Mach Learn (2015) 98:301–330
DOI 10.1007/s10994-013-5393-0

Greedy learning of latent tree models for multidimensional clustering

Teng-Fei Liu · Nevin L. Zhang · Peixian Chen · April Hua Liu · Leonard K.M. Poon · Yi Wang

Received: 15 May 2012 / Accepted: 8 June 2013 / Published online: 29 June 2013

Abstract Real-world data are often multifaceted and can be meaningfully clustered in more than one way. There is a growing interest in obtaining multiple partitions of data. In previous work we learnt from data a latent tree model (LTM) that contains multiple latent variables (Chen et al. 2012). Each latent variable represents a soft partition of data and hence multiple partitions result in. The LTM approach can, through model selection, automatically determine how many partitions there should be, what attributes define each partition, and how many clusters there should be for each partition. It has been shown to yield rich and meaningful clustering results.

Our previous algorithm EAST for learning LTMs is only efficient enough to handle data sets with dozens of attributes. This paper proposes an algorithm called BI that can deal with data sets with hundreds of attributes. We empirically compare BI with EAST and other more efficient LTM learning algorithms, and show that BI outperforms its competitors on

Editors: Emmanuel Müller, Ira Assent, Stephan Günnemann, Thomas Seidl, and Jennifer Dy.

T.-F. Liu · N.L. Zhang (✉) · P. Chen · A.H. Liu
Department of Computer Science and Engineering, The Hong Kong University of Science and Technology, Hong Kong, Hong Kong
e-mail: lzhang@cse.ust.hk

T.-F. Liu
e-mail: liutf@cse.ust.hk

P. Chen
e-mail: pchenac@cse.ust.hk

A.H. Liu
e-mail: aprillh@cse.ust.hk

L.K.M. Poon
Department of Mathematics and Information Technology, The Hong Kong Institute of Education, Hong Kong, Hong Kong
e-mail: kmpoon@ied.edu.hk

Y. Wang
Institute of High Performance Computing, A*STAR, 1 Fusionopolis Way, Singapore 138632, Singapore
e-mail: wangyi@ihpc.a-star.edu.sg

data sets with hundreds of attributes. In terms of clustering results, BI compares favorably with alternative methods that are not based on LTMs.

Keywords Model-based clustering · Multiple partitions · Latent tree models

1 Introduction

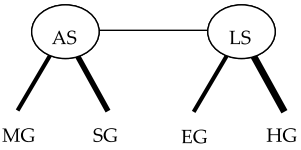
Latent tree models (LTMs) are tree-structured probabilistic graphical models where the leaf nodes represent observed variables, while the internal nodes represent latent variables. Special LTMs such as phylogenetic trees (Durbin et al. 1998) and latent class models (Bartholomew and Knott 1999) have been studied for decades. General LTMs were first investigated by Zhang (2004), where they are called hierarchical latent class models. LTMs can be used for latent structure discovery (Zhang et al. 2008a, 2008b; Chen et al. 2008), density estimation (Wang et al. 2008), and clustering (Chen et al. 2012; Poon et al. 2010). This paper is concerned with the use of LTMs for clustering. We consider only the case where all variables (observed and latent) are discrete.

Previous algorithms for learning LTMs can be divided into three groups. Algorithms in the first group aim at finding models that are optimal according to a scoring metric. They conduct search in the space of LTMs and are hence called *search algorithms* (Zhang 2004; Zhang and Kocka 2004; Chen et al. 2012). Algorithms in the second group introduce latent variables based on results of attribute clustering (Harmeling and Williams 2011; Mourad et al. 2011). We call them *AC-based algorithms*, where AC stands for attribute clustering. Algorithms in the third group are inspired by work on phylogenetic tree reconstruction (PTR) (Choi et al. 2011; Anandkumar et al. 2011). We will refer to them as *PTR-motivated algorithms*.

Empirical results reported by Mourad et al. (2013) indicate that search algorithms usually find the best models on small data sets with dozens of attributes, while AC-based algorithms can handle data sets with as many as 10,000 attributes. PTR-motivated algorithms have theoretical guarantees, but they require all latent variables have the same number of states and the number be known beforehand.

Liu et al. (2012) propose an algorithm, called the bridged-islands (BI) algorithm, for learning LTMs. The algorithm aims at finding *flat LTMs* where each latent variable is directly connected to at least one observed variable. All the observed variables that are directly connected to a given latent variable are said to be *siblings* and form a *sibling cluster*. Similar to AC-based algorithms, BI determines potential siblings by considering how closely correlated each pair of observed variables are. Unlike AC-based algorithms, BI has a novel procedure called *uni-dimensionality (UD) test* to detect highly correlated observed variables that should be siblings. Initial sibling clusters are thereby created. One latent variable is then introduced for each sibling cluster and the latent variables are connected to form a tree structure using Chow-Liu's algorithm (Chow and Liu 1968). There is also a final global adjustment step that refines the model.

This paper builds upon and extends Liu et al. (2012). We compare BI with previous algorithms for learning LTMs in terms of computational complexity and model quality. We also study the impact of key parameters and sample size to the performance of BI. A variety of experiments on both synthetic and real-world data sets are presented. As it turns out, BI is significantly more efficient than search algorithms. It can handle data sets with hundreds of attributes. Moreover, it produces better models on such data sets than AC-based and PTR-motivated algorithms. As in Liu et al. (2012), we also use BI as a clustering method that



(a) Structure of an LTM

$P(AS)$	AS=low	AS=high
	0.7	0.3

(b) Distribution $P(AS)$

$P(MG \mid AS)$	MG=low	MG=high
AS=low	0.8	0.2
AS=high	0.2	0.8

(c) Conditional distribution $P(MG \mid AS)$

$P(LS \mid AS)$	LS=low	LS=high
AS=low	0.4	0.6
AS=high	0.6	0.4

(d) Conditional distribution $P(LS \mid AS)$

Fig. 1 An example LTM for high school students. Diagram (**a**) shows the structure of the model. It consists two discrete latent variables *AS* (*Analytical Skill*) and *LS* (*Literacy Skill*), and four discrete observed variables *MG* (*Math Grade*), *SG* (*Science Grade*), *EG* (*English Grade*) and *HG* (*History Grade*). We simply assume that all the variables have two possible values 'low' and 'high'. *Edges between variables* indicate probabilistic dependence. The *tables* show some probability distributions for the model. Other distributions are not shown to save space. The conditional distributions characterize dependence between variables. The *edge widths* visually show the strength of correlation between variables. They are not part of the model definition and are computed from the probability distributions of the model

produces multiple partitions of data, and compare BI with previous methods for the task that are not based on LTMs.

This paper is divided into two parts. In the first part, we briefly review LTMs (Sect. 2) and previous algorithms for learning LTMs (Sect. 3). Then we describe BI algorithm (Sect. 4) and compare it with previous algorithms (Sect. 5). In the second part, we survey previous methods for multi-partition clustering (Sect. 6) and investigate the use of BI for the task (Sects. 7–9). The paper concludes in Sect. 10.

2 Latent tree models

A *latent tree model (LTM)* is a Markov random field over an undirected tree, where variables at leaf nodes are observed and variables at internal nodes are hidden. An example LTM is shown in Fig. 1. For technical convenience, we often root an LTM at one of its latent nodes and regard it as a directed graphical model, i.e., a *Bayesian network* (Pearl 1988). In the example, suppose that we root the model at the node *AS*. Then numerical information of the model includes a marginal distribution $P(AS)$ for the root and one conditional distribution for each edge. For the edge $AS \rightarrow MG$, for instance, we have distribution $P(MG \mid AS)$, which characterizes the dependence of *MG* on *AS*. The product of all these distributions defines a joint distribution over all the latent and observed variables. Note that we can choose to root the model at any of the latent nodes because the selection of root node would not change the joint distribution (Zhang 2004). In other words, different choices of root nodes lead to an equivalent class of directed tree models.

In general, suppose there are n observed variables $X_1, \ldots, X_n$ and m latent variables $Y_1, \ldots, Y_m$ in an LTM. Assume the model be rooted at one of the latent variables. Denote the parent of a variable Z as $parent(Z)$ and let $parent(Z)$ be the empty set when Z is the root. The LTM defines a joint distribution over $X_1, \ldots, X_n, Y_1, \ldots, Y_m$ as follows:

$$P(X_1, \ldots, X_n, Y_1, \ldots, Y_m) = \prod_{Z \in \{X_1, \ldots, X_n, Y_1, \ldots, Y_m\}} P\big(Z \mid parent(Z)\big).$$

Throughout the paper, we use the terms 'node' interchangeably with 'variable', and term 'leaf node' interchangeably with 'attribute' and 'observed variable'. The set of attributes that are connected to a given latent variable is called a *sibling cluster*. Attributes in the cluster are said to be *siblings*. In Fig. 1, *MG* and *SG* form one sibling cluster because they all connected to latent node *AS*. Attributes *EG* and *HG* form another sibling cluster.

To learn an LTM from a data set D, one needs to determine: (1) the number of latent variables, (2) the number of states of each latent variable, which is sometimes called the *cardinality* of the variable, (3) the connections among the latent variables and observed variables, and (4) the probability parameters. In the following, we will use m to denote the information for the first three items and θ to denote the collection of parameter values.

To see how LTMs can be used for clustering, imagine that the model in Fig. 1 is learned from student transcript data. Then two partitions of the data are obtained, each being represented by a latent variable. In this scenario the latent variables were introduced during data analysis and are not properly named. We refer to them using their relative positions. The latent variable on the left is mainly related to Math grade and Science grade. Hence, it represents a soft partition of the students based primarily on their analytical skill. The latent variable on the right is mainly related to English grade and History grade. Hence, it represents a soft partition of the students based primarily on their literacy skill. We refer the reader to Sect. 7 for a much more detailed discussion of the use of LTMs for multi-partition clustering.

3 Previous algorithms for learning LTMs

A variety of algorithms for learning LTMs have been proposed previously. In this section, we give a brief overview of those algorithms. The reader is referred to Mourad et al. (2013) for a detailed survey.

We start with search algorithms which aim at finding the model m that maximizes the BIC score (Schwarz 1978):

$$BIC(m \mid D) = \log P\big(D \mid m, \theta^*\big) - \frac{d(m)}{2} \log N,$$

where θ^* is the maximum likelihood estimate of the parameters, $d(m)$ is the number of free probability parameters in m, and N is the sample size. The first search algorithm for learning LTMs was proposed by Zhang (2004). It is capable of handling only small toy data sets. Later two other search-based algorithms HSHC (Zhang and Kocka 2004) and EAST (Chen et al. 2012) were developed. These two algorithms gain efficiency by reducing the number of candidate models examined during search and by reducing the time spent on evaluating the candidate models. Between the two, EAST has a more principled method for evaluating candidate models and adopts a more intelligent search strategy. It is capable of handling data sets with dozens of attributes. However, it is unable to deal with data sets with hundreds of or more attributes because it still needs to evaluate a quadratic number of candidate models at each step of search.

Although search algorithms usually find high quality models, they are relatively slow. Greedy algorithms were consequently developed (Harmeling and Williams 2011; Mourad et al. 2011). These algorithms determine the structure of an LTM by performing agglomerative hierarchical clustering of observed variables. A latent variable is introduced for each node in the resulting dendrogram. The number of states for the latent variable is determined by considering its neighbors and using model selection criteria or some heuristics. The final

model can be a binary tree, a non-binary tree, or a forest. These AC-based algorithms are efficient, and can handle data sets with as many as 10,000 observed variables (Mourad et al. 2013). As will be demonstrated in Sect. 4, however, the efficiency comes with a compromise of model quality.

Phylogenetic trees (PTs) (Durbin et al. 1998) are diagrams depicting the evolutionary relationships among organisms. They can be viewed as special LTMs where all the (latent and observed) variables take the same possible values (i.e., A, C, G, T) and the conditional probability distributions are parameterized by edge lengths, which represent evolution time. *Phylogenetic tree reconstruction (PTR)* refers to the task of inferring the evolution tree for a collection of current species. A large number of algorithms have been developed for the task, including neighbor-joining (Saitou and Nei 1987) and quartet-based methods (Ranwez and Gascuel 2001). A key property that those algorithm exploits is that, in PTs, a concept of distance is defined for any two nodes.

Recently, new algorithms for learning LTMs have been proposed by drawing ideas from research on PTR. Choi et al. (2011) and Song et al. (2011) identify classes of LTMs where a concept of distance between nodes can be defined, while Mossel et al. (2011) and Anandkumar et al. (2011) define classes of LTMs where quartet tests can be performed. Unlike search algorithms and AC-based algorithms introduced above which are designed for discrete data, the PTR-motivated methods can deal with both discrete data and continuous data. For continuous data, they usually assume data are normally distributed. In the discrete case, they require all the observed variables have the same number of states. Theoretical results on consistence and sample complexity for PTR-motivated methods have been proved.

4 The bridged-islands algorithm

We now set out to present the greedy algorithm for learning LTMs. The algorithm aims at obtaining *flat LTMs* where each latent variable is connected to at least one observed variable. This restriction is motivated by the observation that search algorithms almost always yield flat LTMs. The algorithm proceeds in four steps:

1. Partition the set of attributes into sibling clusters;
2. Introduce a latent variable for each sibling cluster;
3. Connect the latent variables to form a tree;
4. Refine the model based on global considerations.

If we imagine the sibling clusters formed in Step 1, together with the latent variables added in Step 2, as islands in an ocean, then the islands are connected in Step 3. So we call the algorithm the *bridged-islands (BI)* algorithm.

The pseudo code for BI is given in Algorithm 1. In the following four subsections, we describe the steps of BI in details.

4.1 Sibling cluster determination

The first step of BI consists of lines 1–18 of the pseudo code. The objective is to determine sibling clusters. To identify potential siblings, BI considers how closely correlated each pairs of attributes are in terms of mutual information. The *mutual information (MI)* $I(X;Y)$ (Cover and Thomas 2006) between the two variables X and Y is defined as follows:

$$I(X;Y) = \sum_{X,Y} P(X,Y) \log \frac{P(X,Y)}{P(X)P(Y)}, \tag{1}$$

Algorithm 1 BI(D, δ)

Inputs: D—A data set with attributes V, δ—A real number.
Output: An LTM over V.

1: Calculate the empirical MI between each pair of variables in V.
2: $S^* \leftarrow \emptyset$
3: **while** $V \neq \emptyset$ **do**
4: $S \leftarrow$ the pair of variables with the highest MI.
5: **loop**
6: $X \leftarrow$ the variable in $V \setminus S$ that has the highest MI with S.
7: $S \leftarrow S \cup \{X\}$, $V \leftarrow V \setminus \{X\}$.
8: $D' \leftarrow$ projection of D onto S.
9: $m_1 \leftarrow$ `learnLCM`(D'), $m_2 \leftarrow$ `learnLTM-2L`(D').
10: **if** (m_2 contains 2 latent variables and $BIC(m_2 \mid D') - BIC(m_1 \mid D') > \delta$) **then**
11: Obtain a sibling cluster C from m_2 as described in Sect. 4.1. Add C to S^* and put the other observed variables in m_2 back to V. **Break**.
12: **else**
13: **if** ($V = \emptyset$) **then**
14: Make S a sibling cluster and add it to S^*.
15: **end if**
16: **end if**
17: **end loop**
18: **end while**
19: **for** each sibling cluster $C \in S^*$ **do**
20: $D_C \leftarrow$ projection of D onto C.
21: Learn an LCM for C by running `learnLCM`(D_C).
22: **end for**
23: $L^* \leftarrow$ the set of the latent variables of all LCMs.
24: $m \leftarrow$ Link up the latent variables of the LCMs by running `LearnCL`(L^*).
25: $m^* \leftarrow$ Optimize the parameters of m by running EM.
26: $m^* \leftarrow$ `ModelRefinement`(m^*).
27: Optimize the parameters of m^* by running EM, and return m^*.

where the summation is taken over all possible states of X and Y. In BI, the joint distribution $P(X, Y)$ is estimated from data. It is the joint empirical distribution of the two variables. The MI value calculated from the empirical distribution is called the *empirical MI*.

To determine the first sibling cluster, BI maintains a working set S of attributes that initially consists of the pair of attributes with the highest MI (line 4). Other attributes are added to the set one by one. At each step, BI chooses to add the attribute that has the highest MI with the current set (lines 6 and 7). The MI between a variable X and a set S of variables is estimated as follows:

$$I(X; S) = \max_{Z \in S} \sum_{X,Z} P(X, Z) \log \frac{P(X, Z)}{P(X)P(Z)}. \tag{2}$$

A key question is when to stop expanding the working set S. BI answers the question by performing a Bayesian statistical test to determine whether correlations among the variables in S can be properly modeled using one single latent variable (lines 8–11). The test is hence called the *uni-dimensionality test* or simply the *UD-test*. The expansion stops when the UD-test fails.

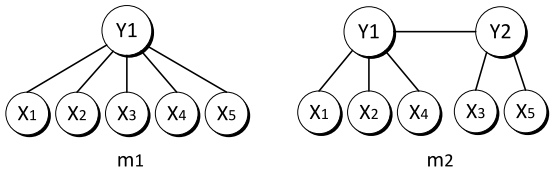


Fig. 2 The two models m_1 and m_2 considered in the UD-test

An LTM that contains only one latent variable is called a *latent class model* (LCM). To perform the UD-test, BI first projects the original data set D onto the working set S to get a smaller data set D' (line 8). Then it obtains from D' the best LCM m_1 and the best mode m_2 that contains 1 or 2 latent variables (line 9). The subroutines for these two tasks are given in Sect. 4.4. BI concludes that the *UD-test passes* if and only if one of these two conditions is satisfied: (1) m_2 contains only one latent variable, or (2) m_2 contains two latent variables and

$$BIC(m_2 \mid D') - BIC(m_1 \mid D') \leq \delta, \tag{3}$$

where δ is a threshold parameter.

The left hand side of inequality (3) is an approximation to the natural logarithm of the Bayes factor (Kass and Raftery 1995) for comparing m_2 with m_1. In Kass and Raftery (1995), guidelines are given for the use of Bayes factor (page 777). One set of guidelines are given in terms of twice the natural logarithm of Bayes factor. In terms of simply the natural logarithm of Bayes factor, the guidelines are as follows: a value between 1 and 3 is positive evidence favoring m_2, a value between 3 and 5 is strong evidence favoring m_2, and a value larger than 5 is very strong evidence favoring m_2. In our empirical evaluation, we usually set $\delta = 3$. Sometimes, we also consider $\delta = 1$, 5 or 10.

To illustrate the process, suppose that the working set $S = \{X_1, X_2\}$ initially. Two other attributes X_3 and X_4 are added to the set one by one and the UD-test passes in both cases. Then X_5 is added. Suppose the models m_1 and m_2 obtained at line 9 for $S = \{X_1, X_2, X_3, X_4, X_5\}$ are as shown in Fig. 2. Further suppose the BIC score of m_2 exceeds that of m_1 by threshold δ. Then UD-test fails and BI stops growing the set S.

When the UD-test fails, the model m_2 contains two latent variables. Each gives us a potential sibling cluster. Hence there are two potential sibling clusters. BI chooses one of them at line 11. If one of the two potential sibling clusters contains both the two initial attributes, it is picked. Otherwise, BI picks the one with more attributes and breaks ties arbitrarily.

In the aforementioned example, the two potential sibling clusters are $\{X_1, X_2, X_4\}$ and $\{X_3, X_5\}$. BI picks $\{X_1, X_2, X_4\}$ because it contains both the two initial attributes X_1 and X_2.

After the first sibling cluster is determined, BI removes the attributes in the cluster from the data set (line 11), and repeats the process to find other sibling clusters (line 3). This continues until all attributes grouped into sibling clusters.

4.2 Tree formation

The second step of BI is to learn an LCM for each sibling cluster (lines 19–22). One latent variable is introduced for each sibling cluster and the cardinality of the latent variable is determined. This is done using a subroutine that will be described in Sect. 4.4.

After the second step, we have a collection of LCMs. In Step 3, BI links up the latent variables of the LCMs to form a tree (lines 23–24). Chow and Liu (1968) give a well-known algorithm for learning tree-structured models among observed variables. It first estimates the

MI between each pair of variables from data, then constructs a complete undirected graph with the MI values as edge weights, and finally finds the maximum spanning tree of the graph. The resulting tree model has the maximum likelihood among all tree models.

Chow-Liu's algorithm can be adapted to link up the latent variables of the LCMs. We call the algorithm (i.e., subroutine *LearnCL* in line 24) to learn a Chow-Liu tree among latent variables. Then all LCMs are connected and form an LTM (line 24). We only need to specify how the MI between two latent variables is to be estimated. Let m and m' be two LCMs with latent variables Y and Y' respectively. We calculate the MI $I(Y; Y')$ between Y and Y' using (1) from the following joint distribution:

$$P\big(Y, Y' \mid D, m, m'\big) = C \sum_{d \in D} P(Y \mid m, d) P\big(Y' \mid m', d\big) \tag{4}$$

where $P(Y \mid m, d)$ is the posterior distribution of Y in m given data case d, $P(Y' \mid m', d)$ is that of Y' in m', and C is the normalization constant.

4.3 Model refinement

The sibling clusters and the cardinalities of the latent variables were determined in Step 1 and Step 2. Each of those decisions was made in the context of a small number of attributes. In Step 4 (lines 25–27), BI tries to detect the possible mistakes made in those steps based on global considerations and adjust the model accordingly. Specifically BI checks each attribute to see whether it should be relocated and each latent variable to see if its cardinality should be changed (i.e., subroutine *ModelRefinement*). To facilitate global considerations, BI first optimizes the probability parameters of the model resulted from Step 3 using EM algorithm (Dempster et al. 1977). The optimized model is denoted by m^* (line 25).

To detect beneficial node relocations, BI completes the data using the model m^*, re-estimates the MI between each pair of variables using the completed data, and considers adjusting connections among variables accordingly. To be specific, let X be an observed variable and Y be a latent variable. BI calculates the mutual information $I(X; Y)$ using (1) from the following distribution:

$$P\big(X, Y \mid D, m^*\big) = \frac{1}{N} \sum_{d \in D} P\big(X, Y \mid m^*, d\big), \tag{5}$$

where $P(X, Y \mid m^*, d)$ is the joint posterior distribution of X and Y in m^* given data case. Note that X is an observed variable. When the data set D contains no missing values, the equation can be rewritten as

$$P\big(X, Y \mid D, m^*\big) = \frac{1}{N} \sum_{d \in D} P\big(X \mid m^*, d\big) P\big(Y \mid m^*, d\big). \tag{6}$$

With this new formula, we need to compute only the posterior distribution for each latent variable, rather than for each latent variable—observed variable combination. It is computationally more efficient. In implementation, we use (6) even when there are missing values.

Let Y be the latent variable that is currently directly connected to X and Y^* be the latent variable that has the highest MI with X. If $Y^* \neq Y$, then BI deems beneficial to relocate X from Y to Y^*. This means to remove the edge between X and Y, and add an edge between X and Y^*.

To determine whether a change in the cardinality of a latent variable is beneficial, BI freezes all the parameters that are not affected by the change, runs EM locally (Chen et al. 2012) to optimize the parameters affected by the change, and recalculates the BIC score. The

change is deemed beneficial if the BIC is increased. BI starts from the current cardinality of each latent variable and considers increasing it by one. If it is beneficial to do so, further increases are considered.

All the potential adjustments (node relocation and cardinality change) are evaluated with respect to the model m^*. The beneficial adjustments are executed in one batch after all the evaluations. Adjustment evaluations and adjustment executions are not interleaved because that would require parameter optimization after each adjustment and hence be computationally expensive.

After model refinement, BI runs the EM algorithm on the whole model one more time to optimize the parameters (line 27).

It is well known that the EM algorithm generally converges to a local maxima likelihood estimate. To avoid the local maxima, we adopt the scheme proposed by Chickering and Heckerman (1997). The scheme first randomly generates a number α of initial values for the new parameters, resulting in α initial models. One EM iteration is run on all the models and afterwards the bottom $\alpha/2$ models are discarded. Then two EM iterations are run on the remaining models and afterwards the bottom $\alpha/4$ models are discarded. Then four EM iterations are run on the remaining models, and so on. The process continues until there is only one model. After that, more EM iterations are run on the remaining model, until the total number of iterations reaches a predetermined number.

4.4 Two subroutines

The BI algorithm needs two subroutines `learnLCM` and `learnLTM-2L`. The input to `learnLCM` is a data set D' with attributes S. This subroutine aims at finding the LCM for S that has the highest BIC score. To do so, it first creates an initial LCM for S and sets the cardinality of the only latent variable to 2. The parameters of the initial model are optimized by running EM and its BIC score is calculated. The subroutine considers repeatedly increasing the cardinality of the latent variable. After each increase, model parameters are re-optimized. The process stops when the BIC score ceases to increase.

The subroutine `learnLTM-2L` is more complex than `learnLCM`. Its pseudo code is given in Algorithm 2 to aid understanding. The objective is to find, among LTMs for S that contain 1 or 2 latent variables, the model that has the highest BIC score. The subroutine achieves the goal by searching the restricted model space. It starts with the LCM where the latent variable has only two states (line 1). At each step of search, it generates a collection of candidate models by modifying the current model m (lines 4 and 6). Each candidate model is evaluated. The one with the highest BIC score is picked as the next model. This last step is achieved using another subroutine `pickBestModel`. The search continues until model score ceases to increase (line 15).

When the current m contains two latent variables, we consider only candidate models produced by the *state introduction (SI)* operator (line 6). The operator creates a new model by increasing the cardinality of each latent variable by one. Hence two candidate models are produced.

When the current model m contains only one latent variable Y, the *node introduction (NI)* operator is also considered in addition to SI (line 4). This operator considers each pair of neighbors of Y. It creates a new model by introducing a new latent node Y' to mediate between Y and the two neighbors. The cardinality of Y' is set to be the same as that of Y. In the model m_1 shown in Fig. 2, if we introduce a new latent variable Y_2 to mediate of Y_1 and its neighbors X_3 and X_5, we get the model m_2. In this example, there are totally $\binom{5}{2}$ many possible ways to apply the NI operator. Hence NI produces $\binom{5}{2}$ candidate models. Because there is only one latent variable, SI produces only one candidate model.

Algorithm 2 `learnLTM-2L`(D')

Inputs: D'—a data set with attributes S.
Output: An LTM over S that contains 1 or 2 latent variables.

```
1:  m ← LCM with observed variables S and one latent variable Y with two values.
2:  loop
3:    if (m has only 1 latent variable) then
4:      m' ← pickBestModel (NI(m) ∪ SI(m)).
5:    else
6:      m' ← pickBestModel(SI(m)).
7:    end if
8:    if (m' was obtained from m by introducing a new latent variable Y') then
9:      loop
10:       m'' ← pickBestModel (NR(m', Y, Y')).
11:       if (BIC(m'' | D') ≤ BIC(m' | D')), break.
12:       m' ← m''.
13:     end loop
14:   end if
15:   if (BIC(m' | D') ≤ BIC(m | D')), return m.
16:   m ← m'.
17: end loop
```

Line 8 tests whether the best candidate model m' is produced at line 4 by the NI operator. The condition can be true for at most once. When it is true, suppose m' was obtained by introducing a new latent variable Y' mediates the existing latent variable Y and two of its neighbors. Then `learnLTM-2L` repeatedly tries to relocate other neighbors of Y to Y' until it is no longer beneficial to do so (lines 9–13). In the pseudo code, $NR(m', Y, Y')$ stands for the collection of models that can be obtained from m' by relocating one neighbor of Y to Y'.

4.5 Complexity analysis

The running time of BI is dominated by calls to the subroutine `learnLTM-2L` at line 9 and the two calls to EM on the whole model at lines 25 and 27. We analyze the complexity in terms of the following quantities: N—the sample size; n—the number of observed variables; l—the number of latent variables in the final model; c—the maximum cardinality of a latent variable; k—the maximum number of observed variables in the working set S; e—the maximum number of iterations of EM.

Let us first consider the complexity of one call to `learnLTM-2L`. Lines 4 and 6 in Algorithm 2 are executed no more than $2(c-2)$ times in total. In each of those calls, there are no more than $\binom{k}{2} < k^2$ candidate models. Line 11 is executed no more than $k-2$ times. In each of those calls, there are no more than k candidate models. Therefore, one call to `learnLTM-2L` involves no more than $2(c-2)k^2 + (k-2)k < 2ck^2$ candidate models.

To evaluate each candidate model, we need to run EM once to optimize its parameters. A candidate model contains no more than $k+2$ variables. There are N samples. Since inference in trees takes linear time, each EM iteration takes $O((k+2)N)$ time. Consequently, the time it takes to evaluate one candidate model is $O(ekN)$.

The while-loop of BI is executed l times. In each pass through the while-loop, `learnLTM-2L` is called no more then $k-2$ times. Putting everything together, we see that the total time that all calls to `learnLTM-2L` is $O(l \cdot (k-2) \cdot 2ck^2 \cdot ekN) = O(Nlec k^4)$. It

is linear in the sample size N and the number of latent variables l. In a more careful analysis, N can be replaced by the number of distinct data cases one gets by projecting N samples onto the working set S. It can be much smaller than N. The maximum cardinality c is usually very small relative to N and l and can be regarded as a constant. The maximum number of EM iterations e can be controlled. The term k^4 looks bad. Fortunately, k is usually much smaller than total number of observed variables n.

Now consider the two calls to EM at lines 25 and 27. They are run on the global model, which consists of n observed variables and l latent variables. Each iteration takes $O((l+n)N)$ times. Hence, the total time is $O(2N(l+n)e)$. It is linear in the sample size and the number of observed variables n.

In implementation, we let EM run from multiple random starting points to avoid local maxima. It is allowed to run a large number of random starts and a large number of iterations at line 27 since the parameters of the final model are optimized here. The number of random starts and the number of iterations take smaller values in other places, e.g., in `learnLTM-2L`. The reason is that the models encountered in `learnLTM-2L` contain no more than $k+2$ variables, which is much fewer than in the global model. Hence convergence can be reached in much fewer iterations.

5 Empirical comparison with previous LTM learning algorithms

In this section we empirically compare BI with previous algorithms for learning LTMs.[1] As discussed in Sect. 3, previous algorithms can be divided into three groups. Representative algorithms from each of the three groups are included in the comparisons, namely the search algorithm EAST (Chen et al. 2012), the AC-based algorithm BIN (Harmeling and Williams 2011), and the PTR-motivated algorithms CLNJ and CLRG (Choi et al. 2011).

5.1 Comparisons on synthetic data

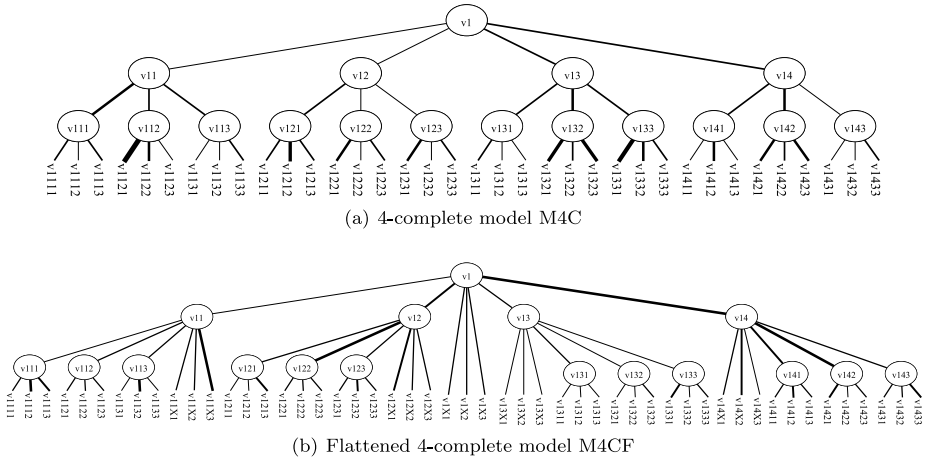
We first compare the algorithms on synthetic data.

5.1.1 The setups

One objective of our experimental work is to demonstrate the scalability of BI. To this end, we created several generative models with different numbers of observed variables. The simplest one is shown in Fig. 3(a). It consists of 3 levels of latent variables and one level of observed variables. Each latent variable has exactly 4 neighbors. The model is hence called the *4-complete* model. We denote it as *M4C*. It contains 36 observed variables. Two other models were created by adding latent variables to levels 2 and 3 and observed variables to level 4 so that each latent variable has exactly 5 and 7 neighbors respectively. Those two models are called the *5-complete* and *7-complete* models and will be denoted as *M5C* and *M7C*. They contain 80 and 252 observed variables respectively.

The models M4C, M5C and M7C are not flat because latent variables at level 1 and 2 are not directly connected to observed variables. Three flat models were created by adding more observed variables to the model so that each latent variable at level 1 and 2 has the same number of observed neighbors as a latent variable at level 3. The resulting flat models

[1] Key programs and data used in this paper are available at http://www.cse.ust.hk/~lzhang/ltm/index.htm.



(a) 4-complete model M4C

(b) Flattened 4-complete model M4CF

Fig. 3 Two of the generative models: (**a**) shows the 4-complete model (M4C). It consists of 3 levels of latent variables and one level of observed variables. Each latent variable has exactly 4 neighbors. The total number of observed variables is 36. (**b**) shows a flat model obtained from M4C by adding 3 observed variables to each of the latent variables on levels 1 and 2. It is called M4CF. It contains 51 observed variables

are denoted as *M4CF*, *M5CF* and *M7CF* respectively and they contain 51, 104, and 300 observed variables. M4CF is shown in Fig. 3(b). The numbers n of observed variables in the 6 generative models are summarized in the following table:

Models	M4C	M4CF	M5C	M5CF	M7C	M7CF
n	36	51	80	104	252	300

In the six models, cardinalities of the variables (observed and latent) were set to 2. The model parameters were randomly generated so that the normalized MI (Strehl et al. 2002) between each pair of neighboring nodes is between 0.05 and 0.75. From each generative model, a training set of 5,000 samples and a testing set of 5,000 samples were obtained. Each sample contains values for all the observed variables. It does not contain values for latent variables.

Each algorithm was run on the training set for 10 times. All experiments were conducted on a desktop machine. The maximum time allowed was 60 hours. All the algorithms have parameters that the user needs to set. For the previous algorithms, we use the default settings given by the authors. For BI, we always set δ at 3 except when investigating its impact (Sect. 5.1.3). For the call of EM in line 27, we used 64 random starting points and the maximum iteration was set at 100. For the calls of EM in other places, we used 32 random starting points and the maximum iteration was set at 32.

The model m learned by an algorithm from a training set is evaluated using the following metrics:

1. The Robinson-Foulds (RF) distance (Robinson and Foulds 1981) that measures how much the structure of m deviates from the structure of the corresponding generative model m_0 is computed as follows:

$$d_{RF}(m, m_0) = \frac{|C(m) - C(m_0)| + |C(m_0) - C(m)|}{2}, \tag{7}$$

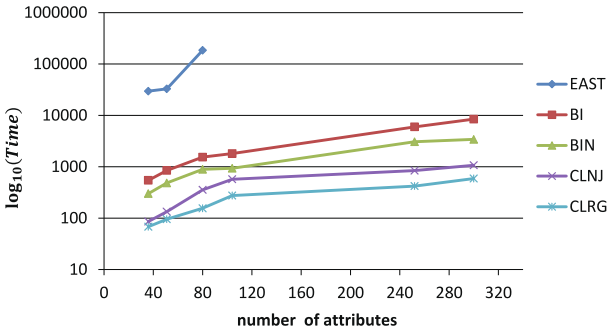


Fig. 4 Running time (seconds) of the algorithms on the 6 training sets. The statistics were collected on a desktop machine. All the training sets contain the same number (5,000) of data cases, but they involve different number of attributes

where $C(m)$ denotes the set of bipartitions defined by all edges in tree m. For example, in Fig. 2, removing the edge between Y_1 and Y_2 in model m_2 separates the observed variables into two subsets $\{X_1, X_2, X_3\}$ and $\{X_4, X_5\}$. So this edge defines a bipartition $X_1X_2X_3|X_4X_5$. $C(m)$ is the set of all such bipartitions defined by all edges in tree m. Term $|C(m) - C(m_0)|$ represents the number of bipartitions that appear in $C(m)$ but not in $C(m_0)$. Term $|C(m_0) - C(m)|$ represents the number of bipartitions that appear in $C(m_0)$ but not in $C(m)$. The sum of this two terms is the number of bipartitions that differ between tree m and tree m_0.

2. The *empirical KL divergence* of m from the generative model m_0 is computed from the testing set $D_{\texttt{test}}$ as follows:

$$KL(m_0, m|D_{\texttt{test}}) = \frac{1}{N_{\texttt{test}}}\left(\sum_{d\in D_{\texttt{test}}} \log P(d|m_0) - \sum_{d\in D_{\texttt{test}}} \log P(d|m)\right), \quad (8)$$

where $N_{\texttt{test}}$ is the size of the testing set. Note that the first term inside the parentheses is the loglikelihood of m_0 on the testing set and the second is that of m. The second term measures how well the model m predicts unseen data.

5.1.2 The results

Running time statistics are shown in Fig. 4. We see that BI is much more efficient than EAST. On the M4C, M4CF and M5C data sets, BI took only 9, 14 and 25 minutes while EAST took 8, 9 and 51 hours respectively. BI was about 55, 39 and 120 times faster than EAST. EAST did not finish in 60 hours on the other three data sets, while BI took 30 minutes, 1.7 hours and 2.4 hours respectively. Those results indicate that BI scales up fairly well. However, BI is not as efficient as BIN, CLNJ and CLRG. In our experiments, it was several times slower than the alternative algorithms.

Table 1 shows the performances of the algorithms on data sampled from the three flat generative models M4CF, M5CF and M7CF. The RF values for BIN are missing because it produced forests rather than trees, and RF is not defined for forests. On the M4CF data, EAST found the best models. The models obtained by BI are also of high quality. They are better than those produced by BIN, CLNJ and CLRG both in terms of empirical KL and RF values. The differences between BI and the three alternative algorithms are more pronounced on the M5CF and M7CF data. Those results indicate that BI was significantly better in recovering the structures of the generative models, and the models it obtained can predict unseen data much better than those produced by the three alternative algorithms.

Table 1 Performances of LTM learning algorithms on data sets from flat models

	M4CF		M5CF		M7CF	
	RF	emp KL	RF	emp KL	RF	emp KL
BIN	–	0.15 ± 0.01	–	0.56 ± 0.01	–	2.82 ± 0.07
CLNJ	8.50 ± 0.00	0.06 ± 0.00	39.50 ± 0.00	0.14 ± 0.02	49.50 ± 0.00	0.17 ± 0.01
CLRG	3.00 ± 0.00	0.07 ± 0.00	22.50 ± 0.00	0.09 ± 0.01	11.50 ± 0.00	0.35 ± 0.00
EAST	$\mathbf{0.50 \pm 0.00}$	$\mathbf{0.02 \pm 0.00}$	–	–	–	–
BI	2.60 ± 0.32	0.04 ± 0.00	$\mathbf{2.15 \pm 0.34}$	$\mathbf{0.03} \pm 0.00$	$\mathbf{4.00 \pm 0.62}$	$\mathbf{0.12 \pm 0.01}$

Table 2 Performances of LTM learning algorithms on data sets from non-flat models

	M4C		M5C		M7C	
	RF	emp KL	RF	emp KL	RF	emp KL
BIN	–	0.03 ± 0.00	–	0.11 ± 0.02	–	0.79 ± 0.19
CLNJ	15.00 ± 0.00	0.03 ± 0.00	23.00 ± 0.00	0.08 ± 0.01	49.50 ± 0.00	0.30 ± 0.01
CLRG	10.50 ± 0.00	0.03 ± 0.00	10.50 ± 0.00	0.09 ± 0.01	17.50 ± 0.00	0.34 ± 0.00
EAST	$\mathbf{5.20 \pm 0.79}$	0.02 ± 0.00	$\mathbf{0.75 \pm 0.42}$	$\mathbf{0.03 \pm 0.00}$	–	–
BI	5.50 ± 0.00	$\mathbf{0.01 \pm 0.00}$	8.35 ± 0.94	0.08 ± 0.01	$\mathbf{5.00 \pm 0.00}$	$\mathbf{0.18 \pm 0.02}$

BI is restricted to find flat LTMs. How does this restriction influence the performance of BI when the generative models are not flat? To answer the question, we show in Table 2 the results on the data sampled from the three non-flat models. EAST found the best models on M5C data. In terms of empirical KL, the models obtained by BI are of similar quality as those produced by the other three alternative methods on the M4C and M5C data, and are clearly better on the M7C data. In terms of RF values, BI performed much better than the other three alternative methods in all cases. This means that, despite the restriction imposed on it, BI recovered the generative structures much better than the three alternative algorithms BIN, CLNJ and CLRG. The results on real-world data to be presented in the next subsection also show that the restriction does not put BI at a disadvantage relative to other algorithms.

The PTR-motivated algorithms CLNJ and CLRG require that all the variables (observed and latent) have the same cardinality. The other algorithms, including BIN, EAST and BI, do not have the restriction and allow the cardinalities of variables to vary. To investigate the impact of this restriction, we created three other models by modifying M4CF, M5CF and M7CF. Specifically, the cardinalities of the latent variables at level 1 and 3 were increased from 2 to 3. The resulting models are denoted as M4CF1, M5CF1 and M7CF1. Data were sampled from those models as before and the algorithms were run on the data. For the PTR-motivated algorithms, the cardinalities of latent variables are automatically set to 2 since they should be the same as that of the observed variables as the algorithms required. For other algorithms, the cardinalities are determined automatically during the learning.

The results are presented in Table 3. We see that CLNJ and CLRG performed substantially worse than before relative to BI. On the M7CF1 data, for instance, the empirical KL values for CLNJ and CLRG are several times larger than that for BI. The results indicate that the restriction imposed by the PTR-algorithms put them at a severe disadvantage as compared to BI when the latent variables in the 'true model' have different cardinalities.

Table 3 Performances of LTM learning algorithms on data sets from flat models where cardinalities of latent variables vary

	M4CF1		M5CF1		M7CF1	
	RF	emp KL	RF	emp KL	RF	emp KL
BIN	–	0.18 ± 0.01	–	0.84 ± 0.01	–	3.48 ± 0.03
CLNJ	19.00 ± 0.00	0.08 ± 0.01	36.00 ± 0.00	0.23 ± 0.01	98.00 ± 0.00	1.39 ± 0.00
CLRG	9.50 ± 0.00	0.07 ± 0.00	21.50 ± 0.00	0.24 ± 0.01	52.90 ± 0.52	1.52 ± 0.03
EAST	4.00 ± 0.00	$\mathbf{0.04 \pm 0.01}$	–	–	–	–
BI	$\mathbf{2.75 \pm 1.06}$	$\mathbf{0.04 \pm 0.01}$	$\mathbf{12.45 \pm 1.40}$	$\mathbf{0.14 \pm 0.02}$	$\mathbf{7.10 \pm 1.47}$	$\mathbf{0.24 \pm 0.06}$

Table 4 Impact of δ on the performance of BI. For this set of experiments, only data sampled from the model M5CF1 were used

	RF	emp KL	Time (sec)
$\delta = 1$	14.30 ± 2.57	0.15 ± 0.03	2,181
$\delta = 3$	12.45 ± 1.40	0.14 ± 0.02	2,205
$\delta = 5$	13.20 ± 1.21	0.13 ± 0.01	2,101
$\delta = 10$	11.95 ± 1.07	0.13 ± 0.01	2,236

5.1.3 Impact of δ and sample size on BI

BI has one parameter that the user has to set, namely the threshold δ for the UD-test. To get some intuition about the impact of the parameter, take another look at Algorithm 1. We see that the larger δ is, the harder it is to satisfy the condition at line 10, and the longer the set S would keep expanding, which often implies larger sibling clusters.

In all the experiments reported so far, δ was set at 3 as suggested by Kass and Raftery (1995). In the context of this paper, the use of the value 3 implies that we would conclude the correlations among attributes in the set S can be properly modeled using one single latent variable if there is no *strong* evidence pointing to the opposite. Two other possible values 1 and 5 for δ were also suggested by Kass and Raftery (1995). The use of those values would mean, respectively, to draw the same conclusion when there is no *positive* or *very strong* evidence pointing to the opposite.

To investigate the impact of δ, we tried both 1 and 5 in addition to 3. The value 10 was also included as a reference. The results are shown in Table 4. We see that the choice of δ did not influence performance of BI significantly in terms of RF values, empirical KL, and running time.

In all the experiments reported so far, the sample size for the training set was 5,000. To investigate the impact of sample size, we sampled from the model M5CF1 two other training sets that contain 1,000 and 10,000 data cases respectively. So we have three different training sets for the model. Experiments were carried out on all those three data sets. The results are shown in Table 5. It is clear that model quality increases almost monotonically with sample size, and running times increases with it more or less linearly.

Table 5 Impact of sample size on the performances of BI. For this set of experiments, only the data sampled from the model M5CF1 were used

Sample size	RF	emp KL	Time (sec)
1k	18.85 ± 1.33	0.53 ± 0.01	652
5k	12.45 ± 1.40	0.14 ± 0.02	2,205
10k	14.55 ± 1.04	0.11 ± 0.01	4,431

Table 6 Information about real-world data sets used in our experiments

	# attributes	Cardinality	Size (train)	Size (test)
Coil-42	42	2–9	5,822	4,000
Alarm	37	2–4	1,000	1,000
News-100	100	2	8,121	8,121
WebKB	336	2	830	208

5.2 Comparisons on real-wold data

We now compare the algorithms on four real-world data sets: Coil-42 (Zhang et al. 2008a), Alarm (Mourad et al. 2013), WebKB[2] and News-100.[3] Information about the version of the data sets used in our experiments is shown in Table 6. Note that all attributes in WebKB and News-100 have 2 possible values, while attributes in Coil-42 and Alarm have between 2 to 9 possible values.

We tested the algorithms on the four data sets. Note that the CLNJ and CLRG require that all the observed variables have the same cardinalities. As such they are not applicable to Coil-42 and Alarm. For the applicable cases, each algorithm was run on each data set for 10 times. The average running time is reported in Table 7. The quality of a learned model is measured using BIC score on the training set and loglikelihood on the testing set. The statistics are shown in Table 8.

We see that EAST found best models on Coil-42 data both in terms of BIC score and loglikelihood. For the Alarm data, the models found by EAST also have higher BIC scores than those obtained by BI, indicating that they fit the training data better. But their loglikelihood scores on the test set of the Alarm data are slightly lower than that of BI. For this two data that EAST can handle, it took 11 hours to process the Coil-42 data and 1.7 hours to process the Alarm data while BI only took about 9 minutes and 3 minutes respectively. For News-100 data and WebKB data, BI was able to process within 1 hour while EAST did not finish in 60 hours.

In comparison with BIN, CLNJ and CLRG, BI was several times slower. However, the models that it obtained are much better in terms of both BIC score on training data and loglikelihood on test data. Those results indicate that the restriction to flat models does not put BI at a disadvantage relative to the alternative algorithms when it comes to the analysis of real-world data. However, the restriction that all attributes must have the same cardinality makes the PTR-algorithms inapplicable to real world data sets such as Coil-42 and Alarm.

[2] Available from http://www.cs.cmu.edu/afs/cs.cmu.edu/project/theo-20/www/data/.

[3] Available from http://cs.nyu.edu/~roweis/data/20news_w100.mat.

Table 7 Average running time (seconds) of LTM learning algorithms on the real-world data sets

	Coil-42	Alarm	News-100	WebKB
BIN	506	97	1,941	628
CLNJ	–	–	917	193
CLRG	–	–	594	144
EAST	38,355	6,122	–	–
BI	528	153	2,494	2,254

Table 8 Performances of LTM learning methods on the real world data

	BIC (train)			
	Coil-42	Alarm	News-100	WebKB
CLRG	–	–	$-116{,}973 \pm 042$	$-66{,}492 \pm 11$
CLNJ	–	–	$-117{,}015 \pm 050$	$-66{,}723 \pm 11$
BIN	$-53{,}568 \pm 000$	$-13{,}295 \pm 000$	$-119{,}331 \pm 000$	$-70{,}473 \pm 00$
EAST	$\mathbf{-51{,}482 \pm 075}$	$\mathbf{-11{,}551 \pm 069}$	–	–
BI	$-52{,}461 \pm 143$	$-11{,}760 \pm 110$	$\mathbf{-116{,}451 \pm 125}$	$\mathbf{-65{,}854 \pm 90}$
	Loglikelihood (test)			
	Coil-42	Alarm	News-100	WebKB
CLRG	–	–	$-116{,}242 \pm 045$	$-16{,}349 \pm 05$
CLNJ	–	–	$-116{,}059 \pm 058$	$-16{,}365 \pm 03$
BIN	$-35{,}861 \pm 000$	$-16{,}221 \pm 000$	$-117{,}920 \pm 000$	$-16{,}794 \pm 00$
EAST	$\mathbf{-34{,}344 \pm 057}$	$-10{,}706 \pm 100$	–	–
BI	$-35{,}217 \pm 097$	$\mathbf{-10{,}666 \pm 106}$	$\mathbf{-115{,}619 \pm 185}$	$\mathbf{-16{,}112 \pm 17}$

5.3 Summary

We have compared BI with previous algorithms for learning LTMs on a host of data sets. The number of attributes ranges from dozens to hundreds. In comparison with the search algorithm EAST, the advantage of BI is efficiency. While EAST could handle only small data sets (M4C, M4CF, M5C, Coil-42 and Alarm), BI was able to deal with all the data sets. On those small data sets, BI found models of similar quality as those obtained by EAST. In comparison with the AC-based algorithm BIN and the PTR-motivated algorithms CLNJ and CLRG, the advantage of BI is model quality. In fact, BI found better models in most cases. On the other hand, it was several times slower.

6 Multi-partition clustering

Clustering is a primary and traditional task in knowledge discovery and data mining. Conventional clustering methods usually assume that there is only one true clustering in a data set. This assumption does not hold for many real-world data which are multifaceted and can be meaningfully clustered in multiple ways. For example, a student population can be clustered in one way based on course grades and in another way based on extracurricular activities. Movie reviews can be clustered based on both sentiment (positive or negative) and

genre (comedy, action, war, etc.). The respondents in a social survey can be clustered based on demographic information or views on social issues.

In this section, we briefly survey previous methods proposed to produce multiple clusterings. We refer to them as *multi-partition clustering* (MPC) methods. MPC is not to be confused with multi-view clustering (Bickel and Scheffer 2004), which makes use of multiple views of data to improve the quality of one single clustering solution. MPC methods, according to the way that partitions are found, can be divided into two categories: *sequential MPC* methods and *simultaneous MPC* methods.

Sequential MPC methods produce multiple partitions sequentially. One such method is known as *alternative clustering* (Cui et al. 2007; Gondek and Hofmann 2007; Qi and Davidson 2009; Bae and Bailey 2006). It aims at discovering a new clustering that is different from a previously known clustering. The key issue is how to ensure the novelty of the new clustering with respect to the previous clustering. Gondek and Hofmann (2007) adopt the information bottleneck framework and maximize the mutual information between the new clustering and the attributes, conditioned on the previous clustering. Bae and Bailey (2006) suggest that two objects placed into the same group in the previous clustering should be pushed into different groups in the new clustering. Cui et al. (2007) project data into a space that is orthogonal to the previous solution and find a new clustering in the projected space. Qi and Davidson (2009) first transform data with respect to the previous clustering and then cluster the projected data. During transformation, constraints are imposed to pull data points away from their original clusters.

Simultaneous MPC methods, on the other hand, produce multiple partitions simultaneously. Both distance-based and model-based methods have been proposed. The distance-based methods require as inputs the number of partitions and the number of clusters in each partition. They try to optimize the quality of each individual partition while keeping different partitions as dissimilar as possible. Jain et al. (2008) propose a method to find two clusterings simultaneously by iteratively minimizing the sum-squared errors of each clustering along with the correlation between the two clusterings. Niu et al. (2010) try to find multiple clusterings by optimizing an objective function that has one spectral-clustering term for each clustering, plus a term that penalizes the similarity among different clusterings. Dasgupta and Ng (2010) use the suboptimal solutions of spectral clustering as multiple clusterings.

Model-based MPC methods fit data with probabilistic models that contain multiple latent variables. Each latent variable represents a soft partition and can be viewed as a hidden dimension of data. Because of this, model-based simultaneous MPC is sometimes also called *multidimensional clustering* (Chen et al. 2012). Unlike distance-based methods, model-based methods can automatically determine the number of partitions and the number of clusters in each partition based on statistical principles. Galimberti and Soffritti (2007) determine the number of partitions and the composition of each partition through greedy search guided by a scoring function. Guan et al. (2010) propose a nonparametric Bayesian approach. Prior distributions are placed on both attribute partitions and object partitions using Dirichlet process. The number of partitions and partition composition are determined from the posterior distributions. Both Galimberti and Soffritti (2007) and Guan et al. (2010) assume that each latent variable, which gives one *facet* of data, is associated with a distinct subset of attributes, and that the latent variables are mutually independent. In contrast, Chen et al. (2012) uses LTMs where that latent variables are not assumed independent and the dependence among them are explicitly modeled.

Among the methods mentioned above, the method proposed by Chen et al. (2012) is designed for discrete data. Bae and Bailey (2006) presented two versions of their algorithm

which can handle categorical and continuous data separately. The methods presented in Gondek and Hofmann (2007) can handle binary data and continuous data. The other algorithms mainly work with continuous data.

MPC is related to another research area known as *subspace clustering* (Parsons et al. 2004). Here, we use the term *subspace clustering* in a general manner. It includes pattern-based clustering, bi-clustering, and correlation clustering. Kriegel et al. (2009) give a good survey on these topics. Subspace clustering attempts to find all clusters in all subspaces. A cluster is defined as a group of data points that are densely clustered in the subspace determined by a subset of attributes. To explain the relationship between subspace clustering and MPC, we note that the clusters resulting from MPC can be arranged into columns such that clusters in each column constitute a partition of all the data points and all can be characterized using the same subset of attributes. In contrast, subspace clustering imposes no such constraints on the collection of clusters it produces. It usually yields a large number of clusters.

7 LTMs for multi-partition clustering

In this section, we demonstrate with an example how LTMs can be used for MPC and discuss some practical issues.

The example we use is the WebKB data used in Sect. 5.2. It consists of web pages collected in 1997 from the computer science departments of 4 universities, namely Cornell, Texas, Washington and Wisconsin. The web pages were originally divided into seven categories. The categories which contain very few web pages or do not have clear class labels are removed in preprocessing. Only four categories of web pages remain, namely `student`, `faculty`, `project` and `course`. There are totally 1041 pages in the collection. Stop words and words with a low occurrence frequency are also removed in preprocessing. Afterwards, there are totally 336 different words left in the whole collection. Each of them is regarded as an attribute. It takes value 1 when it appears in a web page and 0 otherwise.

The WebKB data has two class labels. One identifies the university from which a web page is collected, and the other denotes the category to which it belongs. The class labels were removed prior to analysis.

Performing latent tree analysis on a data set can result in a large number of latent variables. Figure 5 shows part of the model learned from the WebKB data using the BI algorithm. There are 75 latent variables in the model. Theoretically, each latent variable represents a soft partition of the data. From the application point of view, several questions arise: (1) What exactly constitutes a partition? (2) What is the meaning of the partition? (3) How can one quickly identify interesting partitions among all the partitions? A software tool called *Lantern*[4] has been created to help finding answers to those questions.

The meaning of a latent variable is determined based on how it is related to observed variables. Lantern allows one to see which observed variables are closely correlated with a latent variable using the concept of *information curves*. We introduce the concept through an example. Consider the latent variable Y_{28} from the WebKB model. It has 4 values and hence represents a soft partition of the data into four clusters. We will refer to the partition as the Y_{28} partition. The information curves of Y_{28} are shown in Fig. 6(a). There are two curves. Suppose there are totally n attributes. The lower curve shows the *pairwise mutual information* (PMI) $I(Y_{28}; X_i)$ between Y_{28} and each attribute X_i $(i = 1, 2, \ldots, n)$. The at-

[4] Available from http://www.cse.ust.hk/~lzhang/ltm/index.htm.

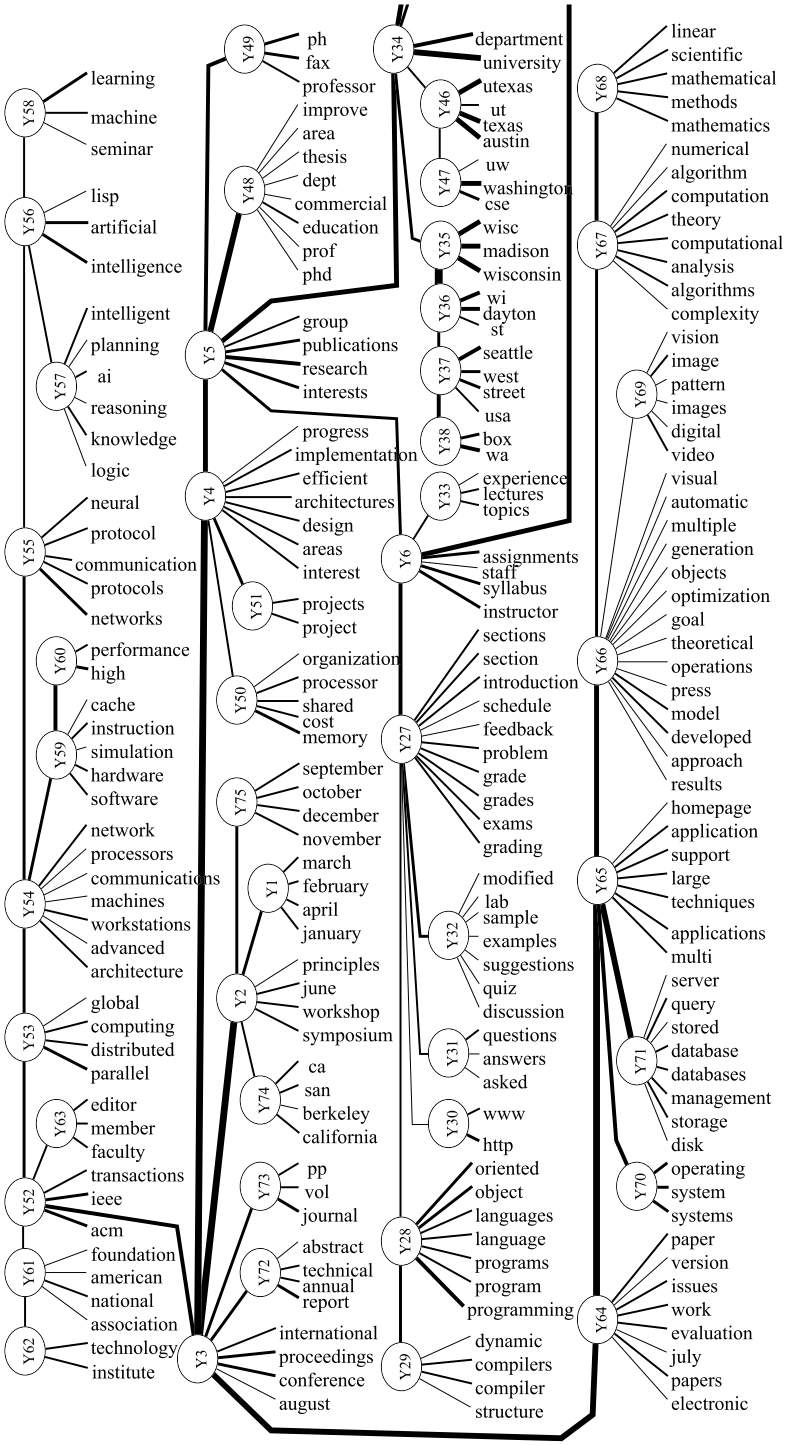
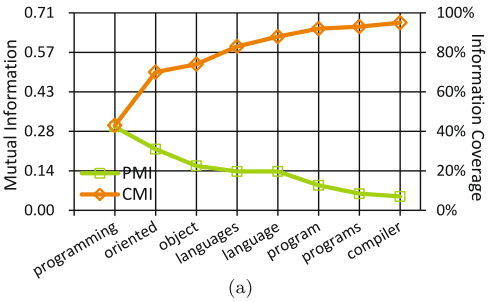


Fig. 5 Part of the latent tree model learned from the WebKB data using BI



(a)

Y_{28}: IC=95%

cluster	1	2	3	4
programming	1.00	0.77	0.06	0.72
oriented	0.16	0.00	0.00	1.00
object	0.51	0.05	0.02	0.91
languages	0.12	0.41	0.01	0.50
language	0.81	0.34	0.05	0.61
program	0.92	0.29	0.09	0.31
programs	0.53	0.23	0.04	0.17
compiler	0.31	0.18	0.01	0.17
Size	0.04	0.21	0.69	0.06

(b)

Fig. 6 (**a**) Information curves of latent variable Y_{28}. The *left Y-axis* indicates the values of mutual information, while the *right Y-axis* indicates the value of information coverage. (**b**) Occurrence probability of the top words in the four clusters

tributes are sorted in decreasing order of PMI. We see that Y_{28} is most highly correlated with word variables `programming`, `oriented`, ..., etc. in that order. This means that the clusters in Y_{28} partition differ the most on those variables.

The upper curve shows the *cumulative mutual information* (CMI) $I(Y_{28}; X_1{-}X_i)$ $(i = 2, 3, \ldots, n)$ between Y_{28} and the first i attributes. It increases monotonically with i. Intuitively, the CMI value is a measure of how much the differences among the clusters in the Y_{28} partition are captured by the first i attributes. In particular, $I(Y; X_1{-}X_n)$ is a measure of how much the differences among the clusters are captured by all the attributes. The ratio

$$I(Y; X_1{-}X_i)/I(Y; X_1{-}X_n) \tag{9}$$

is called the *information coverage* (IC) of the first i attributes.[5] It is a relative measure of how much the characteristics of the Y_{28} partition is captured by the first i attributes. The information coverage of the 8 word variables shown in Fig. 6(a) exceeds 95 %. Hence we can say that the partition is primarily based on those variables. It is then clear that the partition is about the occurrence/absence of words related to programming.

Lantern also allows one to see the statistical characteristics of each cluster by giving its *class conditional probability distributions* (CCPDs). The CCPDs for the four clusters in the Y_{28} partition are given in Fig. 6(b). The value 0.77 at line 2 and column 3, for instance, is the probability that word `programming` appears in cluster $Y_{28} = 2$. We see that the words `programming`, `oriented` and `object` occur with high probability in the cluster $Y_{28} = 4$. Hence we interpret it as the collection of web pages on objected-oriented programming (OOP). In the cluster $Y_{28} = 2$, the word `programming` occurs with high probability, but `oriented` and `object` almost do not occur at all. Hence we interpret it as a collection of web pages on programming but not OOP. Those might be web pages of OOP courses and of introductory programming courses respectively. The cluster $Y_{28} = 1$ seems to correspond to web pages of other courses that involve programming, while $Y_{28} = 3$ seems to mean web pages not on programming.

Further, Lantern allows one to check the membership of each cluster. It facilitates both soft assignment and hard assignment. In soft assignment, it shows the posterior distributions of latent variables for each data case. In hard assignment, it assigns each data case to the value of a latent variable (which represents a cluster) that has the maximum posterior probability.

[5]The quantity is approximately calculated by sampling.

Table 9 Word variables that have highest MI with Y_{55} and Y_{57} and their distributions in the clusters given by the two latent variables

Y_{55}: IC = 96 %				Y_{57}: IC = 100 %		
Cluster	1	2	3	Cluster	1	2
networks	0.88	0.43	0.01	intelligence	0.03	0.59
communication	0.03	0.39	0.01	artificial	0.03	0.58
neural	0.77	0.00	0.01	knowledge	0.03	0.60
protocols	0.00	0.23	0.00	ai	0.02	0.49
protocol	0.00	0.19	0.00	intelligent	0.01	0.34
intelligence	0.68	0.03	0.05	planning	0.01	0.32
artificial	0.66	0.03	0.05	reasoning	0.01	0.33
parallel	0.20	0.43	0.11	learning	0.05	0.31
network	0.14	0.33	0.06	logic	0.05	0.30
architecture	0.15	0.33	0.07	neural	0.02	0.19
high	0.17	0.38	0.09	lisp	0.01	0.11
performance	0.17	0.36	0.09	networks	0.09	0.26
Size	0.04	0.14	0.82	Size	0.93	0.07

Using Lantern, we were able to quickly identify interesting partitions given by the WebKB model. Table 9 shows two examples in details. We see that Y_{55} represents a partition based on the occurrence/absence of words such as `networks`, `communication`, `neural`, `protocols`, `artificial` and `intelligence`. The cluster $Y_{55} = 1$ seems to correspond to web pages on neural networks, while $Y_{55} = 2$ seems to correspond to web pages on networking and high performance computing area. Similarly, $Y_{57} = 2$ seems to correspond to web pages related to AI, while $Y_{57} = 1$ seems to identify web pages that are not related to AI.

Table 10 shows more intuitively interesting latent variables. CCPDs are not given here to save space. The latent variables Y_{73} and Y_{49} represent partitions based on words that usually appear on faculty homepages; Y_{69}, Y_{71} and Y_{59} represent partitions based on words that distinguish different research areas. Y_{10}, Y_{47}, Y_{46} and Y_{35} represents partitions based on words that identify the four universities from which the data were collected. Finally, Y_6 represents a partition based on words that usually appear on course web pages.

8 Comparisons of MPC methods on labeled data

In this and the next sections, we compare BI, as an MPC algorithm, with previous MPC algorithms on labeled data and unlabeled data respectively. Three previous MPC algorithms Orthogonal Projection (OP) (Cui et al. 2007), Singular Alternative Clustering (SAC) (Qi and Davidson 2009) and DK-means (DK) (Jain et al. 2008). OP and SAC are representative *sequential MPC* algorithms, while DK is a representative distance-based *simultaneous MPC* algorithm.

In this section, we compare the algorithms on the WebKB data. Each data case in this data set has two class labels. Hence the are two true partitions. The first partition divides the data into 4 classes according to where the web pages were collected. The four classes are `Cornell`, `Texas`, `Washington` and `Wisconsin`. The second partition divides the

Table 10 More latent variables that are intuitively meaningful. For each latent variable, a list of word variables are given. Those are the variables that have the highest MI with the latent variables. In each case, the information coverage of the variables listed exceeds 95 %

Faculty homepage		Research areas		
Y_{73}	Y_{49}	Y_{69}	Y_{71}	Y_{59}
journal	ph	image	management	high
pp	fax	video	database	performance
vol	research	vision	systems	hardware
conference	professor	images	system	software
proceedings	university	pattern	databases	instruction
international	publications	digital	storage	parallel
symposium	interests	applications	large	network
acm			support	architecture
workshop			applications	cache

Universities				Course
Y_{10}	Y_{47}	Y_{46}	Y_{35}	Y_6
ithaca	washington	utexas	wisc	assignments
ny	cse	texas	madison	instructor
cornell	uw	austin	wisconsin	hours
upson		ut	dayton	syllabus
hall			wi	class
				grading
				pm
				lecture
				homework

data into another 4 classes according to the nature of the web pages. The four classes are `student`, `faculty`, `project` and `course`.

In our experiments, the class labels were first removed. The algorithms were then run on the resulting unlabeled data. DK can only produce two partitions. In our experiments, it was instructed to find two partitions each with four clusters. For OP, we, following the authors, first ran K-means (MacQueen 1967) to find a partition of four clusters, and then continue to run OP to find one more partition with four clusters. The same was done for SAC. So, all the three alternative methods were all told to find two partitions each with four clusters.

Table 11 shows some information about the partitions. We see that the partitions obtained by the three algorithms are similar in terms of the variables that they rely on. One of the partitions is primarily based on words that often appear on course web pages, while the other is primarily based on words that identify the four universities.

We compare the performances of BI and the alternative algorithms quantitatively by considering how well they have recovered the ground truth. It is clear from Table 10 that BI has recovered the four university classes. However, they are given in the form of four latent variables instead of one. For comparability, we transform the true university partition into four logically equivalent binary class partitions. Each binary partition divides the web pages according to whether they are from a particular university. The same transformation is applied to the other true partition. After transformation, we calculate the *normalized mutual information* (*NMI*) (Strehl et al. 2002) between each true binary partition C and each latent variable Y as follows:

Table 11 Information about partitions obtained by DK, OP and SAC on the unlabeled WebKB data. Top 20 word variables that have the highest MI with the partitions are shown. The information coverage of these variables exceeds 80 % in all cases

DK		OP		SAC	
Partition 1	Partition 2	Partition 1	Partition 2	Partition 1	Partition 2
hours	madison	university	wisc	assignments	austin
assignments	austin	computer	utexas	hours	madison
instructor	dayton	department	cornell	instructor	university
lecture	texas	assignments	asutin	lecture	texas
syllabus	wi	research	madison	research	wisconsin
pm	wisconsin	hours	texas	syllabus	science
class	utexas	science	wisconsin	class	utexas
grading	wisc	instructor	washington	homework	computer
homework	seattle	lecture	wi	pm	wisc
exam	computer	publications	dayton	publications	department
thursday	department	class	cse	grading	wi
acm	washington	pm	ithaca	conference	dayton
office	university	conference	seattle	proceedings	seattle
monday	wa	grading	wa	exam	washington
research	street	syllabus	upson	handouts	street
wednesday	west	office	ut	thursday	cornell
final	ut	acm	street	monday	wa
handouts	science	homework	west	due	west
assignment	box	interests	ny	office	ut
conference	st	exam	uw	notes	research

Table 12 *NMI* values between the true binary partitions and the closest partitions obtained by each algorithm. They show how well the algorithms have recovered the true partitions

	DK	SAC	OP	BI
`course`	0.43 ± 0.01	0.47 ± 0.01	0.47 ± 0.02	$\mathbf{0.63 \pm 0.02}$
`faculty`	0.18 ± 0.04	0.17 ± 0.07	0.18 ± 0.01	$\mathbf{0.30 \pm 0.01}$
`project`	0.04 ± 0.00	0.04 ± 0.00	0.05 ± 0.04	$\mathbf{0.07 \pm 0.00}$
`student`	0.18 ± 0.00	0.20 ± 0.00	0.20 ± 0.01	$\mathbf{0.25 \pm 0.01}$
`cornell`	0.22 ± 0.15	0.09 ± 0.02	$\mathbf{0.36 \pm 0.24}$	0.34 ± 0.01
`texas`	0.31 ± 0.18	0.20 ± 0.20	0.45 ± 0.23	$\mathbf{0.61 \pm 0.02}$
`washington`	0.22 ± 0.13	0.41 ± 0.23	$\mathbf{0.56 \pm 0.25}$	0.55 ± 0.12
`wisconsin`	0.38 ± 0.12	0.16 ± 0.12	0.45 ± 0.13	$\mathbf{0.55 \pm 0.11}$

$$NMI(C;Y) = \frac{I(C;Y)}{\sqrt{H(C)H(Y)}}$$

where $H(.)$ denotes the entropy of a variable. *NMI* ranges from 0 to 1, with a higher value meaning a closer match between C and Y. Then we match each true binary partition with the latent variable with which it has the highest *NMI*, and we use the *NMI* between the pair as a measure of how well BI has recovered the partition. The results are shown in the last column of Table 12. The average was taken over 10 runs.

Each of the partitions found by the alternative method consists of four clusters. For compatibility, it is also transformed into four logically equivalent binary partitions. Those partitions are matched up with the true binary partitions. The *NMI* between the matched pairs are also shown in Table 12.

We see that the *NMI* values for BI are the highest with regard to 6 of the 8 true binary partitions. They are significantly higher than those for the alternative methods with regard to `course`, `faculty`, `student`, `texas` and `wisconsin`. This means that BI has recovered those true binary partitions much better than the other methods. The performance of BI is only slightly worse than OP with regard to the binary partitions `cornell` and `washington`.

To get a global picture, compare the results given in Table 11 and those presented in the previous section. It is clear that BI has found many more interesting partitions than the alternative algorithms. None of the alternative methods have found partitions similar to those represented by Y_{28}, Y_{55}, Y_{57}, Y_{69}, Y_{71} and Y_{59}.

9 Comparisons of MPC methods on unlabeled data

In this section we compare BI and other MPC algorithms on an unlabeled data set known as ICAC data. The data set originates from a survey in 2004 of public opinions on corruption in Hong Kong and the performance of ICAC—the anti-corruption agency of the city. After preprocessing, the data set consists of 31 attributes and 1200 records. The attributes correspond to questions in the survey.

9.1 Results on the ICAC data by BI

On the ICAC data, BI produced an LTM with 7 latent variables. It will be referred to as the *ICAC model*. The structure is shown in Fig. 7. Basic information about the latent variables is given in Table 13. For each latent variable, we list the observed variables that are highly correlated with it.

It is clear from Table 13 that Y_1 represents a partition of the data based on people's `view on change of corruption level`; Y_2 represents a partition based on `demographic information`; Y_3 represents a partition based on people's `tolerance toward corruption`; Y_4 represents a partition based on people's `view on the`

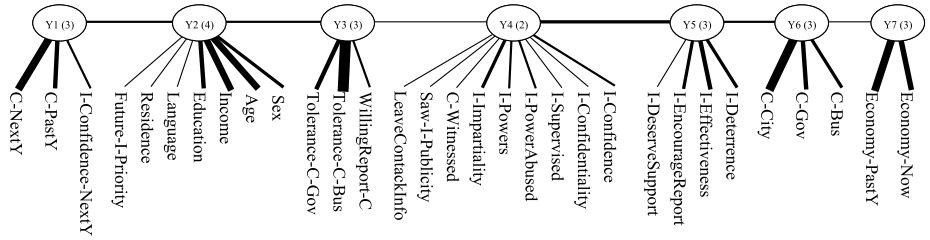


Fig. 7 The structure of LTM learned by BI from the ICAC data. The width of edges represent strength of probabilistic dependence. Abbreviations: C—Corruption, I—ICAC, Y—Year, Gov—Government, Bus—Business Sector. Meanings of attributes: Tolerance-C-Gov means 'tolerance towards corruption in the government'; C-City means 'level of corruption in the city'; C-NextY means 'change in the level of corruption next year'; I-Effectiveness means 'effectiveness of ICAC's work'; I-Powers means 'ICAC powers'; etc.

Table 13 Information about the latent variables in the ICAC model. For each latent variable, a list of observed variables is given. Those are the variables that have the highest MI with the latent variables. In each case, the information coverage of the variables listed exceeds 95 %

Y_1	C-NextY, C-PastY, I-Confidence-NextY
Y_2	Income, Age, Education, Sex
Y_3	Tolerance-C-Bus, Tolerance-C-Gov
Y_4	I-Impartiality, I-Confidence, I-PowerAbused, I-Effectiveness, I-Deterrence, I-Powers, I-EncourageReport, Saw-I-Publicity, I-Supervised
Y_5	I-Effectiveness, I-Deterrence, I-EncourageReport, I-Impartiality, I-Confidence
Y_6	C-City, C-Gov, C-Bus
Y_7	Economy-PastY, Economy-Now

`accountability of ICAC`; Y_5 represents a partition based on people's `view on the performance of ICAC`; Y_6 represents a partition based on people's `view on corruption level`; and Y_7 represents a partition based on people's `view on the economy`.

Let us examine the partitions given by Y_2 and Y_3. Y_2 has four states and hence it represents a partition of 4 clusters. The CCPDs are given in Table 14. We see that 99 % of the people in cluster $Y_2 = 4$ are under the age of 24. The average income is quite low. So $Y_2 = 4$ can be interpreted as `youngsters`. Cluster $Y_2 = 2$ only consists of women. There is no income (s_0) or the income is low. So it can be interpreted as `women with no/low income`. For the remaining two classes, the people in $Y_2 = 1$ have higher education and higher income than the people in $Y_2 = 3$. Hence $Y_1 = 1$ can be interpreted as `people with good education and good income`, while $Y_2 = 3$ as `people with poor education and average income`.

Y_3 has three states and hence it represents a partition of 3 clusters. The CCPDs are given in Table 15. It is clear that the three latent classes $Y_3 = 1$, $Y_3 = 2$, and $Y_3 = 3$ can be interpreted as classes of people who find corruption `tolerable`, `intolerable` and `totally intolerable` respectively.

In addition to identifying interesting partitions, BI also determines relationships between different partitions. As an example, consider the relationship between Y_2 and Y_3. The conditional probability $P(Y_3 \mid Y_2)$ is shown in Table 16. We see that $Y_2 = 1$ (`people with good education and good income`) is the class with the least tolerance towards corruption. In fact, 93 % (31 % + 62 %) of the people in this class think corruption intolerable or totally intolerable ($Y_3 = 2$ or 3). On the other hand, $Y_2 = 3$ (`people with poor education and average income`) is the class with more tolerance towards corruption. In fact, 39 % of the people in the class who find corruption tolerable ($Y_3 = 1$). The other two classes fall in between those two extreme classes.

9.2 Results on the ICAC data by other MPC algorithms

The ICAC data contains missing values. However, the MPC algorithms DK, OP and SAC do not allow missing values. Two options are tried to handle this issue. In the first option, data cases with missing value are simply removed. In the second option, the missing values are replaced with the most prevalent values. Two complete versions of the data set result in. They consist of 523 and 1,200 records respectively.

The three alternative MPCs were run on each of the two complete versions of the data. As in the previous section, the algorithms were instructed to find two partitions. Several

Table 14 Information about the partition represented by Y_2. The information coverage of the four variables shown is 98 %. The states of variables are: `Income`: s_0 (none), s_1 (less than 4k), s_2 (4–7k), s_3 (7–10k), s_4 (10–20k), s_5 (20–40k), s_6 (more than 40k); `Age`: s_0 (15–24), s_1 (25–34), s_2 (35–44), s_3 (45–54), s_4 (above 55); `Education`: s_0 (none), s_1 (primary), s_2 (Form 1–3), s_3 (Form 4–5), s_4 (Form 6–7), s_5 (diploma), s_6 (degree); `Sex`: s_0 (male), s_1 (female)

	$P(\cdot \mid Y_2)$	s_0	s_1	s_2	s_3	s_4	s_5	s_6
$Y_2 = 1$ Size: 0.37	`Income`	0.02	0	0.04	0.10	0.42	0.28	0.14
	`Age`	0.05	0.35	0.39	0.17	0.03		
	`Education`	0	0	0.04	0.41	0.09	0.09	0.37
	`Sex`	0.57	0.43					
	$P(\cdot \mid Y_1)$	s_0	s_1	s_2	s_3	s_4	s_5	s_6
$Y_2 = 2$ Size: 0.24	`Income`	0.43	0.29	0.24	0.04	0	0	0
	`Age`	0.03	0.08	0.41	0.35	0.13		
	`Education`	0.05	0.29	0.35	0.26	0.04	0	0.01
	`Sex`	0	1					
	$P(\cdot \mid Y_2)$	s_0	s_1	s_2	s_3	s_4	s_5	s_6
$Y_2 = 3$ Size: 0.22	`Income`	0.10	0.11	0.17	0.25	0.31	0.07	0
	`Age`	0	0.07	0.22	0.40	0.30		
	`Education`	0.02	0.29	0.43	0.19	0.05	0.01	0
	`Sex`	0.80	0.20					
	$P(\cdot \mid Y_2)$	s_0	s_1	s_2	s_3	s_4	s_5	s_6
$Y_2 = 4$ Size: 0.17	`Income`	0.02	0.78	0.08	0.09	0.03	0	0
	`Age`	0.99	0.01	0	0	0		
	`Education`	0	0	0.08	0.47	0.21	0.10	0.16
	`Sex`	0.50	0.50					

Table 15 Information about the partition represented by Y_3. States of the two observed variables: s_0 (totally intolerable), s_1 (intolerable), s_2 (tolerable), and s_3 (totally tolerable). The information coverage for these two attributes is around 99 %

$P(\cdot \mid Y_3)$	$P(Y_3 = 1) = 0.18$				$P(Y_3 = 2) = 0.24$				$P(Y_3 = 3) = 0.58$			
	s_0	s_1	s_2	s_3	s_0	s_1	s_2	s_3	s_0	s_1	s_2	s_3
`Tolerance-C-Bus`	0	0.08	0.84	0.08	0.02	0.97	0.02	0	1	0	0	0
`Tolerance-C-Gov`	0.31	0.24	0.41	0.04	0.54	0.46	0	0	0.97	0.02	0	0

Table 16 The conditional probability distribution $P(Y_3 \mid Y_2)$

	$Y_3 = 1$	$Y_3 = 2$	$Y_3 = 3$
$Y_2 = 1$	0.07	0.31	0.62
$Y_2 = 2$	0.18	0.14	0.68
$Y_2 = 3$	0.39	0.12	0.49
$Y_2 = 4$	0.17	0.40	0.42

Table 17 Information about the partitions found by DK, OP and SAC on the ICAC data. For each partition, we show the variables that have the highest MI with the partition. In each case, the information coverage of the variables listed exceeds 95 %

When data cases with missing values are removed	
DK-1	Age, Income, I-Confidence, I-Effectiveness, C-City, I-PowerAbused, I-Impartiality, I-Deterrence
DK-2	Tolerance-C-Gov, Tolerance-C-Bus, I-Impartiality, I-Powers, C-City, I-Confidence, I-PowerAbused, C-NextY, I-Deterrence, C-Bus, C-PastY
OP-1	Sex, C-PastY, C-NextY, Income, I-PowerAbused, WillingReport-C
OP-2	LeaveContackInfo, I-Supervised
SAC-1	Sex, C-PastY, C-NextY, Income, I-PowerAbused, WillingReport-C
SAC-2	I-Supervised, I-Impartiality
When missing values are replaced by the most prevalent values	
DK-1	Tolerance-C-Bus, Tolerance-C-Gov, I-Confidence, I-Deterrence, C-City, I-Effectiveness, I-Impartiality, I-PowerAbused, C-Bus, C-Gov, C-NextY, I-Powers
DK-2	Tolerance-C-Bus, Tolerance-C-Gov, C-City, I-Deterrence, I-Confidence, I-Effectiveness, I-PowerAbused, I-Impartiality, C-Bus, C-Gov, C-NextY
OP-1	I-Supervised, Sex
OP-2	Saw-I-Publicity, C-NextY, C-PastY, C-Gov, C-Bus, Income, I-Deterrence
SAC-1	Sex, C-NextY, C-PastY, Income, I-Deterrence
SAC-2	LeaveContackInfo, I-Supervised

options were tried with regard to the number of clusters in each partition. Table 17 shows information about the resulting partitions that we deem the most reasonable.

Comparing Table 17 with Table 13, we see that the partitions found by the alternative methods are not as meaningful as those found by BI. Take DK-1 from the top table as an example. This partition is based on demographic variables `age` and `income`, variables that reflect people's confidence in ICAC (e.g., `I-Confidence` and `I-PowerAbused`), variables that reflect people's view on the performance of ICAC (`I-Effectiveness` and `I-Deterrence`), and people's view on the prevalence of corruption in the city (`C-City`). Since it is a mixture of variables, it is not clear what the partition is about. In contrast, all the partitions given by BI on the ICAC data seem meaningful. It seems to the authors that BI has discovered most, if not all, the meaningful ways to cluster the data.

10 Conclusions

Many real-world data sets are multifaceted and can be meaningfully clustered in multiple ways. There is a growing interest in methods that produce multiple partitions of data. Latent tree models (LTMs) are a promising tool for multi-partition clustering. When one analyzes data using LTM, one typically gets a model with multiple latent variables. Each latent variable represents a soft partition of data and hence multiple partitions are obtained. As a model-based approach, the LTM-based method can automatically determine how many partition views there should be, what attributes constitute each partition, and how many clusters there should be for each partition through model selection. The relationships between different partitions can also be inferred from the model.

In this paper, we propose a novel greedy algorithm called BI for learning LTMs. It is significantly more efficient than the state-of-the-art search algorithm EAST. While EAST can only handle data sets with dozens of attributes, BI can deal with data sets with hundreds of attributes. On data sets with dozens of attributes, BI learns models of similar quality to those obtained by EAST.

BI is not as efficient as AC-based algorithms and PTR-motivated algorithms, which can handle thousands of observed variables. However, those algorithms place restrictions on cardinalities of latent and observed variables or connections between variables. On data sets with hundreds of attributes, BI finds better models than the AC-based algorithms and PTR-motivated algorithms.

When used as a tool for multi-partition clustering, BI performs much better than previous methods for the same task. It yields more meaningful and richer clustering results than the alternative methods.

Acknowledgements We thank the anonymous reviewers for their valuable comments. Research on this paper was supported by Hong Kong RGC grants FSGRF12EG63 and FSGRF13EG31, China National Basic Research 973 Program project No. 2011CB505101 and Guangzhou HKUST Fok Ying Tung Research Institute.

References

Anandkumar, A., Chaudhuri, K., Hsu, D., Kakade, S. M., Song, L., & Zhang, T. (2011). Spectral methods for learning multivariate latent tree structure. In *The twenty-fifth conference in neural information processing systems (NIPS-11)*.

Bae, E., & Bailey, J. (2006). Coala: a novel approach for the extraction of an alternate clustering of high quality and high dissimilarity. In *Proceedings of the IEEE international conference on data mining (ICDM 2006)* (pp. 53–62).

Bartholomew, D. J., & Knott, M. (1999). *Latent variable models and factor analysis* (2nd ed.). London: Arnold.

Bickel, S., & Scheffer, T. (2004). Multi-view clustering. In *Proceedings of the IEEE international conference on data mining (ICDM 2004)*.

Chen, T., Zhang, N. L., & Wang, Y. (2008). Efficient model evaluation in the search-based approach to latent structure discovery. In *Proceedings of the 4th European workshop on probabilistic graphical models (PGM 2008)* (pp. 57–64).

Chen, T., Zhang, N. L., Liu, T. F., Poon, K. M., & Wang, Y. (2012). Model-based multidimensional clustering of categorical data. *Artificial Intelligence*, *176*, 2246–2269.

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

Choi, M. J., Tan, V. Y. F., Anandkumar, A., & Willsky, A. S. (2011). Learning latent tree graphical models. *Journal of Machine Learning Research*, *12*, 1771–1812.

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

Cover, T. M., & Thomas, J. A. (2006). *Elements of information theory*. New York: Wiley.

Cui, Y., Fern, X. Z., & Dy, J. G. (2007). Non-redundant multi-view clustering via orthogonalization. In *Proceedings of the IEEE international conference on data mining (ICDM 2007)* (pp. 133–142).

Dasgupta, S., & Ng, V. (2010). Mining clustering dimensions. In *Proceedings of the IEEE international conference on data mining (ICDM 2010)* (pp. 263–270).

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

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

Galimberti, G., & Soffritti, G. (2007). Model-based methods to identify multiple cluster structures in a data set. *Computational Statistics & Data Analysis*, *52*, 520–536.

Gondek, D., & Hofmann, T. (2007). Non-redundant data clustering. *Knowledge and Information Systems*, *12*(1), 1–24.

Guan, Y., Dy, J. G., Niu, D., & Ghahramani, Z. (2010). Variational inference for nonparametric multiple clustering. In *KDD10 workshop on discovering, summarizing and using multiple clusterings*.

Harmeling, S., & Williams, C. K. I. (2011). Greedy learning of binary latent trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *33*(6), 1087–1097.

Jain, P., Meka, R., & Dhillon, I. S. (2008). Simultaneous unsupervised learning of disparate clusterings. *Statistical Analysis and Data Mining*, *1*(3), 195–210.

Kass, R. E., & Raftery, A. E. (1995). Bayes factors. *Journal of the American Statistical Association*, *90*(430), 773–795.

Kriegel, H. P., Kröger, P., & Zimek, A. (2009). Clustering high-dimensional data: a survey on subspace clustering, pattern-based clustering, and correlation clustering. *ACM Transactions on Knowledge Discovery from Data*, *3*, 1–58.

Liu, T. F., Zhang, N. L., Liu, A. H., & Poon, L. K. M. (2012). A novel LTM-based method for multidimensional clustering. In *Proceedings of the sixth European workshop on probabilistic graphical models (PGM-2012)*.

MacQueen, J. (1967). Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, California, USA (Vol. 1, p. 14).

Mossel, E., Roch, S., & Sly, A. (2011). Robust estimation of latent tree graphical models: inferring hidden states with inexact parameters. In *Proceedings of CoRR*.

Mourad, R., Sinoquet, C., & Leray, P. (2011). A hierarchical Bayesian network approach for linkage disequilibrium modeling and data- dimensionality reduction prior to genome-wide association studies. *BMC Bioinformatics*, *12*, 16.

Mourad, R., Sinoquet, C., Zhang, N. L., Liu, T., & Leray, P. (2013). A survey on latent tree models and applications. *Journal of Artificial Intelligence Research*, *47*, 157–203.

Niu, D., Dy, J. G., & Jordan, M. I. (2010). Multiple non-redundant spectral clustering views. In *Proceedings of the 27th international conference on machine learning (ICML2010)*.

Parsons, L., Haque, E., & Liu, H. (2004). Subspace clustering for high dimensional data: a review. *ACM SIGKDD Explorations Newsletter*, *6*(1), 90–105.

Pearl, J. (1988). *Probabilistic reasoning in intelligent systems: networks of plausible inference*. San Mateo: Morgan Kaufmann.

Poon, K. M., Zhang, N. L., Chen, T., & Wang, Y. (2010). Variable selection in model-based clustering: to do or to facilitate. In *Proceedings of the 27th international conference on machine learning (ICML2010)*.

Qi, Z., & Davidson, I. (2009). A principled and flexible framework for finding alternative clusterings. In *Proceedings of the 15th ACM SIGKDD international conference on knowledge discovery and data mining (KDD2009)* (pp. 717–726).

Ranwez, V., & Gascuel, O. (2001). Quartet-based phylogenetic inference: improvements and limits. *Molecular Biology and Evolution*, *18*(6), 1103–1116.

Robinson, D. F., & Foulds, L. R. (1981). Comparison of phylogenetic trees. *Mathematical Biosciences*, *53*, 131–147.

Saitou, N., & Nei, M. (1987). The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Molecular Biology and Evolution*, *4*, 406–425.

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

Song, L., Parikh, A., & Xing, E. (2011). Kernel embeddings of latent tree graphical models. In *Twenty-fifth conference in neural information processing systems (NIPS-11)*.

Strehl, A., Ghosh, J., & Cardie, C. (2002). Cluster ensembles—a knowledge reuse framework for combining multiple partitions. *Journal of Machine Learning Research*, *3*, 583–617.

Wang, Y., Zhang, N. L., & Chen, T. (2008). Latent tree models and approximate inference in Bayesian networks. *Journal of Artificial Intelligence Research*, *32*, 879–900.

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

Zhang, N. L., & Kocka, T. (2004). In *Proceedings of the 16th IEEE international conference on tools with artificial intelligence (ICTAI 2004)* (pp. 585–593).

Zhang, N. L., Wang, Y., & Chen, T. (2008a). Discovery of latent structures: experience with the COIL challenge 2000 data set. *Journal of Systems Science and Complexity*, *21*, 172–183.

Zhang, N. L., Yuan, S. H., Chen, T., & Wang, Y. (2008b). Latent tree models and diagnosis in traditional Chinese medicine. *Artificial Intelligence in Medicine*, *42*, 229–245.