8 ± 4.4	0	3	-298222.4 ± 1.3	0	3	-298223.0 ± 1.4
10^6	-2977109.7 ± 39.9	0	3	-2977072.4 ± 0.0	0	3	-2977079.3 ± 10.8
10^7	-29772167.8 ± 56.3	0	4	**-29772008.3 ± 0.0**	0	3	-29772100.9 ± 70.3
10^8	-297730211.6 ± 1388.3	0	4	**-297728637.2 ± 17.2**	0	5	-297728949.2 ± 375.3
10^9	-2977335064.3 ± 4568.8	0	5	**-2977329260.2 ± 56.2**	1	5	-2977328755.5 ± 1331.9

Table 6.1: Sampled from a 3:3x3 model.

$\|\mathbf{D}\|$	*Standard3*			*SplitMerge*			*StandardFixed*
	score	t	$\|H\|$	score	t	$\|H\|$	score
10^2	-498.1 ± 0.0	0	1	-498.1 ± 0.0	0	1	-498.1 ± 0.0
10^3	-4838.4 ± 0.0	0	1	-4838.4 ± 0.0	0	1	-4838.4 ± 0.0
10^4	-47712.1 ± 0.4	1	3	-47711.9 ± 0.0	0	3	-47712.3 ± 0.8
10^5	-475715.8 ± 54.1	3	7	**-475664.4 ± 0.3**	3	7	-475678.3 ± 12.0
10^6	-4753185.1 ± 196.4	3	8	**-4752826.0 ± 11.2**	7	8	-4752877.3 ± 31.1
10^7	-47531173.0 ± 779.6	4	10	**-47528482.3 ± 30.2**	13	10	-47529145.5 ± 270.2
10^8	-475327706.8 ± 67268.8	3	9	**-475268998.1 ± 85.1**	20	13	-475270455.1 ± 263.9
10^9	-4753270535.0 ± 777481.7	4	13	**-4752732197.1 ± 355.3**	73	19	-4752735541.9 ± 683.5

Table 6.2: Sampled from a 10:7x2 model.

$\|\mathbf{D}\|$	*Standard3*			*SplitMerge*			*StandardFixed*
	score	t	$\|H\|$	score	t	$\|H\|$	score
10^2	-693.7 ± 0.0	0	1	-693.7 ± 0.0	0	1	-693.7 ± 0.0
10^3	-6692.3 ± 0.0	5	4	-6692.5 ± 0.0	3	4	-6692.3 ± 0.0
10^4	-65436.7 ± 0.1	17	6	-65436.5 ± 0.0	16	6	-65436.5 ± 0.1
10^5	-650348.2 ± 107.9	34	9	-650302.2 ± 0.0	34	9	-650304.4 ± 4.0
10^6	-6497124.3 ± 334.5	40	9	-6496881.4 ± 0.0	59	9	-6496957.6 ± 89.6
10^7	-64966268.3 ± 3550.9	40	12	-64964215.0 ± 0.0	82	9	-64964233.1 ± 52.3
10^8	-649632767.5 ± 38429.8	48	10	**-649612538.3 ± 360.2**	159	10	-649614304.0 ± 493.4
10^9	-6496426973.9 ± 410587.2	49	11	-6496152390.6 ± 877.6	302	12	-6496161104.4 ± 9768.0

Table 6.3: Sampled from a 10:10x2 model.

$\|\mathbf{D}\|$	*Standard3*			*SplitMerge*			*StandardFixed*
	score	t	$\|H\|$	score	t	$\|H\|$	score
10^2	-1049.3 ± 0.0	0	1	-1049.3 ± 0.0	0	1	-1049.3 ± 0.0
10^3	-10055.7 ± 0.0	16	4	-10055.7 ± 0.0	8	4	-10055.7 ± 0.0
10^4	-98283.1 ± 0.2	218	7	-98283.1 ± 0.0	203	7	-98282.9 ± 0.0
10^5	**-980627.5 ± 0.0**	600	7	-982084.0 ± 1010.8	600	7	**-980627.5 ± 0.1**
10^6	**-9816413.3 ± 0.0**	600	6	-9850314.1 ± 0.2	600	5	-9850314.1 ± 0.2

Table 6.4: Sampled from a 10:15x2 model.

$\|\mathbf{D}\|$	*Standard3*			*SplitMerge*			*StandardFixed*
	score	t	$\|H\|$	score	t	$\|H\|$	score
10^2	-1120.4 ± 0.0	0	1	-1120.4 ± 0.0	0	1	-1120.4 ± 0.0
10^3	-10914.4 ± 0.0	2	1	-10914.4 ± 0.0	1	1	-10914.4 ± 0.0
10^4	-107020.8 ± 0.0	271	8	-107029.5 ± 18.8	274	8	-107020.7 ± 0.1
10^5	**-1067192.9 ± 0.0**	600	6	-1069382.2 ± 0.0	600	5	-1069381.9 ± 0.2
10^6	-10720139.2 ± 0.7	600	4	-10720138.2 ± 0.0	600	4	-10720138.5 ± 0.2

Table 6.5: Sampled from a 10:10x3 model.

$\|\mathbf{D}\|$	*Standard3*			*SplitMerge*			*StandardFixed*
	score	t	$\|H\|$	score	t	$\|H\|$	score
10^2	-1310.4 ± 0.0	1	2	-1310.4 ± 0.0	0	2	-1310.4 ± 0.0
10^3	-12354.7 ± 0.0	22	4	-12354.7 ± 0.0	13	4	-12354.7 ± 0.0
10^4	-121655.4 ± 0.0	252	5	-121655.4 ± 0.0	212	5	-121655.4 ± 0.0
10^5	-1223176.0 ± 0.0	600	3	-1223176.0 ± 0.0	600	3	-1223176.0 ± 0.0
10^6	-13327103.2 ± 0.0	600	1	-13327103.2 ± 0.0	600	1	-13327103.2 ± 0.0

Table 6.6: Sampled from a 5:20x2 model.

$\|\mathbf{D}\|$	*Standard3*			*SplitMerge*			*StandardFixed*
	score	t	$\|H\|$	score	t	$\|H\|$	score
10^5	-979691.3 ± 3.7	1701	10	-979689.0 ± 0.1	1978	10	-979689.4 ± 0.7

Table 6.7: Sampled from the 10:15x2 model, no time limit.

In the second series of tests, we sampled $\mathbf{D}$ of the same size from models having different structures. The size of $\mathbf{D}$ was always 10^6. The results are shown in Table 6.8. Part of the results here comes from Tables 6.1-6.6.

Structure	*Standard3*			*SplitMerge*			*StandardFixed*
	score	t	$\|H\|$	score	t	$\|H\|$	score
10:7x2	-4753185.1 ± 196.4	3	8	**-4752826.0 ± 11.2**	7	8	-4752877.3 ± 31.1
10:10x2	-6497124.3 ± 334.5	40	9	-6496881.4 ± 0.0	59	9	-6496957.6 ± 89.6
10:15x2	**-9816413.3 ± 0.0**	600	6	-9850314.1 ± 0.2	600	5	-9850314.1 ± 0.2
5:20x2	-13327103.2 ± 0.0	600	1	-13327103.2 ± 0.0	600	1	-13327103.2 ± 0.0
3:3x3	-2977109.7 ± 39.9	0	3	-2977072.4 ± 0.0	0	3	-2977079.3 ± 10.8
10:5x3	-5362696.0 ± 222.0	6	9	**-5362307.7 ± 61.8**	17	9	-5362322.5 ± 34.4
10:7x3	-7478527.3 ± 213.3	89	10	**-7478040.3 ± 1.0**	165	10	-7478072.8 ± 39.7
10:10x3	-10720139.2 ± 0.7	600	4	-10720138.2 ± 0.0	600	4	-10720138.5 ± 0.2
10:2,2,2,5,5,5	-6722963.5 ± 897.6	30	10	**-6722202.8 ± 4.5**	64	9	-6722308.2 ± 72.5
10:2,3,4,5,6,7	-8222214.9 ± 1026.7	213	9	**-8221544.7 ± 0.8**	399	9	-8221556.4 ± 7.3

Table 6.8: Sampled from models having different structures.

In the third series of tests, we sampled $\mathbf{D}$ from models having different parameterisations of the same structure. We did this for two different LC model structures. We have chosen the structure 10:10x2 with $|\mathbf{D}| = 10^6$ and the structure 10:10x3 with $|\mathbf{D}| = 10^4$, because in the first series of tests *SplitMerge* was better than the standard algorithms for the former configuration and worse for the latter one. The results are shown in Tables 6.9 and 6.10. Here $\#\theta$ indicates the number of parameterisation.

$\#\theta$	*Standard3*			*SplitMerge*			*StandardFixed*
	score	t	$\|H\|$	score	t	$\|H\|$	score
1	-6497124.3 ± 334.5	40	9	-6496881.4 ± 0.0	59	9	-6496957.6 ± 89.6
2	-6644081.3 ± 847.2	44	10	**-6643109.8 ± 0.1**	69	10	-6643212.3 ± 67.0
3	-6649520.6 ± 40.2	46	9	**-6649407.2 ± 0.7**	99	10	-6649424.5 ± 24.8
4	-6491945.9 ± 150.2	41	9	**-6491335.3 ± 0.5**	93	10	-6491580.7 ± 151.8
5	-6573339.8 ± 69.1	44	9	**-6573111.5 ± 1.3**	118	10	-6573197.0 ± 100.0

Table 6.9: Sampled from 10:10x2 models, $|\mathbf{D}| = 10^6$.

In the fourth series of tests, we sampled $\mathbf{D}$ from models having 10 binary observed variables and different number of components. The size of $\mathbf{D}$ was always 10^6. The results are shown in Table 6.11.

In the first, second, and fourth series of tests, whenever $|H|$ for *SplitMerge* was different from the true $|H|$ (that is, $|H|$ in the model that data was

$\#\theta$	*Standard3*			*SplitMerge*			*StandardFixed*
	score	t	$\lvert H\rvert$	score	t	$\lvert H\rvert$	score
1	-107020.8 ± 0.0	271	8	-107029.5 ± 18.8	274	8	-107020.7 ± 0.1
2	-107019.3 ± 0.1	294	8	-107019.2 ± 0.0	321	8	-107019.1 ± 0.0
3	-103540.0 ± 0.0	203	7	-103540.0 ± 0.0	205	7	-103539.9 ± 0.1
4	-105615.5 ± 0.1	251	8	-105615.5 ± 0.0	269	8	-105615.5 ± 0.1
5	-105723.3 ± 0.1	282	8	-105723.2 ± 0.0	272	8	-105723.3 ± 0.1

Table 6.10: Sampled from 10:10x3 models, $|\mathbf{D}| = 10^4$.

m	*Standard3*			*SplitMerge*			*StandardFixed*
	score	t	$\lvert H\rvert$	score	t	$\lvert H\rvert$	score
2	-5389022.4 ± 0.0	5	2	-5389022.5 ± 0.2	2	2	-5389022.5 ± 0.1
4	-6091546.9 ± 461.7	17	4	-6091279.2 ± 0.2	22	4	-6091337.3 ± 93.9
6	-6564076.8 ± 15.8	22	5	**-6564031.3 ± 24.5**	36	6	-6564074.6 ± 48.4
8	-6379315.8 ± 119.7	32	7	**-6379119.1 ± 0.0**	57	7	-6379189.2 ± 77.0
10	-6713825.3 ± 1894.0	41	11	**-6711800.7 ± 1.0**	78	10	-6711985.5 ± 59.7
12	-6526733.2 ± 424.8	46	10	**-6525960.9 ± 0.2**	186	10	-6525980.3 ± 11.8
14	-6621460.7 ± 35.9	50	10	**-6621314.3 ± 0.3**	59	10	-6621422.2 ± 97.2
16	-6762383.9 ± 1762.7	41	11	**-6760153.5 ± 48.7**	111	14	-6760326.5 ± 78.7
18	-6733141.8 ± 208.7	65	15	**-6732492.2 ± 59.4**	342	16	-6732537.8 ± 65.4
20	-6737257.7 ± 2433.0	59	18	**-6733599.0 ± 4.1**	343	17	-6733603.6 ± 5.5

Table 6.11: Sampled from m:10x2 models.

sampled from), we have also run *StandardFixed* with parameter m equal to the true $|H|$. In all these runs, whenever "t" for *SplitMerge* was lower than 600 (that is, *SplitMerge* terminated itself), the mean score for *StandardFixed* with true $|H|$ was lower than the mean score for *StandardFixed* with $|H|$ given by *SplitMerge*.

We have selected three data sets (two for which *SplitMerge* performs better than *Standard3* and one for which it performs worse) and displayed how the mean score changes during time. These three data sets are $\mathbf{D}$ of size 10^6 sampled from the 10:10x2, 10:15x2, and 10:7x3 models (the second, the third, and the seventh entries in Table 6.8). The results are displayed in Figures 6.1-6.3 (bars at the end indicate 95% confidence intervals for the mean).

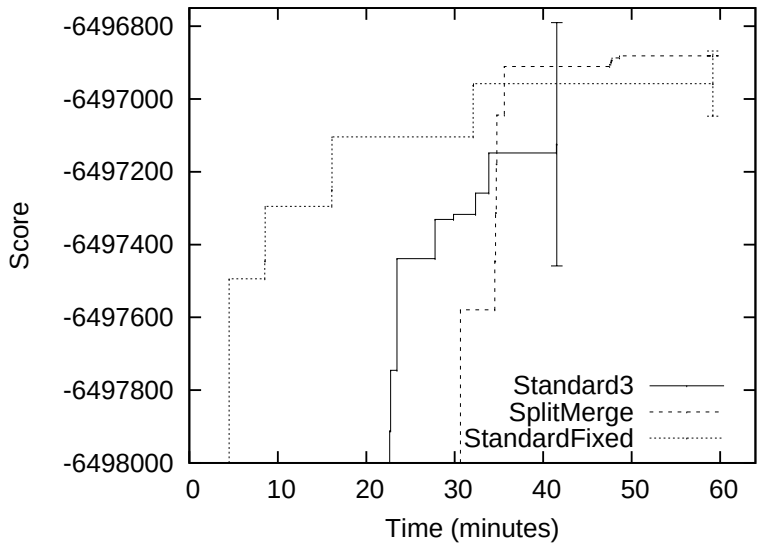

Figure 6.1: Score change during time for data sampled from the 10:10x2 model, $|\mathbf{D}| = 10^6$.

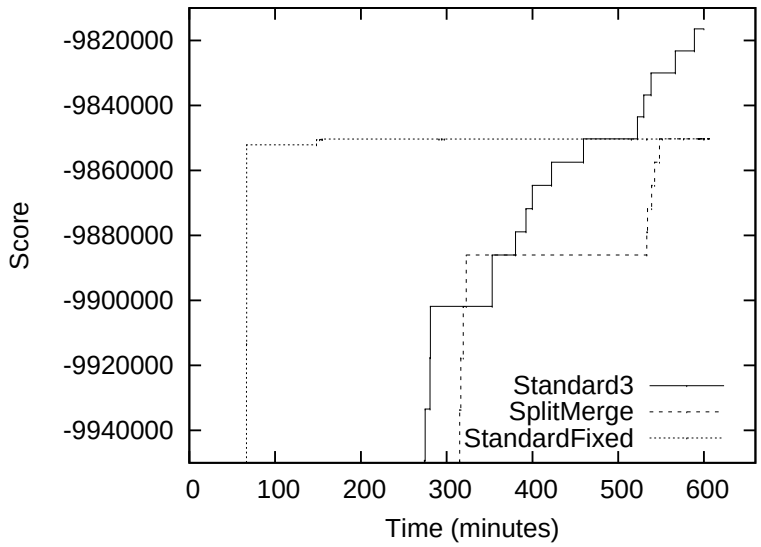

Figure 6.2: Score change during time for data sampled from the 10:15x2 model, $|\mathbf{D}| = 10^6$.

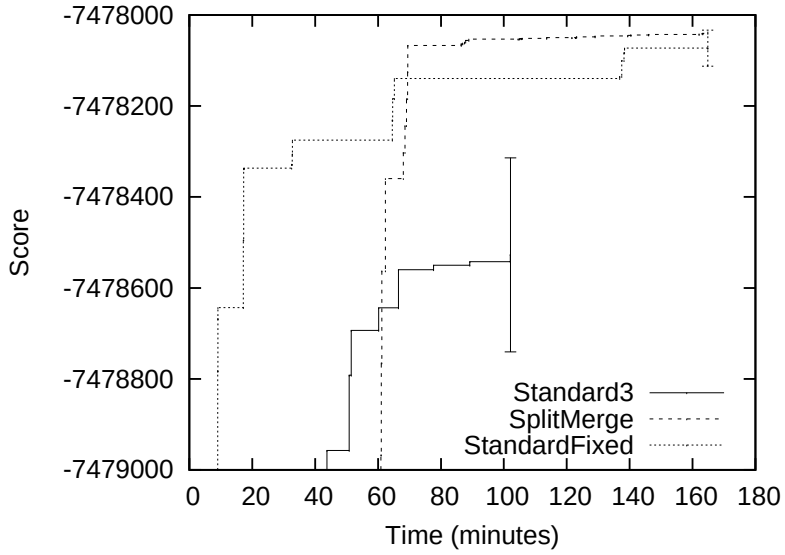

Figure 6.3: Score change during time for data sampled from the 10:7x3 model, $|\mathbf{D}| = 10^6$.

6.3.3 Results for Real Data

For training data, we selected 20 classification data sets from the UCI Machine Learning Repository (Blake and Merz, 1998) and discarded the class information. Continuous variables were converted into binary ones by performing the equal-frequency binning. When separate training and test data were available, they were joined into a single training data. Some of these data sets contain missing data. All the 20 data sets are listed in Table 6.12.[7] The results of the experiments are shown in Table 6.13. To make the later discussion easier, we partitioned data sets into two groups: data sets of size higher than 4500 are above the middle line, and data sets of size lower than 4500 are below it.

Similarly as for synthetic data, we have selected one **D** where t=300 (that is, the algorithms have been stopped because of the time limit) and let *Standard3* and *SplitMerge* run until they terminate themselves. We did this for "Spambase" data set. The results are shown in Table 6.14. Because of long running time, *SplitMerge* and *StandardFixed* were run only 2 times each.

We have also checked what impact the parameter adjustment operation has on the performance of *SplitMerge*. In Table 6.15 we compare the performance of *SplitMerge* with and without this operation.[8] In *SplitMerge-NoAdjust*, lines 31-36 in the pseudocode on page 59 are replaced by line 35. Here we did run not any new tests, but used the runs of *SplitMerge* from Table 6.13. In Table 6.15 we include only those data sets for which at least one of the fields "score", "t", or "$|H|$" is different in *SplitMerge* and *SplitMerge-NoAdjust*. For the other data sets, either the parameter adjustment has not been tried in *SplitMerge* because of the 5 hour limit for the algorithm running time or because trying the parameter adjustment did not improve the score and did not make an impact on the "t" value (which is a *rounded* time in minutes).

Similarly as for synthetic data, we have selected three data sets and displayed how the mean score changes during time. It is shown in Figures 6.4-6.7 (bars at the end indicate 95% confidence intervals for the mean).

[7] In the table, "Credit" stands for "Credit Card Application Approval", "Heart" – for the processed "Cleveland" data set from "Heart Disease", "Image" – for "Image segmentation", "Letter" – for "Letter Recognition", "Pen" – for "Pen-Based Recognition of Handwritten Digits", "Thyroid" – for the data set from "Thyroid Disease" suited for training artificial neural networks, "Wisconsin" – for the original data set from "Wisconsin Breast Cancer". "Satimage", "Shuttle", and "Vehicle" are from the Statlog Project. For "Audiology" and "Wisconsin", the identifier variable has been discarded.

[8] In the same way as in the previous tables, bold and underlined fonts are used to show differences in the performance of the algorithms.

Data set name	Data set size	Number of variables	Number of categorical variables
Abalone	4177	8	1
Adult	48842	14	8
Audiology	226	69	69
Credit	690	15	9
Heart	303	13	1
Housing	506	13	1
Image	2310	19	0
Letter	20000	16	0
Mushroom	8124	22	22
Page	5473	10	0
Pen	10992	16	0
Pima	768	8	0
Satimage	6435	36	0
Shuttle	58000	9	0
Spambase	4601	57	0
Thyroid	7200	21	15
Vehicle	846	18	0
Voting	435	16	16
Wisconsin	699	9	0
Yeast	1484	8	1

Table 6.12: Description of real data sets.

Data set name	*Standard3*			*SplitMerge*			*StandardFixed*
	score	t	$\|H\|$	score	t	$\|H\|$	score
Adult	-532344.7 ± 2790.8	300	6	**-520908.5 ± 375.5**	300	11	-520907.8 ± 717.7
Letter	-172561.3 ± 331.1	300	19	-172432.8 ± 248.7	300	18	-172314.8 ± 39.5
Mushroom	-94155.0 ± 1090.5	300	8	**-82935.1 ± 722.2**	300	17	-83648.2 ± 302.7
Page	-22768.1 ± 39.5	5	14	-22766.6 ± 2.9	5	14	**-22735.1 ± 6.9**
Pen	-82411.6 ± 827.9	260	30	**-81454.4 ± 11.5**	300	41	**-81402.1 ± 16.3**
Satimage	-66602.9 ± 457.0	300	16	**-64598.4 ± 121.9**	300	25	-64419.4 ± 87.1
Shuttle	-256561.5 ± 265.9	5	14	**-256027.6 ± 10.4**	16	21	-256073.4 ± 28.3
Spambase	-83800.0 ± 192.9	300	11	**-83090.0 ± 74.8**	300	15	-83127.5 ± 81.8
Thyroid	-44562.9 ± 26.1	21	6	-44542.9 ± 0.2	21	7	-44534.6 ± 18.3
Abalone	-12389.4 ± 0.0	2	5	-12389.5 ± 0.0	1	5	-12389.4 ± 0.0
Audiology	-3403.0 ± 0.1	9	2	-3403.2 ± 0.0	2	2	-3403.1 ± 0.1
Credit	**-7564.7 ± 3.2**	16	5	-7591.3 ± 22.7	3	4	-7568.3 ± 3.2
Heart	-2388.1 ± 0.0	1	2	-2388.1 ± 0.0	0	2	-2388.1 ± 0.0
Housing	-3192.6 ± 5.7	2	7	-3198.3 ± 9.4	1	7	-3190.8 ± 5.7
Image	-15514.5 ± 35.2	32	14	-15516.5 ± 8.3	28	13	-15503.6 ± 5.5
Pima	-3995.6 ± 0.2	1	4	-3995.7 ± 0.0	0	4	-3995.6 ± 0.0
Vehicle	-6471.5 ± 7.4	13	10	-6471.1 ± 7.0	9	10	-6467.1 ± 1.3
Voting	-3085.6 ± 0.0	5	5	-3085.6 ± 0.0	2	5	-3085.6 ± 0.0
Wisconsin	-2560.7 ± 0.0	1	3	-2560.9 ± 0.0	0	3	-2560.7 ± 0.0
Yeast	-6213.5 ± 0.0	0	2	-6213.7 ± 0.0	0	2	-6213.5 ± 0.0

Table 6.13: Results for real data.

Data set name	*Standard3*			*SplitMerge*			*StandardFixed*
	score	t	$\|H\|$	score	t	$\|H\|$	score
Spambase	-83425.8 ± 500.0	963	16	**-82705.3 ± 4.1**	3738	19	-82850.6 ± 79.3

Table 6.14: Results for "Spambase" data set, no time limit.

Data set name	*SplitMerge*			*SplitMerge-NoAdjust*		
	score	t	$\|H\|$	score	t	$\|H\|$
Mushroom	-82935.1 ± 722.2	300	17	-83091.5 ± 924.3	165	17
Page	**-22766.6 ± 2.9**	5	14	-22831.0 ± 4.7	2	15
Pen	-81454.4 ± 11.5	300	41	-81455.2 ± 13.2	266	41
Shuttle	**-256027.6 ± 10.4**	16	21	-256042.2 ± 2.6	9	21
Thyroid	-44542.9 ± 0.2	21	7	-44542.9 ± 0.2	17	7
Audiology	-3403.2 ± 0.0	2	2	-3403.2 ± 0.0	1	2
Credit	-7591.3 ± 22.7	3	4	-7610.6 ± 13.0	1	4
Housing	-3198.3 ± 9.4	1	7	-3201.7 ± 4.6	1	8
Image	**-15516.5 ± 8.3**	28	13	-15534.3 ± 8.0	12	13
Vehicle	**-6471.1 ± 7.0**	9	10	-6488.4 ± 11.3	5	11
Voting	-3085.6 ± 0.0	2	5	-3086.3 ± 0.7	1	5

Table 6.15: Results for real data: *SplitMerge* without parameter adjustment.

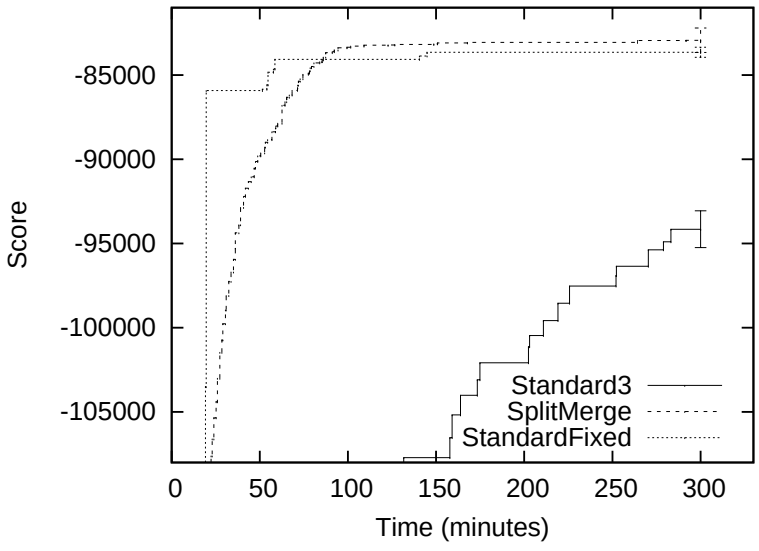

Figure 6.4: Score change during time for "Mushroom" data set.

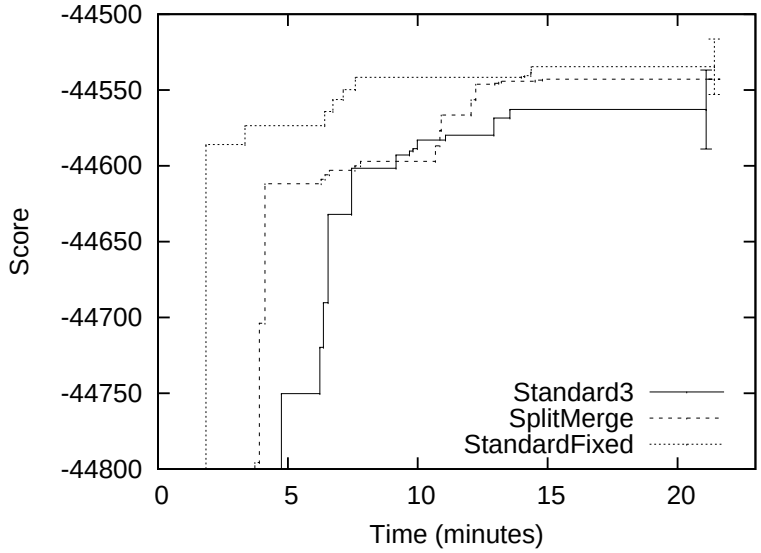

Figure 6.5: Score change during time for "Thyroid" data set.

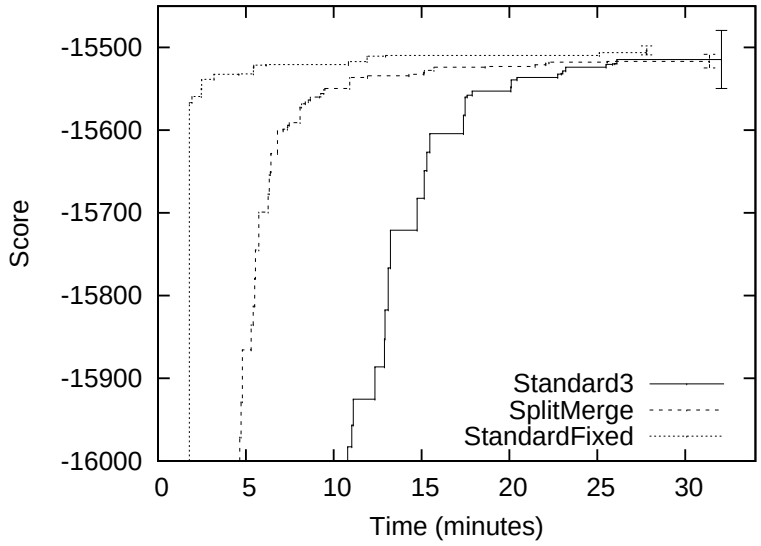

Figure 6.6: Score change during time for "Image" data set.

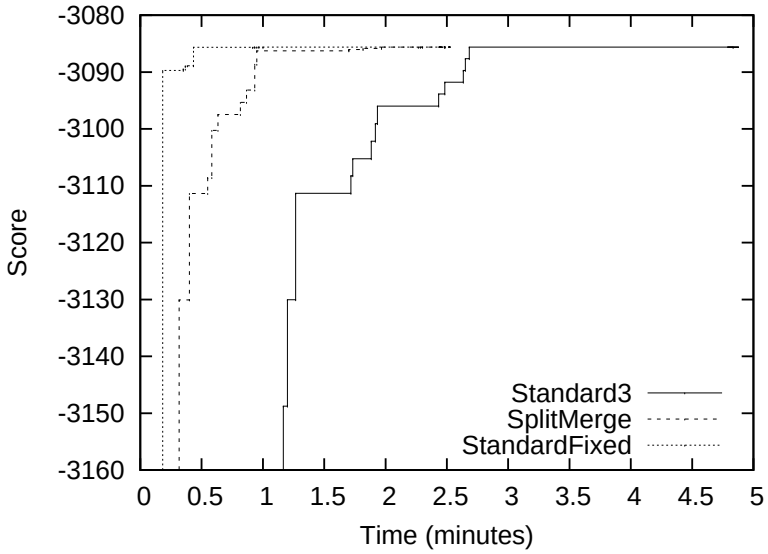

Figure 6.7: Score change during time for "Voting" data set.

6.3.4 Discussion

First, we look at the results for synthetic data. In the first series of tests (Tables 6.1-6.7), *SplitMerge* was better than the standard algorithms for 3:3x3, 10:7x2, and 10:10x2 models, that is when the number of possible configurations of the observed variables (i.e., of $D_1 \times \ldots \times D_k$) was around 1000 or smaller. For small $|\mathbf{D}|$ the performance of different algorithms was similar, but for $|\mathbf{D}|$ around 10^5 and higher *SplitMerge* found higher scoring models than the standard algorithms. Even though the running time of *SplitMerge* was higher than of *Standard3*, the comparison with *StandardFixed* shows that *SplitMerge* was really better. For 10:15x2, 10:10x3, and 5:20x2 models (i.e., when the number of possible configurations of the observed variables was around 30000 or larger) the performance of different algorithms was similar, but for large $|\mathbf{D}|$ *Standard3* increased $|H|$ a little faster than *SplitMerge*. Because of this, for three data sets *Standard3* found significantly better scoring models than *SplitMerge*.[9] But as seen from Table 6.7, the performance of the algorithms was similar when they ran until they terminated themselves. One special case is $|\mathbf{D}| = 10^4$ in Table 6.5, where the standard algorithms found better scoring models than *SplitMerge* and the algorithms terminated themselves. There, *SplitMerge* simply terminated in one of its five runs with a 6–component model, because none of random splits led up to a better model. But as seen from Table 6.10, *SplitMerge* performed the same as the standard algorithms for other random parameterisations of the 10:10x3 structure with $|\mathbf{D}| = 10^4$.

In the second series of tests (Table 6.8) we see a similar pattern as in the first series: *SplitMerge* was better than the standard algorithms when the number of possible configurations of the observed variables was around 5000 (model 10:2,3,4,5,6,7) or smaller, and the performance of the algorithms was similar otherwise (with *Standard3* increasing $|H|$ a little faster than *SplitMerge* for the 10:15x2 model).

In the third series of tests (Tables 6.9 and 6.10) we see that different random parameterisations of the same generative structure generally did not make a big difference in the performance of the algorithms.

In the fourth series of tests (Table 6.11), *SplitMerge* was better than the standard algorithms independently of the number of components in a generative m:10x2 model (with the exception of the simplest case when $m = 2$). Again the running time of *SplitMerge* was higher than of *Standard3*, but the comparison with *StandardFixed* shows that *SplitMerge* was really better.

Now we discuss the results of the tests where m in *StandardFixed* was

[9]Even though for $|\mathbf{D}| = 10^5$ in Table 6.4 $|H|$ for *SplitMerge* is the same as $|H|$ for *Standard3*, the final model contains only 6 components in most of the runs of *SplitMerge*.

equal to the true $|H|$. We consider it normal that for those data sets where $|H|$ given by *SplitMerge* was lower than the true $|H|$, setting m to the true $|H|$ decreased the mean score for *StandardFixed*. This happened because $|\mathbf{D}|$ was probably not big enough to support models with more components. However, we were surprised that setting m to the true $|H|$ decreased the mean score for *StandardFixed* for two last data sets from Tables 6.1 and 6.2 and the last data set from Table 6.3 as well. There, $|H|$ given by *SplitMerge* was higher than the true $|H|$, that is *SplitMerge* overestimated the number of components. But it turns out that this overestimation was "reasonable" in a sense that *StandardFixed* also prefers models with too many components.[10] We think that such a behaviour was caused by the local maxima problem when estimating model parameters. It seems that increasing the number of parameters by having too many components allows the EM algorithm to find better local maximum parameters.

As seen from Figures 6.1-6.3, *Standard3* increases the model score a little faster than *SplitMerge* in the beginning, but in Figures 6.1 and 6.3 *SplitMerge* finds higher scoring models at the end. In Figure 6.2, *Standard3* stops with higher scoring models, but for $|H| = 5$ both *Standard3* and *SplitMerge* found the same scoring models as *StandardFixed*.

We give a summary of our results for synthetic data in Table 6.16. Here we show how did *SplitMerge* perform in comparison with *Standard3* and *StandardFixed* for given data set size ($|\mathbf{D}|$) and number of possible configurations of the observed variables ($|D_1 \times \ldots \times D_k|$).

$\|\mathbf{D}\|$	$\|D_1 \times \ldots \times D_k\|$	
	Around 5000 or lower	Around 30000 or higher
$< 10^5$	The algorithms performed similarly.	The algorithms performed similarly.
$\geq 10^5$	*SplitMerge* performed better than the standard algorithms.	The algorithms performed similarly, but *Standard3* was a little faster than *SplitMerge*.

Table 6.16: A summary of the results for synthetic data.

Now we look at the results for real data. As seen from Table 6.13, the results depend on the data set size. For all 9 data sets of size higher than 4500, *SplitMerge* found better scoring models than *Standard3*. For all 11 data sets of size lower than 4500, *Standard3* found better scoring models than *SplitMerge* (for 3 data sets) or the scores were similar (for 8 data sets).

[10] We made sure that the BIC score is not misleading by checking that for model M_1 with true $|H|$ and model M_2 with too high $|H|$ we have $KL(P_{M_2}, P_G) < KL(P_{M_1}, P_G)$ when $BIC(M_2) > BIC(M_1)$, where G is the generative model.

When $|\mathbf{D}| > 4500$, *SplitMerge* found significantly better scoring models than *Standard3* for 6 data sets. For most of these data sets, the algorithms stopped because of the 5 hour time limit. But contrary to the results for synthetic data, there is a big difference in $|H|$ values of *SplitMerge* and of *Standard3* when the algorithms stop because of the time limit. So, *SplitMerge* increased the cardinality faster then *Standard3*. We have also performed the experiments where the number of starting points for EM in *Standard3* was set to 16 rather than 64. This way, *Standard3* increased the cardinality faster than previously at a cost of worse parameters. In this setup, *SplitMerge* found significantly better scoring models than *Standard3* for 6 data sets again. Also, the results in Table 6.14 indicate that letting the algorithms run until they terminate themselves does not cause *SplitMerge* to perform the same as the standard algorithms. When $|\mathbf{D}| < 4500$, *Standard3* found significantly better scoring models than *SplitMerge* for 1 data set.

As for *StandardFixed*, on average it performed a little better than *SplitMerge*.

For some data sets, the highest scoring models found by *Standard3* and by *SplitMerge* have different $|H|$. For those 6 data sets where *SplitMerge* found significantly better scoring models than *Standard3*, $|H|$ determined by *SplitMerge* is much higher than $|H|$ determined by *Standard3*. For "Pen" and "Shuttle", *Standard3* was simply unable to increase $|H|$. For "Adult", "Mushroom", "Satimage", and "Spambase", *SplitMerge* was faster than *Standard*, and both algorithms were stopped because of the time limit. For 4 more data sets ("Image", "Letter", "Credit", and "Thyroid"), $|H|$ determined by *Standard3* and by *SplitMerge* differ by 1. In all of them, the best model found by *Standard* has a higher score than the best model found by *SplitMerge*, even though for "Letter" and "Thyroid" *SplitMerge* performed better on average. This is related to the fact that generally the variance of the score is lower for *SplitMerge* than for *Standard3*.

The results in Table 6.15 show that the parameter adjustment operation is useful. For 7 data sets, it allowed to find better scoring models. For 4 out of those 7 data sets, it allowed to find significantly better scoring models. For 3 data sets ("Page", "Housing", and "Vehicle"), it led to different $|H|$, which quite interestingly was always lower by 1 than $|H|$ determined by *SplitMerge-NoAdjust*. It was because after the parameter adjustment *SplitMerge* was able to increase the score by performing the state merging operation. Only for 2 data sets ("Audiology" and "Thyroid"), using the parameter adjustment did not cause the score to improve while at the same time increasing "t" a little.

As for the component merging operation, it caused the score to improve for 7 out of 16 data sets where it has been tried (for 4 out of 5, when considering only the data sets of size higher than 4500). So, the component

merging seems to be useful as well.

As seen from Figures 6.4-6.7, models found by *SplitMerge* most of the time have higher score than models found by *Standard3*. And usually, towards the end *SplitMerge* gets close to *StandardFixed*.

Similar results for real data have been obtained in the experiments from Section 4.5. It may seem surprising that compared to the experiments from that section, where only the component splitting has been used, the parameter reusing algorithm does not show more improvement over the standard algorithms. However, the setup of the experiments from Section 4.5 and of the experiments here is quite different.

Summing up the results for both synthetic and real data, on average *SplitMerge* performed better than *Standard3*. It seems that *SplitMerge* becomes more useful as $|\mathbf{D}|$ increases. Another advantage of *SplitMerge* is that the variance of the score for it is lower than for *Standard3*. *SplitMerge* performed a little better than *StandardFixed* on synthetic data and a little worse on real data. Since *StandardFixed* has the advantage of knowing the number of components, we consider this to be a good result.

Of course, very often *SplitMerge* still does not find the global maximum parameters. Indications of this are the situations where the variance of the score for *SplitMerge* is not zero or models found by *SplitMerge* have lower score than models found by *StandardFixed*.

Chapter 7

Learning of Hierarchical Latent Class Models

In this chapter we extend our algorithm that learns LC models to hierarchical latent class (HLC) models. Here we assume that the skeleton of an HLC model is fixed (that is, all the variables and edges between them are fixed) and the goal is to determine the cardinalities of hidden variables and model parameters.

First, we extend the component splitting, component merging, and parameter adjustment operations to HLC models. Then we describe the algorithms that learn the cardinalities and parameters in HLC models by either using these extended operations or standard starting points for EM. Finally, we present experiments comparing the performance of these algorithms.

7.1 Parameter Reusing for HLC Models

In this section, we describe how parameters can be reused in HLC models. We reuse them by extending the component splitting, component merging, and parameter adjustment operations from Definitions 4.2 (page 22), 5.2 (page 50), and 6.1 (page 57). The extensions are quite straightforward: each operation is performed on the root node in an HLC model. Below we give formal definitions. For an HLC model M, $LC(M)$ will denote the LC model consisting of the root node of M and all the child nodes of the root, with the probability distributions for these nodes being the same as in M.

Definition 7.1 *We say that HLC model M^* is obtained from HLC model M by* splitting *a component if $LC(M^*)$ is obtained from $LC(M)$ by splitting a component and the rest of M^* and M are identical.*

Definition 7.2 *We say that HLC model M^* is obtained from HLC model M by* merging *two components if $LC(M^*)$ is obtained from $LC(M)$ by merging two components and the rest of M^* and M are identical.*

Definition 7.3 *We say that HLC model M' is obtained from HLC model M by* parameter adjustment *if there exists HLC model M^* such that:*

- *M^* is obtained from M by splitting a component and then running EM, and M' is obtained from M^* by merging two components and then running EM,*
  or
- *M^* is obtained from M by merging two components and then running EM, and M' is obtained from M^* by splitting a component and then running EM.*

Here the EM algorithm updates all the parameters of an HLC model.[1]

All these three operations work with the root node of an HLC model. When performing parameter reusing for a non-root node, first we do root walking until the node of interest becomes the root and then we work with this new root. So, by using the fact that root walking leads to an equivalent model, we restrict ourselves to working with the root node and this way avoid the problem of considering the parents of a node when changing its cardinality.

We do not know about theoretical properties (like those from Sections 4.3 and 5.3) of these operations for HLC models.

7.2 Algorithms

In this section, we discuss the algorithms that will be used in our experiments on learning the cardinalities and parameters in HLC models.

7.2.1 Algorithm Based on Splitting and Merging

In this section, we provide our algorithm that learns the cardinalities and parameters in HLC models by using the component splitting, component

[1]We do not define the parameter adjustment for HLC models by simply requiring $LC(M')$ to be obtained from $LC(M)$ by adjusting the parameters, because we allow the EM algorithm to update all the parameters of an HLC model, not only those attached to the root and its child nodes.

merging, and parameter adjustment operations. Algorithm *HLCSplitMerge* takes training data $\mathbf{D}$ and model skeleton $\mathbf{s}$ as input and returns an HLC model. It tries to find an HLC model that is the best according to the function *score*, which can be any penalised likelihood scoring function. The pseudocode of *HLCSplitMerge* is given in page 84. This algorithm is very similar to the *SplitMerge* algorithm (page 59). Only steps 1, 7, 20, and 31 are changed. In step 1, the algorithm starts with the HLC model L where the cardinality of each hidden variable is one and the maximum likelihood parameters are deterministically computed from $\mathbf{D}$. In steps 7, 20, and 31, the procedures *HLCSplit*, *HLCMerge*, and *HLCAdjust* instead of the corresponding procedures for LC models are called. We will now describe these three procedures.

The procedure *HLCSplit* increments the cardinality of one hidden variable from an HLC model L. The pseudocode is given in page 85. The procedure increments the cardinality of each hidden variable alone by making that variable the root, splitting a component as described in Definition 7.1, and running the EM algorithm. The model where incrementing the cardinality gave the highest score is taken as the final one.

The procedure *HLCMerge* decrements the cardinality of one hidden variable from an HLC model L. The pseudocode is given in page 85. The procedure decrements the cardinality of each hidden variable alone by making that variable the root, merging two components as described in Definition 7.2, and running the EM algorithm. The model where decrementing the cardinality gave the highest score is taken as the final one.

The procedure *HLCAdjust* tries to increase the score of an HLC model L by performing the parameter adjustment operation. The pseudocode is given in page 86. The procedure considers hidden variables one by one by making a hidden variable to be the root and then trying to improve model parameters similarly as in *Adjust* for LC models from page 62. The procedure terminates without considering other hidden variables as soon as a model giving an increase in score higher than a positive parameter δ is found. Similarly as in *Adjust*, models L' in step 3 and L_s in step 10 are not computed directly but taken from the last run of *HLCMerge* and the last run of *HLCSplit*.

We can see that the algorithm *HLCSplitMerge* allows to have hidden variables with cardinality 1. This is different from the approach of Zhang (2002); Zhang (2004), where the cardinality of a hidden variable can not be lower than 2. We allow the cardinality to be 1, because this provides a natural outset for the component splitting and because the algorithm can indicate that a hidden node is not needed by making its cardinality 1 in the final model. Strictly speaking, a model containing a hidden node with cardinality 1 is no longer an HLC model, because it is equivalent to the model where this

Procedure 7.1 *HLCSplitMerge*$(\mathbf{D}, \mathbf{s})$

Let L be the HLC model with skeleton $\mathbf{s}$ and the cardinality of each hidden variable being one.
$doPhase1 \leftarrow true,\ doPhase2 \leftarrow true.$
loop
  if *doPhase1* **then**
    $L_0 \leftarrow L.$
    repeat
      $L' \leftarrow HLCSplit(L, \mathbf{D}).$
      if $score(L') > score(L)$ **then**
        $L \leftarrow L'.$
      end if
    until $L \neq L'.$
    if $score(L) > score(L_0)$ **then**
      $doPhase2 \leftarrow true.$
    end if
    $doPhase1 \leftarrow false.$
  end if
  if *doPhase2* **then**
    $L_0 \leftarrow L.$
    repeat
      $L' \leftarrow HLCMerge(L, \mathbf{D}).$
      if $score(L') > score(L)$ **then**
        $L \leftarrow L'.$
      end if
    until $L \neq L'.$
    if $score(L) > score(L_0)$ **then**
      $doPhase1 \leftarrow true.$
    end if
    $doPhase2 \leftarrow false.$
  end if
  if not *doPhase1* and not *doPhase2* **then**
    $L' \leftarrow HLCAdjust(L, \mathbf{D}).$
    if $score(L') > score(L)$ **then**
      $L \leftarrow L',\ doPhase1 \leftarrow true,\ doPhase2 \leftarrow true.$
    else
      return $L.$
    end if
  end if
end loop

Procedure 7.2 *HLCSplit*$(L, \mathbf{D})$

1: **for each** hidden variable H_i of L **do**
2: Obtain from L the HLC model L^* with H_i as the root by doing root walking.
3: $L_i \leftarrow$ *Split*$(L^*, \mathbf{D})$, where the procedure *Split* is performed as in page 60, but in each step (producing two-component models, data $\mathbf{D}_s$, model L_s, running EM) working with HLC rather than LC models.
4: **end for**
5: $L' \leftarrow \arg\max_{L_i}$ *score*(L_i).
6: **return** L'.

Procedure 7.3 *HLCMerge*$(L, \mathbf{D})$

1: **for each** hidden variable H_i of L **do**
2: Obtain from L the HLC model L^* with H_i as the root by doing root walking.
3: $L_i \leftarrow$ *Merge*$(L^*, \mathbf{D})$, where the procedure *Merge* is performed as in page 61, but in each step (producing model L^*, running EM) working with HLC rather than LC models.
4: **end for**
5: $L' \leftarrow \arg\max_{L_i}$ *score*(L_i).
6: **return** L'.

Procedure 7.4 *HLCAdjust*$(L, \mathbf{D})$

1: **for each** hidden variable H_i of L **do**
2: Obtain from L the HLC model L^* with H_i as the root by doing root walking.
3: $L' \leftarrow Merge(L^*, \mathbf{D})$, with *Merge* performed as in step 3 of *HLCMerge*.
4: $L'' \leftarrow Split(L', \mathbf{D})$, with *Split* performed as in step 3 of *HLCSplit*.
5: **if** $score(L'') > score(L) + \delta$ **then**
6: **return** L''.
7: **else**
8: $\mathcal{L} \leftarrow \emptyset$.
9: **for each** component h_s of L^* **do**
10: Obtain an HLC model L_s as in the procedure *Split* from page 60, but in each step (producing two-component models, data $\mathbf{D}_s$, model L_s, running EM) working with HLC rather than LC models.
11: **for each** pair of components $\{h_a, h_b\}$ where h_a is a new component and h_b is an old component of L_s **do**
12: Obtain model L' by merging h_a and h_b in L_s.
13: Add L' to $\mathcal{L}$.
14: **end for**
15: **end for**
16: Obtain model L'' by running the multiple restart EM with $\mathcal{L}$ as starting points and data $\mathbf{D}$.
17: **if** $score(L'') > score(L) + \delta$ **then**
18: **return** L''.
19: **end if**
20: **end if**
21: **end for**
22: **return** L.

hidden node and the adjacent edges are removed. Such a model partitions the observed variables into groups, where variables in different groups are assumed to be independent of each other.

Another difference from the approach of Zhang (2002); Zhang (2004) is that we do not check whether the obtained models are regular. We allow the cardinality of a hidden variable to be higher than required in the definition of a regular structure, because this may allow to perform the component merging later.

7.2.2 Algorithms Based on Standard Starting Points

Similarly as in the experiments with LC models, the algorithm *HLCSplitMerge* will be compared with the algorithms that use standard starting points for EM. One of them, called *HLCStandard3*, has almost the same main procedure as *HLCSplitMerge* (see page 84). *HLCStandard3* also has 3 phases: increasing cardinalities, decreasing cardinalities, and improving parameters. The difference from *HLCSplitMerge* is that instead of calling *HLCSplit*, *HLCMerge*, and *HLCAdjust* it calls *HLCIncrement*, *HLCDecrement*, and *HLCNewParams*, the pseudocodes for which are given below.

HLCIncrement (page 87) increments the cardinality of each hidden variable alone and estimates model parameters by running EM from random starting points. The model where incrementing the cardinality gave the highest score is taken as the final one. In the same pattern, *HLCDecrement* (page 88) decrements the cardinality of each hidden variable alone. *HLCNewParams* (page 88) simply runs EM from random starting points. In all these three procedures, $|\mathcal{M}|$ is specified in advance and random parameters are sampled from the uniform distribution.

Procedure 7.5 *HLCIncrement*$(L, \mathbf{D})$

1: **for each** hidden variable H_i of L **do**
2: Let $\mathbf{m}$ be the structure of L, where the cardinality of H_i is increased by 1.
3: Produce a set $\mathcal{M}$ of HLC models having structure $\mathbf{m}$ and random parameters.
4: Obtain model L_i by running the multiple restart EM with $\mathcal{M}$ as starting points.
5: **end for**
6: $L' \leftarrow \arg\max_{L_i} score(L_i)$.
7: **return** L'.

Procedure 7.6 *HLCDecrement*$(L, \mathbf{D})$

1: **for each** hidden variable H_i of L where $|H_i| > 1$ **do**
2: Let $\mathbf{m}$ be the structure of L, where the cardinality of H_i is decreased by 1.
3: Produce a set $\mathcal{M}$ of HLC models having structure $\mathbf{m}$ and random parameters.
4: Obtain model L_i by running the multiple restart EM with $\mathcal{M}$ as starting points.
5: **end for**
6: $L' \leftarrow \arg\max_{L_i} score(L_i)$.
7: **return** L'.

Procedure 7.7 *HLCNewParams*$(L, \mathbf{D})$

1: Let $\mathbf{m}$ be the structure of L.
2: Produce a set $\mathcal{M}$ of HLC models having structure $\mathbf{m}$ and random parameters.
3: Obtain model L' by running the multiple restart EM with $\mathcal{M}$ as starting points.
4: **if** $score(L') > score(L) + \delta$ **then**
5: **return** L'.
6: **else**
7: **return** L.
8: **end if**

The algorithm *HLCStandardFixed* uses standard starting points for EM and estimates HLC model parameters when the model structure is fixed. Its pseudocode is given in page 89. In the same way as *StandardFixed* for LC models, it repeatedly tries to improve model parameters by running the multiple restart EM from random starting points. The algorithm is stopped when it reaches a time limit, and then L contains the best model. Parameters of models from $\mathcal{M}$ are again sampled from the uniform distribution.

Procedure 7.8 *HLCStandardFixed*$(\mathbf{D}, \mathbf{m})$

1: Let L be an HLC model having structure $\mathbf{m}$ and random parameters.
2: $N \leftarrow N_0$.
3: **loop**
4: Produce a set $\mathcal{M}$ of N models, where each model has structure $\mathbf{m}$ and random parameters.
5: Obtain model L' by running the multiple restart EM with $\mathcal{M}$ as starting points.
6: **if** $score(L') > score(L)$ **then**
7: $L \leftarrow L'$.
8: **end if**
9: $N \leftarrow 2N$.
10: **end loop**

7.3 Experiments

In this section, we describe the experiments where we compare the approach that uses the component splitting and merging (algorithm *HLCSplitMerge*) with the approach that uses standard starting points (algorithms *HLCStandard3* and *HLCStandardFixed*) for learning cardinalities and parameters in HLC models. First, we describe the setup of experiments. Then we give the results for synthetic and for real data. Finally, we discuss the results.

7.3.1 Setup of Experiments

The setup of experiments is the same as for LC models, described in Section 6.3.1. In *HLCIncrement*, *HLCDecrement*, and *HLCAdjust*, $|\mathcal{M}| = 64$. Contrary to the experiments with LC models, the algorithms are not stopped before they terminate themselves. The model structure $\mathbf{m}$ in *HLCStandardFixed* is the structure of the best scoring model found by *HLCSplitMerge* for

a given data set. The running time of *HLCStandardFixed* is equal to the mean running time of *HLCSplitMerge* for a given data set.

The results are displayed in the same way as for LC models, except that in the field "hidden" the cardinalities of all the hidden variables in the best scoring model are shown (the cardinalities of the root, its children, and its grandchildren are separated by colons).

7.3.2 Results for Synthetic Data

In these experiments, training data $\mathbf{D}$ is always sampled from some HLC model. Forward sampling is used and $\mathbf{D}$ over observed variables is always complete. The generative HLC model is always parameterised randomly, with its parameters sampled from the uniform distribution. The skeleton of the generative HLC model is provided as a parameter to *HLCSplitMerge* and *HLCStandard3*.

In the first series of tests, we sampled $\mathbf{D}$ of size 10^4, 10^5, and 10^6 from the same HLC model having the structure shown in Figure 7.1. The cardinality of each observed variable is 3. This structure is taken from Zhang (2004). Here and in further figures, hidden variables are labelled with their cardinalities.

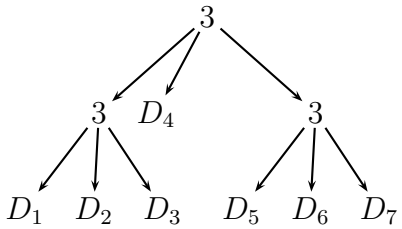

Figure 7.1: Structure with 7 observed variables.

We performed the experiments with two different parameterisations of the structure from Figure 7.1. The first model was parameterised as usually. The second model was also parameterised as usually, but it was ensured that each conditional probability distribution contains an element no smaller than 0.6. Also, it was ensured that in each conditional probability table, the largest elements for different states of the child variable do not all correspond to the same state of the parent variable. The idea of such a setup is taken from Zhang (2004) and Zhang and Kočka (2004). The results of tests are shown in Tables 7.1 and 7.2.

In the second series of tests, we sampled $\mathbf{D}$ of size 10^4 from HLC models having the structures shown in Figures 7.2 and 7.3. The cardinality of each observed variable is again 3. These structures are taken from

$\|\mathbf{D}\|$	*HLCStandard3*			*HLCSplitMerge*			*HLCStandardFixed*
	score	t	hidden	score	t	hidden	score
10^4	-71381.2 ± 2.6	58	2:3,2	-71380.3 ± 0.0	35	2:3,2	**-71378.6 ± 1.1**
10^5	-712205.8 ± 5.8	71	2:3,2	-712200.7 ± 0.0	41	2:3,2	-712200.4 ± 1.3
10^6	-7121968.5 ± 85.6	128	2:3,3	-7121953.3 ± 71.8	148	2:3,3	-7121863.8 ± 18.1

Table 7.1: Sampled from the first model with 7 observed variables.

$\|\mathbf{D}\|$	*HLCStandard3*			*HLCSplitMerge*			*HLCStandardFixed*
	score	t	hidden	score	t	hidden	score
10^4	-65824.5 ± 0.3	62	3:3,3	-65824.4 ± 0.0	43	3:3,3	-65824.1 ± 0.3
10^5	-656733.5 ± 6.8	118	3:3,3	**-656722.2 ± 0.7**	158	3:3,3	-656721.4 ± 0.8
10^6	-6557966.9 ± 47.8	186	3:3,3	**-6557889.6 ± 0.0**	413	3:3,3	**-6557887.5 ± 0.9**

Table 7.2: Sampled from the second model with 7 observed variables.

Zhang and Kočka (2004). Models were parameterised in the same way as the second model from the first series of tests. The results of tests are

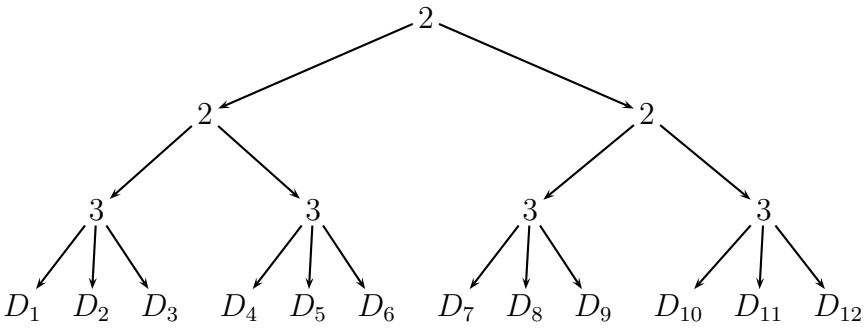

Figure 7.2: Structure with 12 observed variables.

shown in Table 7.3. The field "t" indicates the rounded mean running time in hours. For the 18-variable model, each algorithm was run only 2 times because of long running time.

Similarly as in the experiments with LC models, we display how the mean score changes during time for several data sets. It is shown in Figures 7.4-7.8 (bars at the end indicate 95% confidence intervals for the mean).

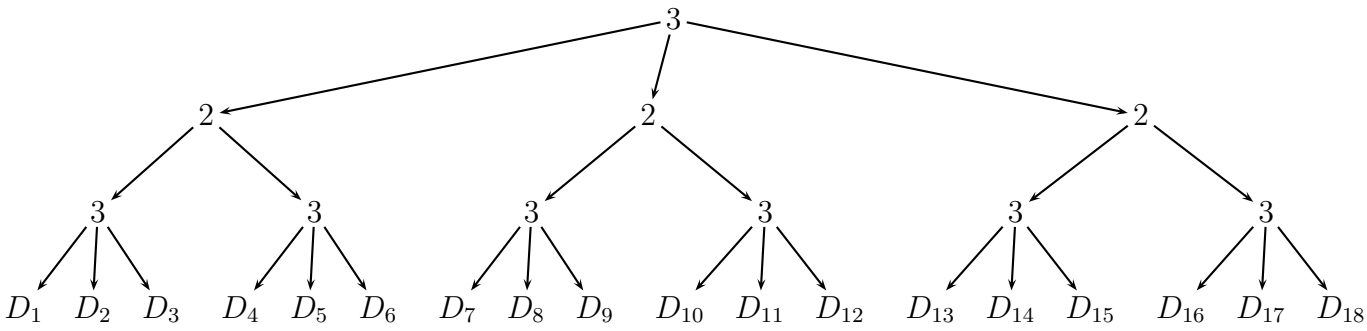

Figure 7.3: Structure with 18 observed variables.

k	*HLCStandard3*			*HLCSplitMerge*			*HLCStandardFixed*
	score	t	hidden	score	t	hidden	score
12	-108271.5 ± 52.8	77	1:2,2: 3,3,3,3	-108302.8 ± 0.0	29	1:1,2: 2,2,3,3	-108301.8 ± 0.1
<u>18</u>	**-166025.7 ± 0.3**	414	2:2,2,2: 3,3,3,3,3,3	<u>-166031.2 ± 0.0</u>	289	2:2,2,2: 2,3,3,3,3,3	**-166030.1 ± 0.1**

Table 7.3: Sampled from models with 12 and with 18 observed variables.

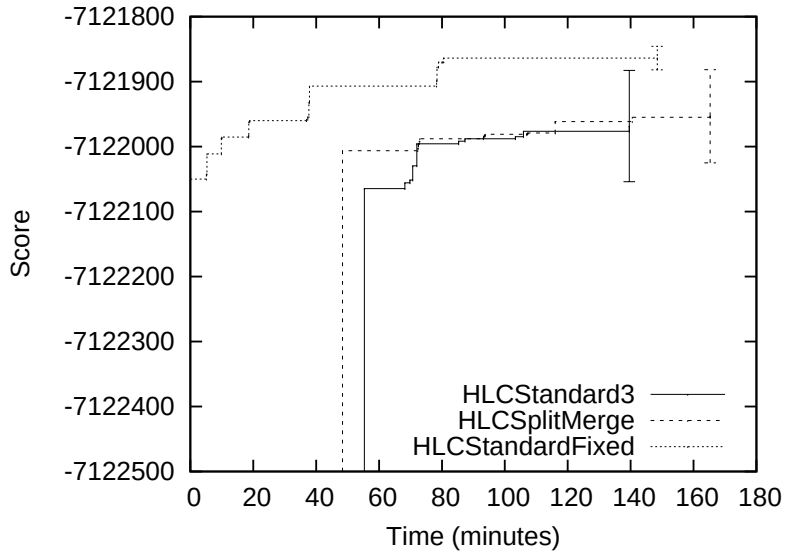

Figure 7.4: Score change during time for data sampled from the first model with 7 observed variables, $|\mathbf{D}| = 10^6$.

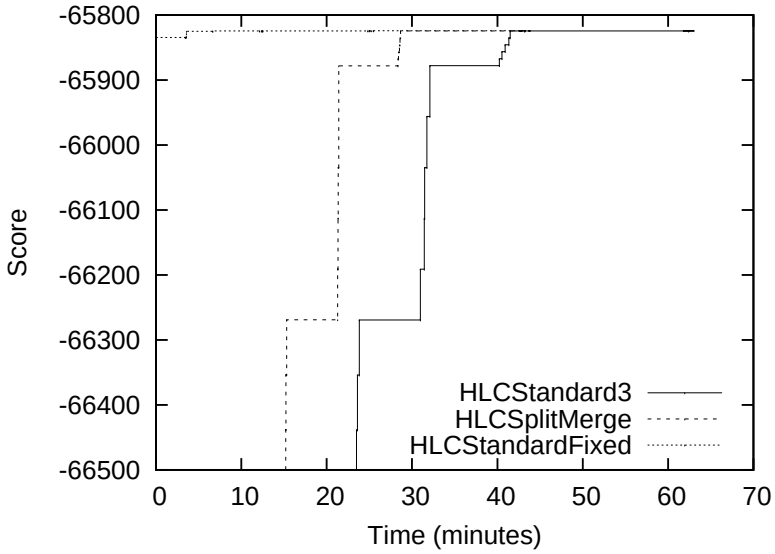

Figure 7.5: Score change during time for data sampled from the second model with 7 observed variables, $|\mathbf{D}| = 10^4$.

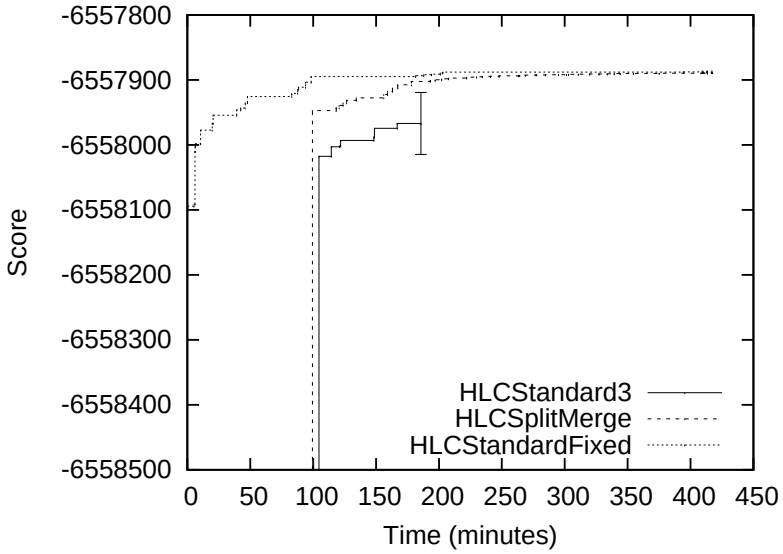

Figure 7.6: Score change during time for data sampled from the second model with 7 observed variables, $|\mathbf{D}| = 10^6$.

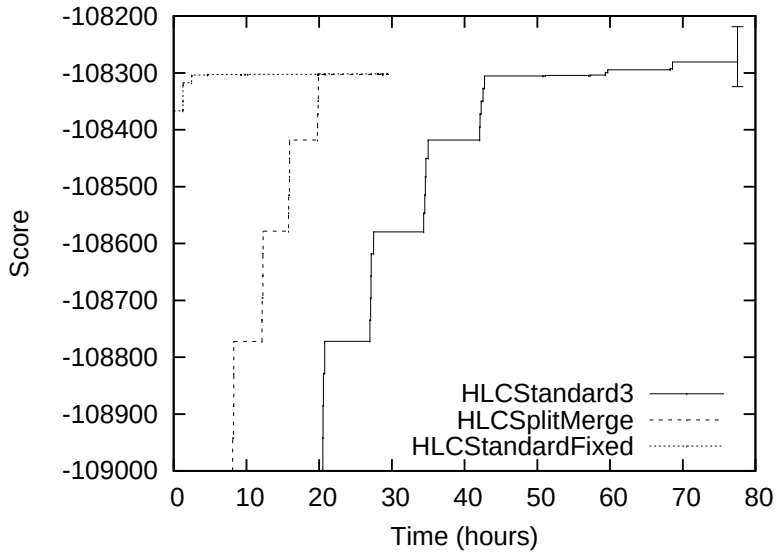

Figure 7.7: Score change during time for data sampled from the model with 12 observed variables.

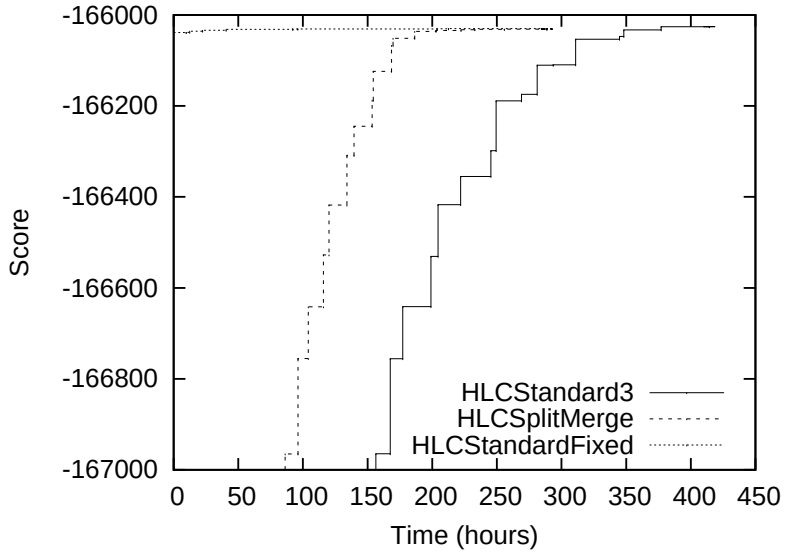

Figure 7.8: Score change during time for data sampled from the model with 18 observed variables.

7.3.3 Results for Real Data

For training data, we used the CoIL Challenge 2000 (van der Putten and van Someren (eds), 2000) data, preprocessed as described in Zhang and Kočka (2004). The preprocessed data set contains 42 variables and has size 5822. Most of the variables are binary. The structure of the best HLC model for this data set found by the algorithm of Zhang and Kočka (2004) is shown in Figure 7.9. For hidden variables, the name of a variable and its cardinality are shown. For observed variables, the name of a variable is shown.

The skeleton from Figure 7.9 was provided as an input to *HLCSplitMerge* and *HLCStandard3*. The results of tests are shown in Table 7.4. The field "t" indicates the rounded mean running time in hours. Each algorithm was run only 2 times because of long running time. The cardinalities of hidden

HLCStandard3		*HLCSplitMerge*		*HLCStandardFixed*
score	t	score	t	score
-51626.1 ± 7.4	971	**-51516.5 ± 19.4**	467	-51515.1 ± 30.1

Table 7.4: Results for CoIL data set.

variables in the final models found by *HLCSplitMerge* and *HLCStandard3* were the same as the cardinalities in Figure 7.9 except for the following differences. For *HLCStandard3*, the cardinalities of h8, h20, and h21 were lower by 1 in the first run (model score -51621.8), and the cardinalities of h0, h20, and h21 were lower by 1 in the second run (model score -51630.3). For *HLCSplitMerge*, the cardinality of h21 was higher by 1 in the first run (model score -51527.5), and all the cardinalities were the same as in Figure 7.9 in the second run (model score -51505.5). The scores of the final models found by *HLCStandardFixed* were -51497.9 and -51532.2.

In Figure 7.10, we display how the mean score changes during time (bars at the end indicate 95% confidence intervals for the mean).

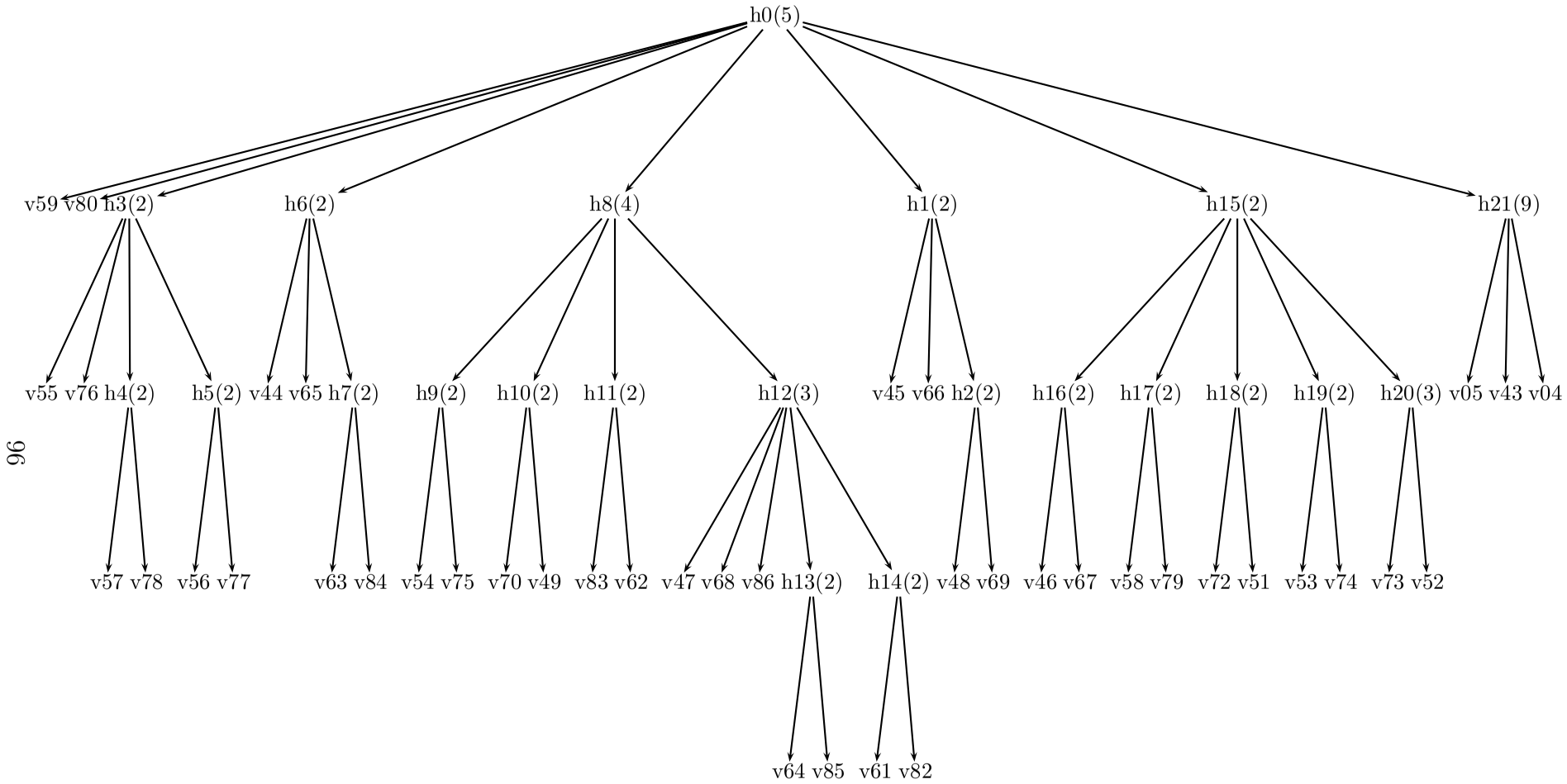

Figure 7.9: HLC model structure for CoIL data set.

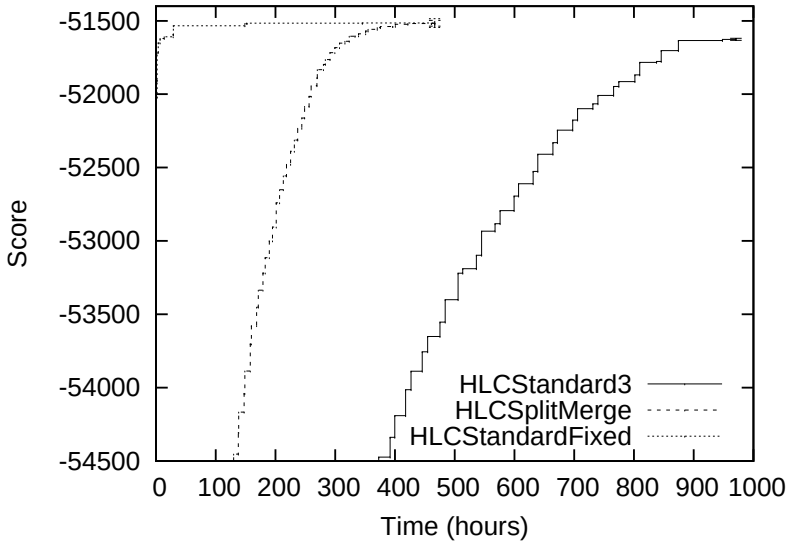

Figure 7.10: Score change during time for CoIL data set.

7.3.4 Discussion

First, we look at the results for synthetic data. For data with 7 observed variables (Tables 7.1 and 7.2), the final models found *HLCSplitMerge* have higher score than the final models found by *HLCStandard3*, especially for $|\mathbf{D}|$ higher than 10^4. The structures of the best final models found by both algorithms are the same. For data with 12 and with 18 observed variables (Table 7.3), the results are the opposite: the final models found by *HLCSplitMerge* have lower score than the final models found by *HLCStandard3*. This is because *HLCStandard3* was able to increase the cardinalities more and this way the structure of its final best model is closer to the structure of the generative model (as seen from the fields "hidden" in Table 7.3).

When comparing *HLCSplitMerge* with *HLCStandardFixed*, we see that the average final score was always higher for *HLCStandardFixed* (and sometimes it was even significantly better), but the difference in the average final scores was really big for one data set only (namely, the last one from Table 7.1). So, usually the parameters found by *HLCSplitMerge* were only a little worse than those find by *HLCStandardFixed* for given HLC model structures.

Figures 7.7 and 7.8 show that for the two data sets where *HLCSplitMerge* found worse models than *HLCStandard3*, *HLCSplitMerge* was about two times faster, and only because of longer running time *HLCStandard3* found better models.

Now we look at the results for real data. As seen from Table 7.4 and Figure 7.10, *HLCSplitMerge* performed much better than *HLCStandard3*. And as described at the end of Section 7.3.3, the structures found by *HLCSplitMerge* are closer to the structure from Figure 7.9. Having in mind the results for synthetic data, it is interesting that this time *HLCSplitMerge* ended up with higher cardinalities for some variables than *HLCStandard3*. As for the *HLCStandardFixed* algorithm, we see that it finds similarly scoring models as *HLCSplitMerge*: for the structure where the model score was -51505.5 for *HLCSplitMerge*, it was -51497.9 and -51532.2 in the two runs of *HLCStandardFixed*. It is interesting to compare that the BIC score of the best model found by the HSHC algorithm (described in Section 3.4.2) of Zhang and Kočka (2004) was -51465.[2] It took 121 hours to learn this model using a processor that has a similar rate as in our experiments. So, the HSHC algorithm, which learns the model skeleton as well, learned better parameters than *HLCStandardFixed* and *HLCSplitMerge*. The reason for this may be that in HSHC running local EM rather than EM that updates all the

[2]We remind that this model has the same structure as the one discovered by *HLCSplitMerge* and used by *HLCStandardFixed*.

parameters of a model causes not only to save time but also to find better parameters. We believe that *HLCSplitMerge* can be improved by running local EM after splitting or merging components, and only after that running EM that updates all the parameters of a model. In local EM, only the probability distributions for the root node and its children would be optimised.

Both in the experiments with synthetic and with real data, the parameter adjustment operation (procedure *HLCAdjust*) often increased the model score. But the component merging operation (procedure *HLCMerge*) increased the model score only once (in one of the runs for real data). The reason for this may be that in our experiments the cardinalities of hidden variables in the optimal HLC models are not high, and in this case the component splitting usually does not overestimate the cardinality.

So, on average *HLCSplitMerge* performed a little better than *HLCStandard3* on synthetic data, and it performed much better than *HLCStandard3* on real data. For synthetic data, the variance of the score for *HLCSplitMerge* was lower than for *HLCStandard3*. For both synthetic and real data, the score of models found by *HLCSplitMerge* was close to the score of models found by *HLCStandardFixed*. As mentioned above, a promising future work would be the introduction of local EM in *HLCSplitMerge*.

Chapter 8

Conclusions and Research Directions

In this thesis, we have addressed the problem of learning Bayesian networks with hidden variables. We have concentrated on a sub-problem of learning the cardinalities of hidden variables and the parameters in latent class (LC) and hierarchical latent class (HLC) models. These models are tree-structured Bayesian networks with categorical variables and are among the simplest types of Bayesian networks with hidden variables. The standard method for learning cardinalities and parameters is to run the EM algorithm from random starting points. We have proposed a method where starting points for EM are obtained by reusing the parameters of the already learned models. When we reuse the parameters by performing the so-called component splitting operation, we increase the cardinality of a hidden variable. When we reuse the parameters by performing the so-called component merging operation, we decrease the cardinality of a hidden variable. By repeatedly performing these operations, we learn both the cardinalities of hidden variables and the parameters at the same time. We have proved that if some data is not described perfectly by some LC model, then with probability 1 it is possible to increase the log-likelihood of that data by splitting a component in that LC model (Theorem 4.2 on page 28). Based on experimental results we have conjectured that if some data over binary variables is described perfectly by an LC model having one component more than the generative model for that data, then the generative model can be obtained by merging two components of that model, provided that the dimension of that model is not higher than the complete dimension of that data (Conjecture 5.2 on page 53).

In the experiments with LC and HLC models, our algorithms based on component splitting and merging in a majority of cases performed better than

the algorithms that use standard starting configurations for EM. In particular, the parameter reusing approach was better for bigger-sized training data. However, we think that our algorithms can still be improved.

One possible improvement could be to introduce criteria for selecting the most promising components for splitting and merging as it has been done by Ueda et al. (2000) for continuous data (Section 3.3). For example, component h_s from model L could be considered for splitting when the distance between the distribution specified by the parameters of the component and the empirical distribution given by $\mathbf{D}_s$ is large. That is, when $KL\big(P_L(D_1|h_s) \cdot \ldots \cdot P_L(D_k|h_s),\ P_{\mathbf{D}_s}(D_1, \ldots, D_k)\big)$ is large. Components h_s and h_t could be considered for merging when the distance between the empirical distributions given by $\mathbf{D}_s$ and $\mathbf{D}_t$ is small.[1]

Another possible improvement would deal with the implementation of component splitting. Currently, we split a component randomly. It may be worth trying to compute from data a good direction for splitting. One may try to improve the deterministic approaches suggested in Section 4.4.2.

For HLC models only, the main improvement should be introduction of local EM, as discussed in Section 7.3.4.

As discussed in Section 3.1.3, the scoring functions we use are not completely suited for networks with hidden variables. It would be interesting to see if using better suited scoring functions would change the performance of the algorithms. The first step here could be using a modified BIC score where effective dimension of a Bayesian network and relabeling of states of hidden variables (see Section 3.1.3) are taken into account.

In this thesis, we first proposed a parameter reusing approach for LC models and after that extended it to HLC models. A natural next step would be to move from tree-structured models to unrestricted Bayesian networks with hidden variables. The main issue here is how to define component (i.e., state) splitting and merging for hidden variables that have parents. The most obvious extension of Definitions 4.2 (page 22) and 5.2 (page 50) would be the following. For a hidden variable H with parents $A_1, \ldots, A_j$ and children $B_1, \ldots, B_k$ in model M, producing of model M^* by *splitting* of state h_s into h_s^1 and h_s^2 would mean setting $P_{M^*}(h_s^1|\mathbf{a}) = P_{M^*}(h_s^2|\mathbf{a}) = \frac{1}{2}P_M(h_s|\mathbf{a})$ for every configuration $\mathbf{a}$ of $A_1, \ldots, A_j$, and setting $P_{M^*}(B_i|h_s^1)$ and $P_{M^*}(B_i|h_s^2)$ to be close to $P_M(B_i|h_s)$, $\forall i = 1, \ldots, k$. Producing of model M^* by *merging* of states h_s^1 and h_s^2 into h_s would mean setting $P_{M^*}(h_s|\mathbf{a}) = P_M(h_s^1|\mathbf{a}) + P_M(h_s^2|\mathbf{a})$ for every configuration $\mathbf{a}$ of $A_1, \ldots, A_j$, and setting $P_{M^*}(B_i|h_s) =$

[1] Measuring the distance between the distributions specified by the parameters of components h_s and h_t could be computationally too expensive because each of these distributions may contain $\prod_{i=1}^{k} |D_i|$ non-zero probabilities.

$\frac{P_M(h_s^1)}{P_{M^*}(h_s)} P_M(B_i|h_s^1) + \frac{P_M(h_s^2)}{P_{M^*}(h_s)} P_M(B_i|h_s^2)$, $\forall i = 1, \ldots, k$, where $P_M(h_s^1)$, $P_M(h_s^2)$, and $P_{M^*}(h_s)$ are marginal probabilities of states of H according to models M and M^*. However, state splitting defined in this way has an undesirable property of working differently for equivalent models. For example, consider the Bayesian networks $A \rightarrow H \rightarrow B$ and $A \leftarrow H \rightarrow B$, where H is hidden and A, B are observed variables. These two models are equivalent. For the first model, the probability distributions $P(A, h_s^1)$ and $P(A, h_s^2)$ after splitting would be identical, while for the second model not. This could be repaired (i.e., the distributions forced to be different for the first model as well) by setting during splitting $P_{M^*}(h_s^1|\mathbf{a}) = \frac{1}{2} P_M(h_s|\mathbf{a}) + \epsilon_{\mathbf{a}}$ and $P_{M^*}(h_s^2|\mathbf{a}) = \frac{1}{2} P_M(h_s|\mathbf{a}) - \epsilon_{\mathbf{a}}$ for every configuration $\mathbf{a}$ of $A_1, \ldots, A_j$, where each $\epsilon_{\mathbf{a}}$ is a real number close to zero. For models containing more than one hidden variable, the easiest way of optimising the cardinalities seems to be the one used in HLC models: increasing or decreasing the cardinality of one variable at a time while keeping the cardinalities of other hidden variables fixed. When testing this parameter reusing approach, the first step could be to perform tests with hierarchical naive Bayes (HNB) models (Zhang et al., 2004), which are like HLC models except that the root variable (i.e., class variable) is observed. HNB models are not as complex a unrestricted Bayesian networks, but at the same time contain hidden variables that have an observed parent.

We reuse the parameters by performing component splitting and merging. As discussed in the beginning of Chapters 4 and 5, the parameters could also be reused by performing component introduction and removal. Working on algorithms that would use these two operations instead of (or in addition to) splitting and merging could be yet another research direction.

A more theoretical research would be further investigation of properties of component merging in LC models.

Concerning practical usage, one could consider another application of the parameter reusing technique. In this thesis, we have assumed a completely automatic learning where the algorithm learns the cardinalities and parameters by using training data alone as input. However, there may be situations where a model for given training data is already available and the goal is to improve the available model rather than to find the model that describes training data the best. In such a case, the parameter reusing approach that would use the available model as the initial one would be very convenient because it would not forget the parameters of the available model while trying to find better models.

Appendix A

Preprocessing of Text Data

In this appendix, we describe how we preprocessed text data, which was used then in the experiments from Section 4.5. The description here is based on pages 29-30 of Karčiauskas (2002).

The Reuters-21578 text categorisation test collection (Distribution 1.0) (Lewis, 1997) is a widely used test collection for text categorisation research. It contains about 20000 documents (Reuters news stories), each of them assigned to zero or more topics. We used the most popular "ModApte" split of this collection into training and test sets. Following the test setup of Yang and Liu (1999), we selected the topics that have at least one document both in the training set and the test set. It resulted in selecting 90 topics. After eliminating documents that do not belong to any of these 90 topics, we got a training set of 7769 documents and a test set of 3018 documents.

When preprocessing documents, for simplicity we did not distinguish between text that appears in the title and text that appears in the body of a document. First, for the extraction of features from a document, we converted text to lowercase and selected from it words (i.e., sequences of alpha symbols delimited by any other symbols). Then we removed function words (i.e. topic-neutral words such as articles, prepositions). After this preprocessing, each document had a number of features – the non-function words appearing in the document. A feature set was then built by taking from the documents in the training set all the features except those that appear only in one document. The feature set consists of 15715 words. As in many other works on text categorisation, we used binary features.

Since for many machine learning algorithms it is computationally impossible to use 15715 features, we performed feature selection. For this, we used the information gain criteria, because it has been reported as one of the most effective by Yang and Pedersen (1997). For each topic, a separate set of the most relevant features was selected.

For the experiments from Section 4.5, we produced 9 different data sets: for topics *Earn*, *Crude*, and *Corn*, with 10, 20, and 30 features for each topic. For these experiments, we merged the training and test sets into one set, giving the data of size 10787.

Bibliography

G. Ball and D. Hall. 1967. A clustering technique for summarizing multivariate data. *Behavioral Science*, 12:153–155.

Y. Barash and N. Friedman. 2001. Context-specific Bayesian clustering for gene expression data. In *RECOMB '01: Proceedings of the fifth annual international conference on Computational biology*, pages 12–21.

E. Bauer, D. Koller, and Y. Singer. 1997. Update rules for parameter estimation in Bayesian networks. In *Uncertainty in Artificial Intelligence: Proceedings of the Thirteenth Conference*, pages 3–13.

M. J. Beal and Z. Ghahramani. 2003. The variational Bayesian EM algorithm for incomplete data: with application to scoring graphical model structures. *Bayesian Statistics*, 7:453–464.

J. Binder, D. Koller, S. J. Russell, and K. Kanazawa. 1997. Adaptive probabilistic networks with hidden variables. *Machine Learning*, 29(2-3):213–244.

C. L. Blake and C. J. Merz. 1998. UCI repository of machine learning databases. http://www.ics.uci.edu/∼mlearn/MLRepository.html.

K. Blekas and A. Likas. 2004. Incremental mixture learning for clustering discrete data. In *Methods and Applications of Artificial Intelligence, Third Helenic Conference on AI, SETN 2004*, pages 210–219.

X. Boyen, N. Friedman, and D. Koller. 1999. Discovering the hidden structure of complex dynamic systems. In *Uncertainty in Artificial Intelligence: Proceedings of the Fifteenth Conference*, pages 91–100.

P. S. Bradley, U. M. Fayyad, and C. A. Reina. 1998. Scaling EM (Expectation-Maximization) clustering to large databases. Technical Report MSR-TR-98-35, Microsoft Research.

M. Brand. 1999. Structure learning in conditional probability models via an entropic prior and parameter extinction. *Neural Computation*, 11(5):1155–1182.

G. Celeux, S. Chrétien, F. Forbes, and A. Mkhadri. 2001. A component-wise EM algorithm for mixtures. *Journal of Computational and Graphical Statistics*, 10(4):699–712.

P. Cheeseman and J. Stutz. 1995. Bayesian classification (AutoClass): Theory and results. In *Advances in Knowledge Discovery and Data Mining*, pages 153–180. AAAI Press.

D. M. Chickering and D. Heckerman. 1997. Efficient approximations for the marginal likelihood of Bayesian networks with hidden variables. *Machine Learning*, 29(2-3):181–212.

D. M. Chickering. 2002. Optimal structure identification with greedy search. *The Journal of Machine Learning Research*, 3:507–554.

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

D. Connolly. 1993. Constructing hidden variables in Bayesian networks via conceptual clustering. In *Machine Learning, Proceedings of the Tenth International Conference*, pages 65–72.

G. F. Cooper and E. Herskovits. 1992. A Bayesian method for the induction of probabilistic networks from data. *Machine Learning*, 9(4):309–347.

G. F. Cooper. 1995. A Bayesian method for learning belief networks that contain hidden variables. *Journal of Intelligent Information Systems*, 4(1):71–88.

A. Dempster, N. Laird, and D. Rubin. 1977. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, 39(B):1–38.

G. Elidan and N. Friedman. 2001. Learning the dimensionality of hidden variables. In *Uncertainty in Artificial Intelligence: Proceedings of the Seventeenth Conference*, pages 144–151.

G. Elidan and N. Friedman. 2003. The Information Bottleneck EM algorithm. In *Uncertainty in Artificial Intelligence: Proceedings of the Nineteenth Conference*, pages 200–208.

G. Elidan, N. Lotner, N. Friedman, and D. Koller. 2000. Discovering hidden variables: A structure-based approach. In *Proceedings of the 13th Annual Conference on Neural Information Processing Systems*, pages 479–485.

G. Elidan, M. Ninio, N. Friedman, and D. Schuurmans. 2002. Data perturbation for escaping local maxima in learning. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence and Fourteenth Conference on Innovative Applications of Artificial Intelligence (AAAI/IAAI)*, pages 132–139.

U. M. Fayyad, C. A. Reina, and P. S. Bradley. 1998. Initialization of iterative refinement clustering algorithms. In *Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining*, pages 194–198.

M. A. T. Figueiredo and A. K. Jain. 2002. Unsupervised learning of finite mixture models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(3):381–396.

J. Fischer and K. Kersting. 2003. Scaled CGEM: A fast accelerated EM. In *Machine Learning: ECML 2003, 14th European Conference on Machine Learning*, pages 133–144.

D. H. Fisher. 1987. Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, 2(2):139–172.

N. Friedman. 1997. Learning belief networks in the presence of missing values and hidden variables. In *Proceedings of the 14th International Conference on Machine Learning*, pages 125–133.

N. Friedman. 1998. The Bayesian structural EM algorithm. In *Uncertainty in Artificial Intelligence: Proceedings of the Fourteenth Conference*, pages 129–138.

D. Geiger, D. Heckerman, and C. Meek. 1996. Asymptotic model selection for directed networks with hidden variables. In *Uncertainty in Artificial Intelligence: Proceedings of the Twelfth Conference*, pages 283–290.

D. Geiger, D. Heckerman, H. King, and C. Meek. 2001. Stratified exponential families: Graphical models and model selection. *Annals of Statistics*, 29(2):505–529.

S. Geman and D. Geman. 1984. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6(6):721–741.

F. Glover and M. Laguna. 1993. Tabu search. In *Modern Heuristic Techniques for Combinatorial Problems*.

L. A. Goodman. 1974. Exploratory latent structure analysis using both identifiable and unidentifiable models. *Biometrika*, 61:215–231.

D. Haughton. 1988. On the choice of a model to fit data from an exponential family. *Annals of Statistics*, 16:342–355.

D. Heckerman. 1995. A tutorial on learning with Bayesian networks. Technical Report MSR-TR-95-06, Microsoft Research.

M. Jamshidian and R. I. Jennrich. 1997. Acceleration of the EM algorithm by using Quasi-Newton methods. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 59(3):569–587.

F. V. Jensen, S. L. Lauritzen, and K. G. Olesen. 1990. Bayesian updating in causal probabilistic networks by local computations. *Computational Statistics Quaterly*, 4:269–282.

G. Karčiauskas, F. V. Jensen, and T. Kočka. 2004a. Parameter reusing in learning latent class models. In *Eighth International Symposium on Artificial Intelligence and Mathematics*.

G. Karčiauskas, T. Kočka, F. V. Jensen, P. Larrañaga, and J. A. Lozano. 2004b. Learning of latent class models by splitting and merging components. In *Second Workshop on Probabilistic Graphical Models*.

G. Karčiauskas. 2002. Text categorization using hierarchical Bayesian network classifiers. MSc Thesis. Aalborg University.

S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. 1983. Optimization by simulated annealing. *Science*, 220:671–680.

T. Kočka and N. Zhang. 2002. Dimension correction for hierarchical latent class models. In *Uncertainty in Artificial Intelligence: Proceedings of the Eighteenth Conference*, pages 267–274.

C.-K. Kwoh and D. F. Gillies. 1996. Using hidden nodes in Bayesian networks. *Artificial Intelligence*, 88(1-2):1–38.

S. L. Lauritzen. 1995. The EM algorithm for graphical association models with missing data. *Computational Statistics and Data Analysis*, 19:191–201.

P. F. Lazarsfeld and N. W. Henry. 1968. *Latent Structure Analysis*. Boston: Houghton Mifflin.

D. D. Lewis. 1997. Reuters-21578 text categorization test collection. http://www.daviddlewis.com/resources/testcollections/reuters21578.

L. Liu, D. C. Wilkins, X. Ying, and Z. Bain. 1990. Minimum error tree decomposition. In *Uncertainty in Artificial Intelligence: Proceedings of the Sixth Conference*, pages 180–185.

D. Madigan and J. York. 1995. Bayesian graphical models for discrete data. *International Statistical Review*, 63:215–232.

J. Martin and K. VanLehn. 1995. Discrete factor analysis: Learning hidden variables in Bayesian networks. Technical report, Department of Computer Science, University of Pittsburgh.

Geoffrey J. McLachlan and Thriyambakam Krishnan. 1997. *The EM Algorithm and Extensions*. John Wiley and Sons.

C. Meek, B. Thiesson, and D. Heckerman. 2002. Staged mixture modelling and boosting. In *Uncertainty in Artificial Intelligence: Proceedings of the Eighteenth Conference*, pages 335–343.

C. Meek. 1997. *Graphical models: selecting causal and statistical models*. Ph.D. thesis, Carnegie Mellon University.

M. Meila and D. Heckerman. 1998. An experimental comparison of several clustering and initialization methods. In *Uncertainty in Artificial Intelligence: Proceedings of the Fourteenth Conference*, pages 386–395.

X.-L. Meng and D. van Dyk. 1997. The EM algorithm – an old folk-song sung to a fast new tune. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 59(3):511–567.

C. Montacié, M.-J. Caraty, and C. Barras. 1996. Mixture splitting technic and temporal control in a HMM-based recognition system. In *Proceedings of the International Conference on Spoken Language Processing*, volume 2, pages 977–980.

A. Moore. 1999. Very fast EM-based mixture model clustering using multiresolution kd-trees. In *Advances in Neural Information Processing Systems 11*, pages 543–549.

R. Neal and G. Hinton. 1998. A view of the EM algorithm that justifies incremental, sparse, and other variants. In M. I. Jordan, editor, *Learning in Graphical Models*, pages 355–368.

R. E. Neapolitan. 2003. *Learning Bayesian Networks*. Prentice Hall.

NorsysSoftwareCorp. 2005. Tutorial on Bayesian Networks with Netica. http://www.norsys.com/tutorials/netica/secD/tut_D1.htm.

L. Ortiz and L. Kaelbling. 1999. Accelerating EM: An empirical study. In *Uncertainty in Artificial Intelligence: Proceedings of the Fifteenth Conference*, pages 512–521.

M. Ostendorf and H. Singer. 1997. HMM topology design using maximum likelihood successive state splitting. *Computer Speech and Language*, 11(1):17–41.

J. Pearl. 1986. Fusion, propagation, and structuring in belief networks. *Artificial Intelligence*, 29(3):241–288.

J. Pearl. 1988. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers Inc.

S. Ramachandran and R. J. Mooney. 1998. Theory refinement for Bayesian networks with hidden variables. In *Proceedings of the 15th International Conference on Machine Learning*, pages 454–462.

S. Richardson and P. J. Green. 1997. On Bayesian analysis of mixtures with an unknown number of components. *Journal of the Royal Statistical Society: Series B*, 59(4):731–758.

D. Rusakov and D. Geiger. 2003. Automated analytic asymptotic evaluation of the marginal likelihood for latent models. In *Uncertainty in Artificial Intelligence: Proceedings of the Nineteenth Conference*, pages 501–508.

R. Salakhutdinov and S. T. Roweis. 2003. Adaptive overrelaxed bound optimization methods. In *Machine Learning, Proceedings of the Twentieth International Conference (ICML 2003)*, pages 664–671.

G. Schwarz. 1978. Estimating the dimension of a model. *Annals of Statistics*, 6(2):461–464.

P. Spirtes, C. Glymour, and R. Scheines. 1993. *Causation, Prediction, and Search*. Springer-Verlag.

P. Spirtes, C. Glymour, and R. Scheines. 2000. *Causation, Prediction, and Search*. The MIT Press, 2 edition.

B. Stenger, V. Ramesh, N. Paragios, F. Coetzee, and J. M. Buhmann. 2001. Topology free hidden Markov models: Application to background modeling. In *Proceedings of the Eighth International Conference On Computer Vision*, pages 294–301.

A. Stolcke and S. M. Omohundro. 1994. Inducing probabilistic grammars by bayesian model merging. In *Grammatical Inference and Applications, Second International Colloquium*, pages 106–118.

J. Takami and S. Sagayama. 1992. A successive state splitting algorithm for efficient allophone modeling. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, volume 1, pages 573–576.

B. Thiesson, C. Meek, and D. Heckerman. 1999. Accelerating EM for large databases. Technical Report MSR-TR-99-31, Microsoft Research.

B. Thiesson. 1995. Accelerated quantification of Bayesian networks with incomplete data. In *Proceedings of the First International Conference on Knowledge Discovery and Data Mining*, pages 306–311.

J. Tian and J. Pearl. 2002. On the testable implications of causal models with hidden variables. In *Uncertainty in Artificial Intelligence: Proceedings of the Eighteenth Conference*, pages 519–527.

J. Uebersax. 2000. A brief study of local maxima solutions in latent class analysis. http://ourworld.compuserve.com/homepages/jsuebersax/local.htm.

J. Uebersax. 2001. LCA frequently asked questions. http://ourworld.compuserve.com/homepages/jsuebersax/faq.htm.

N. Ueda and Z. Ghahramani. 2002. Bayesian model search for mixture models based on optimizing variational bounds. *Neural Networks*, 15(10):1223–1241.

N. Ueda, R. Nakano, Z. Ghahramani, and G. E. Hinton. 2000. SMEM algorithm for mixture models. *Neural Computation*, 12(9):2109–2128.

P. van der Putten and M. van Someren (eds). 2000. CoIL Challenge 2000: The Insurance Company Case. Published by Sentient Machine Research, Amsterdam. Also a Leiden Institute of Advanced Computer Science Technical Report 2000-09.

J. J. Verbeek, N. Vlassis, and B. Krose. 2003. Efficient greedy learning of Gaussian mixture models. *Neural Computation*, 15(2):469–485.

N. Vlassis and A. Likas. 2002. A greedy EM algorithm for Gaussian mixture learning. *Neural Processing Letters*, 15(1):77–87.

Y. Yang and X. Liu. 1999. A re-examination of text categorization methods. In *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 42–49.

Y. Yang and J. O. Pedersen. 1997. A comparative study on feature selection in text categorization. In *Proceedings of the Fourteenth International Conference on Machine Learning*, pages 412–420.

N. L. Zhang and T. Kočka. 2004. Efficient learning of hierarchical latent class models. In *16th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2004)*, pages 585–593.

N. L. Zhang, T. D. Nielsen, and F. V. Jensen. 2004. Latent variable discovery in classification models. *Artificial Intelligence in Medicine*, 30(3):283–299.

N. L. Zhang. 1996. Irrelevance and parameter learning in Bayesian networks. *Artificial Intelligence*, 88(1-2):359–373.

N. L. Zhang. 2002. Hierarchical latent class models for cluster analysis. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence and Fourteenth Conference on Innovative Applications of Artificial Intelligence (AAAI/IAAI)*, pages 230–237.

N. L. Zhang. 2004. Hierarchical latent class models for cluster analysis. *Journal of Machine Learning Research*, 5:697–723.

S. Zilberstein. 1996. Using anytime algorithms in intelligent systems. *AI Magazine*, 17(3):73–83.

Index