Learning Hierarchical Latent Class Models

Technical Report HKUST-CS03-01

Nevin L. Zhang
Department of Computer Science
Hong Kong University of Science & Technology, China
lzhang@cs.ust.hk

Tomáš Kočka, Gytis Karciauskas, Finn V. Jensen
Department of Computer Science
Aalborg University, Denmark.
{kocka, gytis, fvj}@cs.auc.dk

Abstract

Hierarchical latent class (HLC) models generalize latent class models. As models for cluster analysis, they suit more applications than the latter because they relax the often untrue conditional independence assumption. They also facilitate the discovery of latent causal structures and the induction of probabilistic models that capture complex dependencies and yet have low inferential complexity. In this paper, we investigate the problem of inducing HLC models from data. Two fundamental issues of general latent structure discovery are identified and methods to address those issues for HLC models are proposed. Based on the proposals, we develop an algorithm for learning HLC models and demonstrate the feasibility of learning HLC models that are large enough to be of practical interest.

1 Introduction

Hierarchical latent class (HLC) models [Zhang 2002] are tree-structured Bayesian networks (BNs) where leaf nodes are observed while internal nodes are not. They generalize latent class (LC) models [Lazarsfeld and Henry 1968] and were first identified as a potentially useful class of Bayesian networks by Pearl [1988]. This paper is concerned with the problem of learning HLC models from data. The problem is interesting for three reasons.

First, the induction of HLC models from data can give rise to probabilistic models that represent complex dependencies among observed variables and yet are computationally simple to work with. The past decade has seen intensive research activities on learning BN models. The focus has mostly been on finding the model that maximizes a scoring function [e.g. Jordan 1998] or satisfies a set of constraints sanctioned by data [e.g. Spirtes *et al*. 1993]. Much progress has been made. However, relatively little consideration is given to the inferential complexity of the resulting models. There are two notable exceptions. One is the work on learning observed trees, i.e. trees over observed variables [Chow and Liu 1968]. The other is Elidan *et al.* [2001], who propose to reduce the model complexity of a BN by introducing latent variables. HLC models can represent more complex dependencies among observed variables than observed trees. Inference is easier in HLC models than in general BNs with latent variables.

Second, the endeavor of learning HLC models can reveal latent causal structures. Often, observed variables are correlated because they are influenced by some common hidden causes. HLC models can be interpreted as hypotheses about how latent causes influence observed variables and how they are correlated among themselves. Then finding an HLC model that fits data amounts to find a causal hypothesis that explains data. Researchers have already been inferring latent causal structures from observed data. One example is the reconstruction of phylogenetic trees [Durbin *et al*. 1998], which can be viewed as special HLC models.

Third, HLC models alleviate disadvantages of LC models as models for cluster analysis. An LC model consists of one latent variable, namely the class variable, and a number of observed feature variables. It assumes that the feature variables are mutually independent given the class variable. A serious problem with the use of LC models, known as *local dependence*, is that this assumption is often violated. If one does not deal with local dependence explicitly, one implicitly attributes it to the latent variable. In practice, this results in too many latent classes, many of them spurious, and degenerates the accuracy of classification [Vermunt and Magidson 2002]. HLC models alleviate this problem because it can model local dependence. As a matter of fact, the first systematic study on HLC models [Zhang 2002] was motivated by the need, in an application in traditional Chinese medicine, for cluster models that can deal with local dependence.

When learning BNs with latent variables, one needs to determine not only model structures, i.e. connections among variables, but also cardinalities of latent variables, i.e. the numbers of values they can take. Although not using the terminology of HLC models, Connolly [1993] proposed the first, somewhat *ad hoc*, algorithm for learning HLC models and tested it on one small toy example. A more principled algorithm was proposed by Zhang [2002]. This algorithm hill-climbs in the space HLC models guided by a scoring function. It starts with an LC model. At each step of search, it first generates a number of candidate structures by modifying the structure of the current model. It then optimizes cardinalities of latent variables in the candidate structures, resulting in candidate models. Finally, it evaluates the candidate models and pick the best one to seed the next step of search. Search terminates when the best candidate model is no better than the current model. To optimize the cardinalities of the latent variables in a model structure, the algorithm employees a lower level hill-climbing routine. Hence we call it the *double hill-climbing (DHC) algorithm*.

Empirical results reported in [Zhang 2002] show that the DHC algorithm performs well in terms of model quality when the BIC [Schwarz 1978] scoring function is used. However, it has a serious drawback, namely its high complexity. Let n be the number of observed variables. At each step of search, DHC generates $O(n^2)$ models structures. To optimize the cardinalities of the latent variables in a given model structure, the lower level search routine examines $O(n^2k)$ models, where k is the maximum cardinality for a latent variable. Hence there are totally $O(n^4k)$ models to evaluate. Before a model can be evaluated, its parameter must be optimized. Due to the presence of latent variables, parameter optimization requires the EM algorithm, which is known to be computationally expensive.

In this paper, we propose a new algorithm for learning HLC models. Unlike DHC, this algorithm does not obtain candidate models by first creating candidate structures and then optimizing cardinalities of latent variables using a lower level hill-climbing routine. Rather, it generates candidate models directly. We hence call it the *single hill-climbing (SHC) algorithm*. At each step of search, SHC generates and evaluates $O(n^2)$ candidate models and

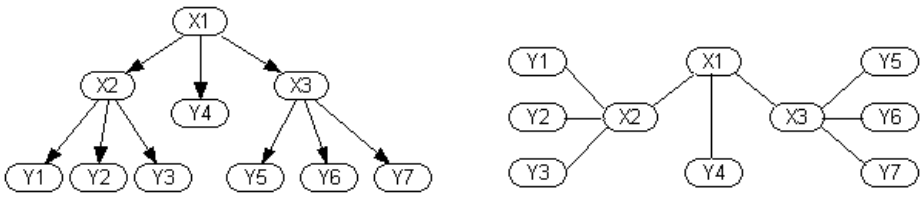

Figure 1: An example HLC model and the corresponding unrooted HLC model. The X_i's are latent variables and the Y_j's are manifest variables.

is hence much more efficient than DHC. In the meantime, empirical studies indicate that SHC works as well as DHC in terms of the quality of the models they produce. Moreover, SHC provides a framework where the idea of structural EM [Friedman 1997] can easily be applied to greatly reduce the number of calls to EM.

There are two fundamental issues in the context of general latent structure discovery. The first is the well-known problem of determining the cardinality of a new latent node. A straightforward solution would be to use a lower level hill-climbing routine. But this would lead to DHC. The second issue is how to make the tradeoff between variable complexity and structure complexity. Here *variable complexity* refers to the number of values latent variables can have, while *structure complexity* refers to the number of nodes and links among them. Those two aspects of model complexity are related because one can always merge two neighboring latent nodes. In other words, on can always increase variable complexity to compensate for the reduction in structure complexity. An issue of trading off naturally arises. In this paper, methods for addressing those two issues for HLC models are proposed.

The next section gives a brief review of some facts about HLC models. Section 3 develops the SHC algorithm and Section 4 reports empirical results. Concluding remarks are provided in Section 5.

2 Hiearchical latent class models

A *hierarchical latent class (HLC) model* is a Bayesian network where (1) the network structure is a rooted tree and (2) the variables at the leaf nodes are observed and all the other variables are not. An example HLC model is shown in Figure 1 (on the left). Following the latent class analysis literature, we refer to the observed variables as *manifest variables* and all the other variables as *latent variables*. A *latent class (LC) model* is an HLC model where there is only one latent node. We usually write an HLC model as a pair $M = (m, \theta)$, where θ is the collection of parameters. The first component m consists of the model structure and cardinalities for the latent variables. We will sometimes refer to m also as an HLC model. When it is necessary to distinguish between m and the pair (m, θ), we call m an *unparameterized HLC model* and the pair (m, θ) a *parameterized HLC model*.

Two parameterized HLC models $M{=}(m, \theta)$ and $M'{=}(m', \theta')$ are *marginally equivalent* if they share the same manifest variables $Y_1, Y_2, \ldots, Y_n$ and

$$P(Y_1, \ldots, Y_n|m, \theta) = P(Y_1, \ldots, Y_n|m', \theta'). \quad (1)$$

An unparameterized HLC models m *includes* another m' if for any parameterization θ' of m', there exists parameterization θ of m such that (m, θ) and (m', θ') are marginally equivalent, i.e. if m can represent any distributions over the manifest variables that m' can. If m includes m' and vice versa, we say that m and m' are *marginally equivalent*. Marginally equivalent (parameterized or unparamterized) models are *equivalent* if they have the same number of independent parameters. We cannot distinguish between equivalent models using penalized likelihood scores [Green 1998].

Let X_1 be the root of an HLC model m. Suppose X_2 is a child of X_1 and it is a latent node. Define another HLC model m' by reversing the arrow $X_1 \rightarrow X_2$. In m', X_2 is the root. The operation is hence called *root walking*; the root has walked from X_1 to X_2. Root walking leads to equivalent models [Zhang 2002]. This implies that it is impossible to determine edge orientation from data. We can learn only *unrooted HLC models*, which are HLC models with all directions on the edges dropped. Figure 1 also shows an example unrooted HLC model. An unrooted HLC model represents a class of HLC models. Semantically it is a Markov random field on an undirected

tree. The leaf nodes are observed while the interior nodes are latent. The concepts of marginal equivalence and equivalence can be defined for unrooted HLC models in the same way as for rooted models.

Let $|X|$ stand for the cardinality of a variable X. For a latent variable Z in an HLC model, enumerate its neighbors as $X_1, X_2, \ldots, X_k$. An HLC model is *regular* if for any latent variable Z,

$$|Z| \leq \frac{\prod_{i=1}^{k} |X_i|}{\max_{i=1}^{k} |X_i|}, \tag{2}$$

and when Z has only two neighbors and one of which is also a latent node,

$$|Z| \leq \frac{|X_1||X_2|}{|X_1| + |X_2| - 1}. \tag{3}$$

Note that this definition applies to parameterized as well as to unparameterized models.

Given an irregular parameterized model M, there exists, a regular model that is marginally equivalent to M and has fewer independent parameters [Zhang 2002]. Such a regular model can be obtained from M by deleting, one by one, nodes that violate Condition (3) and reducing the cardinality of each node that violates Condition (2) to the quantity on the right hand side. The second step needs to be repeated until cardinalities of latent variables can no longer to reduced. We will refer to the process as *regularization*. It is evident that if penalized likelihood is used for model selection, the regularized model is always preferred over M itself.

3 Learning HLC models

Assume that there is a collection of i.i.d samples on a number of manifest variables generated by an unknown regular HLC model. This section presents an algorithm, named SHC, for reconstructing the regular unrooted HLC models that corresponds to the generative model. In the rest of this section, we always mean unrooted models when speaking of HLC models.

3.1 General Issues

The search space that SHC works with consists of all unrooted regular HLC models for the given manifest variables. As for scoring functions, Zhang (2002) tested four with the DHC algorithm, namely AIC, hold-out LS, BIC, and CS. Empirical results indicate BIC and CS are more suitable for the task of reconstructing HLC models than the other two. In this paper, we use BIC. Given a data set $\mathbf{D}$, the BIC score of a model m is

$$BIC(m|\mathbf{D}) = logP(\mathbf{D}|m, \theta^*) - \frac{d}{2} logN$$

where θ^* is the ML estimate of model parameters, d is the dimension of m, i.e. the number of independent parameters, and N is the sample size. The first term is known as maximized loglikelihood. It has the following nice property: If model m includes m', then the maximized loglikelihood of m is larger than or equal to that of m'. The second term is a penalty term. It decreases with sample size.

The overall strategy of SHC is similar to that of greedy equivalence search (GES), an algorithm for learning Bayesian network structures in the case when all variables are observed (Meek 1997). It begins with the simplest HLC model and works in two phases. In Phase I, SHC *expands* models by introducing new latent nodes and additional states for existing nodes. The aim is to improve the likelihood term of the BIC score. In Phase II, SHC *retracts* models by deleting latent nodes or states of latent nodes. The aim is to reduce the penalty term of the BIC score, while keeping the likelihood term more less the same. If model quality is improved in Phase II, SHC goes back to Phase I and the process repeats itself.

3.2 Search operators

SHC hill-climbs in the space of regular HLC models using five search operators, namely State Introduction, Node Introduction, Node Relocation, State Deletion, and Node Deletion. The first three operators are used in Phase I and the rest are used in Phase II.

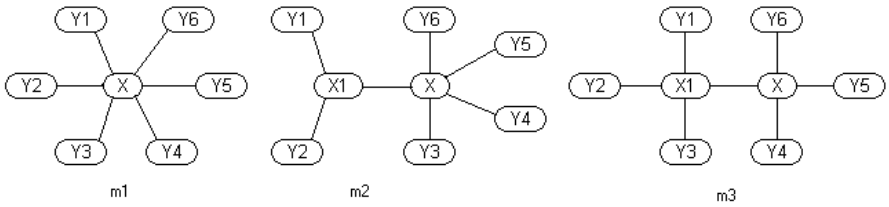

Figure 2: Illustration of Node Introduction and Node Relocation.

Given an HLC model and a latent variable in the model, *State Introduction* creates a new model by adding an additional state of the state space of the variable. Clearly, the new model includes the old model.

Node Introduction involves one latent node X in an HLC model and two of its neighbors. It creates a new model by introducing a new latent node X' to mediate X and the two neighbors. The new node has the same number of states as X. Consider the HLC model m_1 in Figure 2. Applying the Node Introduction operator to the latent Node X and its neighbors Y_1 and Y_2 results in the model m_2. The new node X_1 has the same state space as X. For the sake for computational efficiency, we do not consider introducing a new node to mediate X and more than two of its neighbors. Without this restriction, the number of candidate models the operator could produce is exponential in the number of neighbors. Also note that Node Introduction is disallowed when X has only two neighbors. In this case, it would create a latent node that is also a leaf node. In HLC models, only manifest nodes can be leaves.

Let m' be a model obtained from another model m via Node Introduction. Then m' includes m. Note that if we set the cardinality of the new node X' to $|X|$, then it might no longer be the case that m' includes m. On the other hand, if we set $|X'|$ to a number larger than $|X|$, then m' is more complex than necessary. The only alternative left is to set $|X'|$ to be the same as $|X|$. This addresses the first fundamental issue raised at the end of Section 1.

The two operators discussed so far introduce new ingredients to a model. Called *Node Relocation*, the next operator re-arranges connections among existing nodes. It involve two neighboring latent nodes X_1 and X_2 and a neighbor Z of X_1 that is different from X_2. It creates a new model by relocating Z to X_2, i.e. removing the link between Z and X_1 and adding a link between Z and X_2. Consider the HLC model m_2 in Figure 2. Relocating Y_3 from X to X_1 results in model m_3. Note that a node is allowed to be relocated only "one step away". This is for the sake of computational efficiency and, judging from our experience, more flexibility does not seem necessary. Also note that if the latent node X_1 has only two neighbors, relocating Z to X_2 would make the latent node X_1 a leaf node. In this case, we simply remove X_1.

There is a variant to Node Relocation that we call *Accommodating Node Relocation*. It is the same as Node Relocation except that, after relocating a node, it adds one state to the new neighbor of the node. To understand the need for this variant, consider again the models m_2 and m_3 in Figure 2. Model m_3 is obtained from model m_2 by relocating Y_3 from X to X_1. In m_2, X_1 mediates the interactions among three variables, while in m_3 it mediates the interactions among four variables. The more states a latent variable has, the more interactions among its neighbors it can capture. In other words, the cardinality of a latent variable limits its the ability in mediating interactions. For the sake of argument, assume the relocating Y_3 to X_1 is a "good" move. Sometimes the benefit of such a move cannot be realized unless the cardinality of X_1 is increased.

State Deletion is the opposite of State Introduction. Given an HLC model and a latent node in the model, it creates a new model by deleting a state of the latent node. It is not applicable if the latent node has only two states. *Node Deletion* is the opposite of Node Introduction. It involves two neighboring latent node X and X' in an HLC model. It creates a new model by deleting X and making all neighbors of X other than X' neighbors of X'. If model m' is obtained by applying State or Node Deletion to an model m, then m includes m'.

All of the five operators might lead to the violation of the regularity constraints. We therefore follow each operator immediately with a regularization step.

3.3 Model selection

Given a data set, our task to is find a model that fits the data well and has low complexity. It is possible to achieve perfect fit to data using an LC model where the latent node has a high cardinality. That is to use the model with the lowest structure complexity and high variable complexity. Clearly this model does not meet our objective. We need to find a balance between variable complexity and structure complexity so that the overall model complexity is low. This is the second fundamental issue mentioned at the end of Section 1.

State Introduction increases variable complexity, while Node Introduction increases structure complexity. To find an appropriate tradeoff between the two aspects of model complexity, SHC starts with the model that has the lowest variable complexity and the lowest structure complexity, i.e. the LC model where the latent node has only two states. At each step in Phase I, it decides whether and how to apply State Introduction, Node Introduction, or Node Relocation. The key question is how this decision should be made.

A naive answer to the question is to (1) generate candidate models by applying those three search operators on the current model, (2) evaluate them one by one, and (3) pick the one with the highest score. This strategy does not work. To understand why, consider the first step, where the current model is the LC model with a binary latent node. Suppose there are n manifest variables. State Introduction would increase the number of model parameters by $n+1$, while Node Introduction would increase the number of model parameters only by 2. It therefore comes with no surprise that the (unique) model generated by State Introduction is likely to have a much higher score than models produced by Node Introduction. Consequently, State Introduction is likely to be chosen, resulting in another LC model where the latent node has three states. Repeating the arguments, we see that State Introduction is likely to be applied again in the next step, and again in the step after, and so on. Our experiments have confirmed this behavior. We observed that SHC would never leave LC models when this naive model selection strategy was used.

Define the cost of (a particular application of) an operator to be the increase in model parameters it brings about. A natural way to overcome the above difficulty is to choose the operator that is the most cost-effective. To be more specific, let m be the current model and m' be a candidate model. Define the *unit improvement of m' over m* to be

$$U(m', m) = \frac{\texttt{score}(m') - \texttt{score}(m)}{\texttt{dimension}(m') - \texttt{dimension}(m)},$$

where $\texttt{dimension}(m)$ stands for the number of independent parameters in m. SHC adopts the following model selection strategy:

> Among all candidate models, choose the one that has the highest unit improvement over the current model.

We will refer to this strategy as *cost-effectiveness model selection*.

Node Relocation does not necessarily increase the number of model parameters. Care must be taken when comparing models it produces with models generated by other operators. At each step, SHC considers Node Relocation first. For each candidate model m' obtained from the current model m by relocating a node, SHC check whether the score of m' is greater than that of m. If this is not the case, m' is discarded and SHC creates another model m'' by adding an additional state to the new neighbor of the node that was relocated. In other words, Accommodating Node Relocation is applied. If the score of m'' is still not greater than that of m, it is also discarded. The remaining candidate models, if any, all have scores higher than that of m. If some of the remaining models have no more parameters than m, then SHC skips the rest of the current iteration and moves to the next step with the first such model it encounters. By doing so, it improves model score without increasing model complexity. If no such models exist, SHC compares the remaining models with models generated by State Introduction and Node Introduction in terms of cost-effectiveness and pick the one with the highest unit improvement over m to seed the next step.

Model selection in Phase II is straightforward and is based on model score.

3.4 The SHC algorithm

We now give the pseudo code of the SHC algorithm. The input to the algorithm is a data set $\mathbf{D}$ on a list of manifest variables. Records in $\mathbf{D}$ do not necessarily contain values for all the manifest variables. The output is an unrooted HLC model. Model parameters are optimized using the EM algorithm. Given a model M, the collections of candidate models the search operators produce will be denoted by $\texttt{NI}(M)$, $\texttt{SI}(M)$, $\texttt{NR}(M)$, $\texttt{ANR}(M)$, $\texttt{ND}(M)$, and

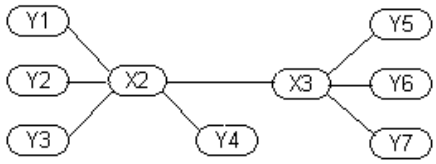

Figure 3: The unrooted HLC model SHC learned from data that were generated using the model in Figure 1.

$\texttt{SD}(M)$. Here NI, for instance, is a shorthand for Node Introduction.

$\texttt{SHC}(\mathbf{D})$
Let M be the LC model with a binary latent node.
Repeat until termination:
  M'=$\texttt{PhaseI}(M, \mathbf{D})$.
  If $\texttt{score}(M') \leq \texttt{score}(M)$, return M. Else $M = M'$.
  M'=$\texttt{PhaseII}(M, \mathbf{D})$.
  If $\texttt{score}(M') \leq \texttt{score}(M)$, return M. Else $M = M'$.

$\texttt{PhaseI}(M, \mathbf{D})$
Repeat until termination:
  $\mathcal{S} = \emptyset$.
  For $M' \in \texttt{NR}(M)$,
    If $\texttt{score}(M') > \texttt{score}(M)$, add M' to $\mathcal{S}$
    Else let M'' be the corresponding model in ANR(M).
      If $\texttt{score}(M'') > \texttt{score}(M)$, add M'' to $\mathcal{S}$
  If there is $M' \in S$ s.t. $\texttt{dimension}(M') \leq \texttt{dimension}(M)$
    $M = M'$ and continue.
  Let $M' \in \mathcal{S} \cup \texttt{NI}(M) \cup \texttt{SI}(M)$ s.t. $U(M', M)$ is maximum.
  If $\texttt{score}(M') \leq \texttt{score}(M)$, return M. Else $M = M'$.

$\texttt{PhaseII}(M, \mathbf{D})$
Repeat until termination:
  Let $M' \in \texttt{ND}(M) \cup \texttt{SD}(M)$ s.t. $\texttt{score}(M')$ is maximum.
  If $\texttt{score}(M') \leq \texttt{score}(M)$, return M. Else $M = M'$.

In the Appendix, we illustrate in detail how SHC works with an example.

4 Empirical results

This section reports experiments designed to determine whether SCH can learn models of good quality and how efficient it is. In all the experiments, EM was configured as follows. To estimate parameters for a given unparameterized model m, we first randomly generated 64 different parameterizations of m. This gave us 64 initial parameterized models. One EM iteration was run on all models and afterwards the worst 32 models were discarded. Then two EM iterations were run on the remaining 32 models and afterwards the worst 16 models were discarded. This process was continued until there was only one model. On this model, EM was terminated either if the increase in loglikelihood fell below 0.01 or the total number of iterations exceeded 500.

Our experiments were based on synthetic data. In the first experiment, data were generated using the model shown in Figure 1. The cardinalities of all variables were set at 3. Parameters for the model were randomly generated except that we ensured that each conditional distribution has a component with mass larger than 0.6. A data set of 10,000 records were sampled.

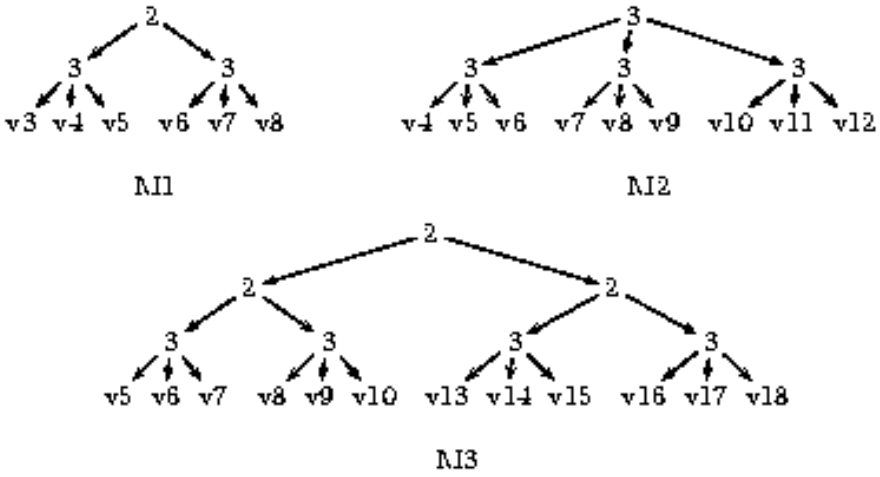

Figure 4: Test Models: Manifest nodes are labelled with their names. All manifest variables have 3 states. Latent nodes are labelled with their cardinalities.

We tested both DHC and SHC on the data set. The tests were carried on a PC with a 1 GHz Pentium III processor. DHC took 97 hours to terminate while SHC finished in 4.4 hours. SHC was about 22 times faster. In terms of model quality, DHC recovered the unrooted model in Figure 1 except that the cardinality of X_1 was underestimated by one. On the other hand, SHC yielded the model in Figure 3, where both X_2 and X_3 have cardinality 3. This model closely resembles the one found by DHC. Denote by P_1 and P_2 the joint distributions of the manifest variables in the models found by DHC and SHC respectively. The KL divergence of P_2 from P_1 is 0.0017. So the two distributions are very close.

It might appear that both DHC and SHC failed in this experiment because they did not exactly recover the generative model. In reality, the generative model is in the neighborhood of the learned model in both cases. This implies that the two models were compared and the learned model was chosen because it is better than the generative model according to data.

For the second experiment, we used the models M_1, $M2$, and M_3 shown in Figure 4. Parameters were randomly generated except that we ensured that each conditional distribution has a component with mass larger than 0.8. We also ensured that, in every conditional probability table, that the large components of different rows are not all at the same column. A data set of 10,000 records were sampled for each model. We will denote the three data sets by $\mathbf{D}_1$, $\mathbf{D}_2$, and $\mathbf{D}_3$ respectively. SHC was tested on the data sets, while DHC was not because of its high complexity.

The unrooted HLC model SHC produced based on $\mathbf{D}_2$ corresponds exactly to the rooted HLC model M_2. Models M_1 and M_3 are not regular. In both models, Condition (3) is violated by the root node. Let M_1' and M_3' be the models obtained respectively from M_1 and M_3 by regularization. They are the same as M_1 and M_3 except that the root node is deleted and an edge is added to connect the two children of the root. The unrooted HLC models SHC produced based on $\mathbf{D}_1$ and $\mathbf{D}_3$ correspond precisely to M_1' and M_3'. In other words, SHC performed well in terms of model quality. In all cases, SHC found the final model in Phase I.

Figure 5 shows the running times of SHC on a PC with a 2.26GHz Pentium 4 processor. We see that, although being much faster than DHC, SHC is still inefficient due to the large number of calls to EM. Fortunately, SHC provides a framework where the idea of structural EM [Friedman 1997] can easily be applied to drastically reduce the number of calls to EM. We have been investigating this opportunity in parallel with the work reported in the current paper. A heuristic version of SHC have been developed. It yielded the same models as SHC on the three data sets and, as can seen from Figure 5, it is much faster than SHC. (Temporary note: Even better results are expected in the near future.) Details of this work will be described in a separate paper.

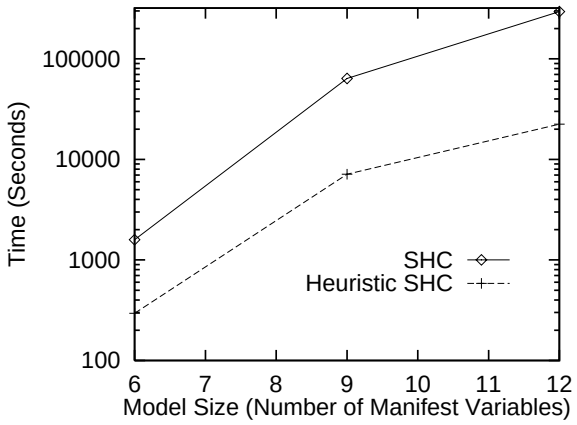

Figure 5: Running time of SHC and a heuristic verison of SHC on three synthetic data sets.

5 Conclusions

It is interesting to learn HLC models because, as models for cluster analysis, they relax the often untrue conditional independence assumption of LC models and hence suit more applications. They also facilitate the discovery of latent causal structures and the induction of probabilistic models that capture complex correlations and yet have low inferential complexity. In this paper, we have developed an algorithm for learning HLC models called SHC and demonstrated that SHC is able to learn HLC models that are large enough to be of practical interest.

Acknowledgement

Research was partially supported Hong Kong Research Grants Council under grant HKUST6088/01E.

References

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

Connolly, D. (1993). Constructing hidden variables in Bayesian networks via conceptual learning. *ICML-93*, 65-72.

Durbin, R., Eddy, S., Krogh, A., and Mitchison, G. (1998). *Biological sequence analysis: probabilistic models of proteins and nucleic acids*. Cambridge University Press.

Elidan, G., Lotner, N., Friedman, N. and Koller, D. (2001). Discovering hidden variables: A structure-based approach. *NIPS-01*.

Friedman, N. (1997). Learning belief networks in the presence of missing values and hidden variables. *ICML-97*, 125-133.

Green, P. (1998). Penalized likelihood. In *Encyclopedia of Statistical Sciences*, Update Volume 2. John Wiley & Sons.

Jordan, M. J. (ed.) (1998). *Learning in graphical models.* Kluwer Academic Publishers.

Lazarsfeld, P. F., and Henry, N.W. (1968). *Latent structure analysis*. Boston: Houghton Mifflin.

Meek, C. (1997). *Graphical models: Selection causal and statistical models.* Ph.D. Thesis, Carnegie Mellon University.

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference* Morgan Kaufmann Publishers, Palo Alto.

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

Spirtes, P., Glymour, C., and Scheines, R. (1993). *Causation, prediction, and search.* Springer-Verlag.

Vermunt, J.K. and Magidson, J. (2002). Latent class cluster analysis. In Hagenaars, J. A. and McCutcheon A. L. (eds.). *Advances in latent class analysis*. Cambridge University Press.

Zhang, N. L. (2002). Hierarchical latent class models for cluster analysis. *AAAI-02*, 230-237.

Appendix: Execution trace of SHC on data sampled from M3 of Figure 4.

Nodes affected by search operators are indicated with ovals.