Computational Aspects of Bayesian Partition Models

Mikko Koivisto MIKKO.KOIVISTO@CS.HELSINKI.FI
HIIT Basic Research Unit, Dept. of Computer Science, University of Helsinki, FIN-00014 Helsinki, Finland

Kismat Sood KISMAT.SOOD@KTL.FI
National Public Health Institute, Dept. of Molecular Medicine, FIN-00251 Helsinki, Finland

Abstract

The conditional distribution of a discrete variable y, given another discrete variable x, is often specified by assigning one multinomial distribution to each state of x. The cost of this rich parametrization is the loss of statistical power in cases where the data actually fits a model with much fewer parameters. In this paper, we consider a model that partitions the state space of x into disjoint sets, and assigns a single Dirichlet-multinomial to each set. We treat the partition as an unknown variable which is to be integrated away when the interest is in a coarser level task, e.g., variable selection or classification. Based on two different computational approaches, we present two exact algorithms for integration over partitions. Respective complexity bounds are derived in terms of detailed problem characteristics, including the size of the data and the size of the state space of x. Experiments on synthetic data demonstrate the applicability of the algorithms.

1. Introduction

Conditional probability distributions are central in both predictive (classification and regression) and inferential (discovery of association and causality) applications. For example, Bayesian networks represent a multivariate probability distribution as a collection of univariate conditional distributions (e.g., Pearl, 1988; Heckerman et al., 1995).

In the case of discrete variables, the multinomial distribution gives the most general parametrization. If y is a discrete variable with r states, then $r-1$ "free" parameters are sufficient to specify the distribution of y. When the distribution of y is considered conditionally on another discrete variable x (possibly a vector $(x_1, \ldots, x_s)$ representing s parents of y) with q states, then the most general parametrization uses altogether $(r-1)q$ free parameters. The meaning of "free" depends on the chosen statistical paradigm. In the Bayesian framework—which we adopt in this paper—it corresponds to a local independence assumption and hence often to a Dirichlet prior on the parameters (Geiger & Heckerman, 2002).

It is generally acknowledged that the Dirichlet-multinomial model is often too flexible. This is especially the case when the number of parameters is large but the size of the data is small. If stronger assumptions concerning the relationship between the multinomial parameters can be made, then learning power can be enhanced. In the context of Bayesian networks, local models have been studied. Friedman and Goldszmidt (1996) present a non-Bayesian treatment of decision trees, whereas Chickering et al. (1997) offer a Bayesian approach using decision graphs. A shortcoming of both these methods is that they search for a single best local structure only. Rather, what we would like to do is to sum over possible structures in a manner of model averaging. To approximate such averages (under a slightly different model), Golinelli et al. (1999) propose a Markov chain Monte Carlo method. These three methods have the drawback that they provide no quality guarantee for the output. Thus, an unwanted layer of uncontrolled uncertainty is introduced by these methods.

In this paper we ask: When can local models be handled *exactly* and with sufficient efficiency? To answer this we restrict our attention to *partition models*. A partition model groups the states of the conditioning variable, x, into a relatively small number of groups called *levels*. Each level specifies a Dirichlet-

Appearing in *Proceedings of the 22nd International Conference on Machine Learning*, Bonn, Germany, 2005. Copyright 2005 by the author(s)/owner(s).

multinomial model that is shared by the states that belong to that group. Partition models are thus more general than decision *trees*, which build a tree based on a fixed representation of x as a vector $(x_1, \ldots, x_s)$ of several variables. However, the expression power of partition models and decision *graphs* are equal: both can represent arbitrary groupings of the states of x. Here we prefer the partition representation over decision graphs, for the partition representation makes explicit certain model characteristics that are interesting from the computational point of view. We note that our partition model is simpler than the hierarchical variant proposed by Golinelli et al. (1999), in which the states within a group have a similar, not necessarily identical, model.

We stress that this paper concerns *learning* with structured models of conditional distributions (partition models in particular). Thus, this work is complementary to those that consider *inference* in Bayesian networks under a given structured model (Poole, 1997). The difference between these two tasks is not only in the goal but also in the algorithmic techniques used.

As an example of a partition model, consider a genetic penetrance model. Suppose there are two genes, each with two variants: one "predisposing" allele and one "normal" allele. A genotype G over two genes consists of two pairs of alleles; one allele is inherited from the mother and the other from the father. For a binary trait, the penetrance of G is the probability that a person who carries G has the trait. For two genes the penetrance function can be represented as a matrix, where the element of the ith row and the jth column is the penetrance of the genotype with $i-1$ copies of the predisposing allele of the first gene and $j-1$ copies of the predisposing allele of the second gene. For example, the matrix

$$F = \begin{pmatrix} 0.1 & 0.1 & 0.3 \\ 0.1 & 0.3 & 0.3 \\ 0.3 & 0.3 & 0.9 \end{pmatrix} \tag{1}$$

involves three levels of penetrances, each corresponding to a subset of genotypes. Grouping genotypes may be justified, as different genotypes can be responsible for the same intermediate products (e.g., proteins).

The rest of this paper is organized as follows. In Section 2, we recall the role of the multinomial model in prediction and inference. The idea of partitioning is formulated in Section 3. In Section 4, we present two exact algorithms for averaging over possible partitions; some modifications are discussed in Section 5. The applicability of the presented algorithms is demonstrated by experiments on synthetic data in Section 6. Section 7 presents concluding remarks.

2. Preliminaries

We consider a *data* sequence $z[1], \ldots, z[m]$, where each $z[t]$ is a pair $(x[t], y[t])$ of two variables; typically each $x[t]$ is vector-valued. For convenience, suppose that these variables take small integer values, $x[t]$ in $\mathcal{X} = \{1, \ldots, q\}$ and $y[t]$ in $\mathcal{Y} = \{1, \ldots, r\}$. We briefly denote $\mathbf{x}$ for the sequence $(x[1], \ldots, x[m])$ and similarly $\mathbf{y}$ for $(y[1], \ldots, y[m])$. We are interested in specifying the conditional distribution $p(\mathbf{y} \mid \mathbf{x})$, because this type of conditional distribution plays a central role in various applications, e.g., in learning Bayesian networks (see, e.g., Cooper & Herskovits, 1992; Heckerman et al., 1995; Chickering et al., 1997).

To incorporate the idea of learning from data, the distribution of $\mathbf{y}$ given $\mathbf{x}$ is supposed to be a mixture of multinomial distributions:

$$p(\mathbf{y} \mid \mathbf{x}) = \int p(\theta) \prod_{j=1}^{q} \prod_{k=1}^{r} \theta_{jk}^{N_{jk}} \mathrm{d}\theta \,. \tag{2}$$

Here N_{jk} is the number of ts for which $y[t] = k$ and $x[t] = j$, and θ_{jk} are nonnegative real numbers, called *multinomial parameters*, satisfying $\sum_{k=1}^{r} \theta_{jk} = 1$ for all j. The density function $p(\theta)$ is called a *prior*.

A conditional model can be used, for instance, in prediction. The prediction, or classification, problem is to guess $y[t]$ given $x[t]$ along with the past observations $z[1], \ldots, z[t-1]$. A Bayesian approach to prediction relies on the conditional $p(y[t] \mid x[t], z[1], \ldots, z[t-1])$, which is proportional to $p(\mathbf{y} \mid \mathbf{x})$.

One may also be interested in learning the underlying structure of the data generating process. For example, in variable selection the goal is to select from a set of n variables $\{x_1, \ldots, x_n\}$ the subset $\{x_{i_1}, \ldots, x_{i_s}\}$ that is responsible for the variation in a variable y, while the rest of the variables are regarded as irrelevant "noise" with respect to y. When approaching this task, it is convenient to specify a conditional distribution of the form $p(\mathbf{y} \mid \mathbf{x}_{i_1}, \ldots, \mathbf{x}_{i_s})$ (assuming $\mathbf{y}$ is conditionally independent of the remaining $n-s$ variables). Identifying the *parents* of a variable in this way is a key task in structure learning in Bayesian networks.

3. Partition Models

The multinomial model does not force us to use $q(r-1)$ "free" parameters. Instead, the prior allows for representing arbitrary soft and hard dependences among the parameters. Here we introduce a simple class of priors that facilitate feasible computations.

Instead of q parameters, we allow for $\ell \leq q$ parameters. This is to say that some states of x share a multinomial parameter vector. More precisely, each state j in $\mathcal{X} = \{1, \ldots, q\}$ is mapped to a *level* $L(j) \in \{1, \ldots, \ell\}$. For different levels $h = 1, \ldots, \ell$ we assign a multinomial parameter vector λ_h. The mapping L, called a *partition*, and the parameters $\lambda_1, \ldots, \lambda_\ell$ are treated as independent random variables (vectors).

To make model specification and computations convenient, we suppose that the prior of L has a modular structure.

Definition 3.1 *A prior on partitions is* state-wise modular *if there exist numbers $\rho_{jh} \geq 0$, with $\sum_{h=1}^{\ell} \rho_{jh} = 1$ for all j, such that the prior probability of any partition L factorizes as*

$$p(L) = \rho_{1L(1)} \cdots \rho_{qL(q)} \,.$$

For example, an "uninformative" state-wise modular prior is given by setting $\rho_{jh} = 1/\ell$ for all j (the uniform prior). Henceforth we suppose that the prior on L is state-wise modular. Further, for each parameter λ_h we assign a Dirichlet prior,

$$\lambda_h \sim \text{Dirichlet}(\alpha_{h1}, \ldots, \alpha_{hk}) \,.$$

Together, the above assignments imply a prior $p(\theta)$. Namely, the parameters θ can be viewed as the result of first drawing the parameters $\lambda_1, \ldots, \lambda_\ell$ and then drawing (independently) a mapping L which sets each θ_j to $\lambda_{L(j)}$.

Substitution into (2) gives

$$p(\mathbf{y} \,|\, \mathbf{x}) = \int \sum_{L} p(L)\, p(\lambda) \prod_{h=1}^{\ell} \prod_{k=1}^{r} \lambda_{hk}^{M_{hk}} \mathrm{d}\lambda \,, \tag{3}$$

where $M_{hk} = \sum_{j \in L^{-1}(h)} N_{jk}$ depends on L (and on the data). Reversing the order of integration and summation yields an alternative mixture representation,

$$p(\mathbf{y} \,|\, \mathbf{x}) = \sum_{L} p(L) f(L) \,, \tag{4}$$

where L runs through all partitions, and for $f(L)$ we have a closed-form expression involving the gamma function Γ,

$$f(L) = \prod_{h=1}^{\ell} \frac{\Gamma(\alpha_h)}{\Gamma(\alpha_h + M_h)} \prod_{k=1}^{r} \frac{\Gamma(\alpha_{hk} + M_{hk})}{\Gamma(\alpha_{hk})} \,. \tag{5}$$

Here $\alpha_h = \sum_k \alpha_{hk}$ and $M_h = \sum_k M_{hk}$. Note the implicit dependence on L (and on the data).

It is worth noting that a partition L not only partitions the state space $\mathcal{X} = \{1, \ldots, q\}$ into disjoint groups, but also assigns a distinct level to each group. Moreover, the number of (nonempty) groups varies between 1 and ℓ depending on the partition L.

Straightforward integration over partitions is computationally demanding. Namely, the number of partitions, ℓ^q, grows rapidly with q. A similar computational problem is faced by Chickering et al. (1997) when they consider maximization over decision graphs. They suggest a greedy algorithm which, in general, is suboptimal. In the next section, we present two different *exact* algorithms for computing the sum (4) and discuss how the applicability of these methods depends on the problem parameters ℓ, m, q, and r.

4. Two Exact Approaches

We give two algorithms for summing over partitions. Both algorithms apply dynamic programming and can be viewed as instances of the general variable elimination algorithm (see, e.g., Stearns & Hunt III, 1996; Dechter, 1999). The *state-wise algorithm* sums over each $L(j)$ (the level of state j) in turn. Its dual, the *level-wise algorithm*, sums over each $L^{-1}(h)$ (the states of level h) in turn.

4.1. The State-Wise Algorithm

From now on, treat N as a matrix of size $q \times r$ and M as a matrix of size $\ell \times r$. Let $N_{j.}$ and $M_{h.}$ denote the jth and the hth row of the matrices N and M, respectively.

The state-wise algorithm is based on the following iterative representation of the sum (4).

Lemma 4.1 *Define*

$$g_0(M) = \prod_{h=1}^{\ell} \frac{\Gamma(\alpha_h)}{\Gamma(\alpha_h + M_h)} \prod_{k=1}^{r} \frac{\Gamma(\alpha_{hk} + M_{hk})}{\Gamma(\alpha_{hk})} \,,$$

and iteratively for $j = 1, \ldots, q$,

$$g_j(M) = \sum_{L(j)=1}^{\ell} \rho_{jL(j)}\, g_{j-1}(M') \,,$$

where M' is obtained from M by replacing the $L(j)$th row $M_{L(j).}$ by $M_{L(j).} + N_{j.}$. Then $p(\mathbf{y} \,|\, \mathbf{x}) = g_q(0)$.

Proof: It is easy to see that $g_q(0) = \sum_L p(L)\, g_0(M)$, where each row $M_{h.}$ of the matrix M satisfies $M_{h.} = \sum_{j \in L^{-1}(h)} N_{j.}$. Thus by (4) and (5) we have $p(\mathbf{y} \,|\, \mathbf{x}) = g_q(0)$ as claimed. □

This representation suggests a dynamic programming algorithm for integration over partitions. First we compute $g_0(M)$ for all relevant matrices M; a matrix M is *relevant* if there exists a partition that together with the observed matrix N yields M. Then we continue by computing the function g_1, and so forth. Finally the desired value can be read as $g_q(0)$.

An alternative implementation, memoization, proceeds recursively. First it calls $g_q(0)$ which in turn calls $g_{q-1}(M)$ for a number of different matrices M, etc., until finally the recursion stops at calling $g_0(M)$ for some relevant matrices M. In this way only the relevant matrices are constructed along the calls.

To analyze the computational complexity of these methods we bound the number of relevant matrices. Since there are m data records, an obvious upper bound is $m^{\ell r}$. However, this bound can be improved fairly easily. Namely, we note that the column sums of the matrices M and N must be equal. In the worst case all columns sums are equal to m/r. Hence, we have at most $(m/r)^{(\ell-1)r}$ relevant matrices. From this we obtain the total time complexity of $O(\min\{m,q\}(m/r)^{(\ell-1)r})$, since only those states of x need to be considered which occur in the data. We conclude that the state-wise algorithm is practical when the numbers r and ℓ are small. In particular, for $r=\ell=2$ the algorithm runs in time quadratic in m. Also, we note that the algorithm is linear in q. The space complexity of the algorithm is $O((m/r)^{(\ell-1)r})$.

4.2. The Level-Wise Algorithm

For a large maximum number of levels, ℓ, the state-wise algorithm is impractical. To tackle this case, we introduce an alternative approach, the level-wise algorithm. The running time of this algorithm is not sensitive to ℓ but it grows exponentially in q.

Here we may slightly relax the assumption of state-wise modular priors.

Definition 4.1 *A prior on partitions is* level-wise modular *if there are numbers $\psi_{Sh} \geq 0$ such that for any partition L,*

$$p(L) = \psi_{L^{-1}(1)1} \cdots \psi_{L^{-1}(\ell)\ell}. \qquad (6)$$

Proposition 4.2 *If a prior on partitions is state-wise modular, then it is also level-wise modular.*

Proof: We assume the notation above. For each level h and each subset S of $\mathcal{X} = \{1,\ldots,q\}$ let $\psi_{Sh} = 1$ if S is empty and set $\psi_{Sh} = \prod_{j\in S} \rho_{jh}$ otherwise. Clearly condition (6) is met. □

The following result gives a useful expression for the conditional probability $p(\mathbf{y}\,|\,\mathbf{x})$.

Lemma 4.3 *For each level $h = 1,\ldots,\ell$ and every subset S of $\mathcal{X} = \{1,\ldots,q\}$ define*

$$w_h(S) \;=\; \psi_{Sh} \frac{\Gamma(\alpha_h)}{\Gamma(\alpha_h + M_h)} \prod_{k=1}^{r} \frac{\Gamma(\alpha_{hk} + M_{hk})}{\Gamma(\alpha_{hk})},$$

where $M_{hk} = \sum_{j\in S} N_{jk}$. Then

$$p(\mathbf{y}\,|\,\mathbf{x}) = \sum_{L} \prod_{h=1}^{\ell} w_h(L^{-1}(h)).$$

Proof: Obvious from the definitions. □

By a level-wise modular prior we can express that a set T cannot appear as the set $L^{-1}(h)$ for some level h, by simply letting $\psi_{Th} = 0$. When this kind of restrictions obey a suitable structure they can be exploited to obtain faster algorithms. We now introduce some notation which facilitates such considerations, however, related discussions are postponed till Section 5. Let $\mathcal{S}_h$ denote the collection of sets that can be obtained as unions of $L^{-1}(1),\ldots,L^{-1}(h)$. That is,

$$\mathcal{S}_h = \Big\{ \bigcup_{h'=1}^{h} L^{-1}(h') : p(L) > 0 \Big\}.$$

We also set $\mathcal{S}_0 = \{\emptyset\}$ for convenience.

The level-wise algorithm is based on the following representation of the sum over partitions.

Lemma 4.4 *For $h = \ell,\ldots,1$ define recursively*

$$u_h(S) = \sum_{T\in\mathcal{S}_{h-1}:T\subseteq S} w_h(S-T)\, u_{h-1}(T),$$

with the boundary $u_0(\emptyset) = 1$. Then $p(\mathbf{y}\,|\,\mathbf{x}) = u_\ell(\mathcal{X})$.

Proof: Immediate from Lemma 4.3. □

The straightforward dynamic programming algorithm is as follows. First compute $u_1(S)$ for every subset S of $\mathcal{X}$, then compute $u_2(S)$ for every subset S of $\mathcal{X}$, and so on, until finally $u_\ell(S)$ is computed—but in this case for $S = \mathcal{X}$ only.

To bound the computational complexity we consider the worst case when $\mathcal{S}_h$ equals the power set $2^{\mathcal{X}} = \{S : S \subseteq \mathcal{X}\}$ for all levels h. We note that the values $w_h(S)$ can be computed in time $O(r\,m + r\,\ell 2^q)$ (assuming that the values of the gamma function can be evaluated in constant time). Then we observe that each function u_h, for $h = 1,\ldots,\ell-1$, can be computed in time $O(3^q)$. Thus, the total time complexity is $O(r\,m + r\,\ell 2^q + \ell 3^q)$. This simplifies to $O(\ell 3^q)$ for all parameter values of our interests. The space requirement is $O(2^q)$.

The applicability of the level-wise algorithm depends essentially on the state space size q. Roughly, computations are feasible when q is less than 20. This is a crucial limitation since the partitioning method, in general, is best suited for cases where the state space is so large that the plain Dirichlet-multinomial model suffers from low power, that is, when q is large. However, as we show next, partitioning much larger state spaces is tractable when suitable restrictions on the possible partitions are made (via the collections $\mathcal{S}_h$).

5. Monotone Partitions

In many cases the variable x is actually a vector $(x_1, \ldots, x_s)$ of length s, where each component x_i takes values in a finite set, say in $\mathcal{X}_i = \{1, \ldots, q_i\}$. Thus we have $q = q_1 \cdots q_s$. To incorporate the idea of monotonicity, we assume that each set $\mathcal{X}_i$ is equipped with a linear order $<$; here we assume the natural order of integers. Building on these orders we let $\prec$ denote the ordinary partial order of vectors in $\mathcal{X} = \mathcal{X}_1 \times \cdots \times \mathcal{X}_s$. That is, $(j_1, \ldots, j_s) \prec (j'_1, \ldots, j'_s)$ if $j_i \le j'_i$ for all i and $j_i < j'_i$ for at least one i.

Definition 5.1 *A partition* $L\colon \mathcal{X} \to \{1, \ldots, \ell\}$, *is* monotone *if* $L(j) \le L(j')$ *whenever* $j \prec j'$.

We note that many popular models for the conditional distribution of y given x induce a monotone partition. For example, an additive model $y = g(x_1 + \cdots + x_s + \varepsilon)$—where y is a binary variable, ε is an independent noise variable, and g is a threshold function—induces a monotone partition. Similarly a multiplicative model implies a partition that is monotone. A concrete example of a monotone partition is given by the penetrance model (1) introduced in Section 1.

To allow monotone partitions only, we slightly modify the structure of the prior of partitions.

Definition 5.2 *A prior on partitions is* monotone level-wise modular *if there are numbers* $\psi_{Sh} \ge 0$ *such that for any partition* L,

$$p(L) = \begin{cases} \psi_{L^{-1}(1)1} \cdots \psi_{L^{-1}(h)h} & \text{if } L \text{ is monotone;} \\ 0 & \text{otherwise.} \end{cases}$$

We next proceed to algorithmic issues on computing $p(\mathbf{y} \mid \mathbf{x})$ allowing monotone partitions only. We observe that the level-wise algorithm applies. Furthermore, each $\mathcal{S}_h$ is the family of $\succ$-*closed* subsets of $\mathcal{X}$, denoted by $\mathcal{S}^{\succ}$: A set S is $\succ$-closed if $j \in S$ and $j \prec j'$ together imply that also j' belongs to S.

To bound the complexity of the level-wise algorithm (under the monotonicity restriction), we estimate the number $|\mathcal{S}^{\succ}|$ of $\succ$-closed subsets of $\mathcal{X}$. Given this number, a time complexity bound for computing $p(\mathbf{y} \mid \mathbf{x})$ is simply $O(\ell |\mathcal{S}^{\succ}|^2)$. The following result covers the cases where $\mathcal{X}$ is one- or two-dimensional.

Proposition 5.1 *Let* $\mathcal{X} = \mathcal{X}_1 \times \cdots \times \mathcal{X}_s$ *with* $|\mathcal{X}_i| = q_i$. *(a) If* $s = 1$, *then* $|\mathcal{S}^{\succ}| = q_1 + 1$. *(b) If* $s = 2$, *then* $|\mathcal{S}^{\succ}| = \binom{q_1+q_2}{q_1}$.

Proof: (a) Note that any $\succ$-closed family is either empty or of the form $\{q_1, q_1 - 1, \ldots, q_1 - c + 1\}$ for some $c \in \{1, \ldots, q_1\}$.

(b) Let S be a $\succ$-closed subset of $\mathcal{X}$. For $i = 1, \ldots, q_2$ let c_i denote the number of vectors of the form (j_1, i) in S. The number of different sets S in $\mathcal{S}^{\succ}$ is given $\sum_{c_1=0}^{q_1} \sum_{c_2=0}^{c_1} \times \cdots \times \sum_{c_{q_2}=0}^{c_{q_2-1}} 1$, which we recognize to be equal to the binomial coefficient $\binom{q_1+q_2}{q_1}$. □

In the general case, $s > 2$, we do not know any closed-form expression for $|\mathcal{S}^{\succ}|$. However, when $q_i = 2$ for all i, then $|\mathcal{S}^{\succ}|$ equals the number of nonempty antichains of subsets of a set of size s. The following bound is due to Kleitman and Markowsky in 1975 (see Kahn 2002, Corollary 1.4); here all logarithms are base 2.

Proposition 5.2 *Let* $\mathcal{X}$ *be a Cartesian product* $\{1, 2\}^s$. *Then* $\log |\mathcal{S}^{\succ}| \le (1 + 2 \log(s + 1)/s) \binom{s}{\lfloor s/2 \rfloor}$.

This bound is not very encouraging, though, showing only that computations are feasible up to around $s = 5$. For larger numbers of states, $q_i > 2$, we can expect to gain more from the monotonicity constraint; we leave this issue for future research.

We notice that the monotonicity constraint reduces the number of possible sets from 2^q to a number that can be computationally feasible. This is especially the case when s is small. For instance, let $s = 2$ and $q_1 = q_2 = 5$. Then, by Proposition 5.1, there are only $\binom{5+5}{5} = 252$ distinct $\succ$-closed sets, which is much less than the number of all subsets, $2^{25} = 33{,}554{,}432$.

6. Experimental Results

We have implemented the state-wise and the (non-monotone) level-wise algorithm described in this paper. Our implementation is written in the C++ programming language. The experiments to be described next were run on an ordinary desktop PC with a 2.4 GHz Pentium processor and 1.0 GB of memory.

The main objective of these experiments is to gauge the speed of the algorithms for different problem parameters. In addition, we illustrate how integration over partitions finds its use in a variable selection task.

For lack of space we omit experiments on an important application: learning Bayesian networks. However, conclusions about when the presented exact algorithms are practical can be drawn (Section 6.3).

6.1. Running Time Analysis

For speed measurements, we generated a data set of 200 records each with 8 binary variables. We used the uniform distribution independently for each record and variable. The first variable was treated as the variable y. The next $\log q$ variables form the variable (vector) x with q states. When experimenting with the state-wise algorithm, we let q take values in $\{8, 32, 128\}$; for the level-wise algorithm q was set to 8 or 16. Besides the data set of 200 records, a smaller data set comprising the first 50 records was also used.

Table 1(a) displays results for the state-wise algorithm. The number of levels, ℓ, was set to 2, 3, or 4; results are not shown for the cases that ran out of memory. The results are in good agreement with our analytic bound. Clearly, the computations are feasible for $\ell = 2$ but they become impractical when $\ell = 4$, unless q is very small. Also the data size, m, plays an important role in the time requirement. However, increasing m from 50 to 200 is less drastic than what is suggested by the complexity bound. We also see that the influence of q on the time requirement is not always linear. For example, when $m = 50$ and $\ell = 3$, increasing q from 8 to 32 does not lead to a 4-fold increase in time requirement, but to a 350-fold increase. This can be explained by noticing that for q much smaller than m, the number of different matrices explored by the algorithm is much less than the rough analytic estimate.

Results for the level-wise algorithm are shown in Table 1(b). The number of levels, ℓ, was set to 4, 8 or 16. The result are in perfect agreement with the derived bound: the state space size q plays a crucial role, and the effect of the number of levels ℓ is about linear. As the time complexity does not depend on the data size, Table 1(b) shows the results for the case $m = 200$ only.

6.2. Variable Selection

The variable selection task was performed on two simple data sets. One of these can be efficiently analyzed using the state-wise algorithm, whereas the level-wise algorithm is faster on the other data set.

The first data set, called Parity5, consists of 200 records, each with a binary class variable y and 20 binary explanatory variables $x_1, \ldots, x_{20}$. We generated each record independently. First, the values for the explanatory variables were drawn from the uniform distribution. Then the value for the class variable was set to the (odd) parity of the first 5 variables, $x_1 + x_2 + \cdots + x_5 \pmod 2$, with probability 0.9 and to its reverse with probability 0.1. Note that the remaining 15 explanatory variables do not carry information about y. Also note that the data generating model corresponds to a partition with two groups.

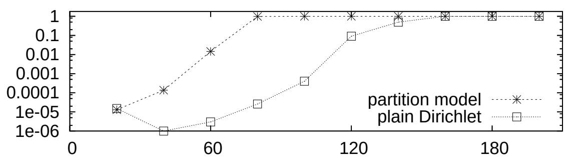
The second data set, called Penetrance2, consists of 400 records, each with a binary class variable y and 10 explanatory variables $x_1, \ldots, x_{10}$ with four states 0, 1, 2, 3. We generated each record independently. First, the values for the explanatory variables were drawn as follows: We drew a binary sequence $x'_1, \ldots, x'_{10}$ along a Markov chain with uniform initial distribution and set $x'_{i+1} = x'_i$ with probability 0.8. Similarly we drew another sequence $x''_1, \ldots, x''_{10}$. Finally, we set $x_i = x'_i + 2x''_i$. The value of the class variable was generated based on the values of the variables x_3 and x_8 as follows: Let i and j be the number of ones in the binary representations of x_3 and x_8, respectively. We set $y = 1$ with the probability $F_{i+1,j+1}$, where F is the 3×3 penetrance matrix (1) introduced in Section 1.

From both Parity5 and Penetrance2 we made 10 data sets of different sizes by including the first 10%, 20%, . . . , 100% records of the original data set.

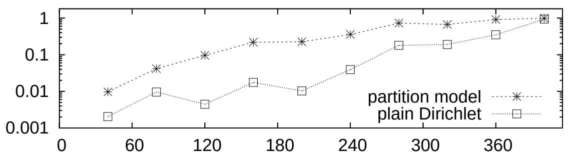
Table 1. The speed of (a) the state-wise algorithm and (b) the state-wise algorithm as a function of the number of data records (m), the number of states of the explanatory variable (q), and the number of levels in the partition (ℓ).

	m	q	ℓ	Time (Sec.)
(a)	50	8	2	0.0008
			3	0.02
			4	0.24
		32	2	0.03
			3	7.0
		128	2	0.13
			3	29.5
	200	8	2	0.001
			3	0.03
			4	0.38
		32	2	0.25
		128	2	1.8

	m	q	ℓ	Time (Sec.)
(b)	200	8	4	0.007
			8	0.019
		16	4	45
			8	134
			16	314



(a)



(b)

Figure 1. Learning the correct variable subset for (a) Parity5 and (b) Penetrance2 under a partition model and a plain Dirichlet model. The posterior probability of the correct subset (vertical axis) shown as a function of the data size (horizontal axis).

Given a data set consisting of sequences $\mathbf{y}$ and $\mathbf{x}$, we selected a set of variables that maximizes the posterior probability. First, to a subset $S = \{i_1, \ldots, i_s\}$ of $\{1, \ldots, n\}$ (n equals 20 for Parity5 and 10 for Penetrance2) we assigned a constrained uniform prior: $p(S) = \binom{n}{s}^{-1}/(u+1)$ if $s \le u$, and $p(S) = 0$ otherwise ($u = 5$ for Parity5 and $u = 2$ for Penetrance2). Then we selected a subset S of $\{1, \ldots, n\}$ so as to maximize $p(S)p(\mathbf{y} \,|\, \mathbf{x}, S)$. Here $p(\mathbf{y} \,|\, \mathbf{x}, S) = p(\mathbf{y} \,|\, \mathbf{x}_{i_1}, \ldots, \mathbf{x}_{i_s})$, computed either by the state-wise algorithm (for Parity5) or by the level-wise algorithm (for Penetrance2). Both algorithms were run under the uniform state-wise modular prior, with the optimal number of levels, $\ell = 2$ for Parity5 and $\ell = 3$ for Penetrance2. In the Dirichlet prior all hyperparameters were set to 1.

Figure 1 shows how the posterior probability of the correct subset grows as a function of the data size. For comparison, results under the plain Dirichlet prior, with all hyperparameters equal to 1, are also given.

For Parity5 the partition model is superior to the plain Dirichlet model. As indicated by the rapid change in the posterior probabilities, the former learns the correct variable set from 80 data records, whereas the latter model requires 140 examples. For Penetrance2 the partition model requires 240 records, whereas the plain Dirichlet model learns the correct subset only when given all the 400 examples. Note that for Penetrance2 we used a non-monotone partition model, although the underlying partition is monotone.

6.3. On Learning Bayesian Networks

The above results help in estimating when it is computationally feasible to use exact partitioning models in learning Bayesian networks. Consider the case where each variable can have at most s parents from the $n-1$ possible candidates, where n is the total number of variables. For $n = 20$ and $s = 4$, for example, 101,120 evaluations are needed.[1] With 1, 10, and 100 hours of running time, we can use, respectively, 0.036, 0.36, and 3.6 seconds per evaluation. From Table 1 we conclude that for many values of the parameters m, q, and ℓ already 1 hour of total running time suffices. (For monotone partitions we can expect that computations are efficient for a larger region of parameter values.)

To learn Bayesian network structures on larger numbers of variables one may combine exact integration over partitions with a heuristic search for the network structure. Chickering et al. (1997) demonstrate that greedy search using decision graphs (equivalent to partition models) as the local models, can result in significantly more accurate network models than using the plain Dirichlet model or decision trees. They consider, e.g., data generated from the Alarm model (Beinlich et al., 1989) which contains 37 categorial variables, each with 4 or less states. We note that in this benchmark domain exact integration over partitions offers a sound alternative to heuristic maximization when the interest is in the network structure. Our running time analysis suggests that the state-wise algorithm runs sufficiently fast, provided that one allows for just two levels; for three or more levels, incorporation of additional constraints seems necessary.

7. Discussion

We have considered computational issues concerning Bayesian partition models of discrete conditional distributions. From the statistical point of view, the advantage of partition models over the commonly used Dirichlet-multinomial model is the smaller number of parameters. We do, of course, acknowledge that no model is *generally* better than any other model, and thus have focused on how well we can perform Bayesian learning, that is, utilize prior information. Regarding partition models, this paper contributes in three ways. First, it presents algorithms for *integration* over unknown partitions, which is required in coarse level Bayesian learning tasks. Second, these algorithms are *exact*, meaning that there is no uncertainty about the quality of the output; as far as we know, existing related methods are restricted to optimization with uncontrolled quality (Friedman & Goldszmidt, 1996; Chickering et al., 1997). Third, we investigated how the computational complexity of the algorithms

[1]Note that *exact* structure discovery is computationally feasible up to around 25 variables (Koivisto & Sood, 2004).

depends on several problem parameters, and found *feasible parameter regions*. Note also that the presented algorithms can be easily modified so that they find an optimal (maximum a posteriori) partition instead. Namely, the algorithms essentially exploit the distributive law on the sum-product semiring—conversion to the max-product semiring is straightforward (see, e.g., Stearns & Hunt III, 1996; Dechter, 1999).

Two exact computational approaches were presented: the state-wise and the level-wise algorithm. The analysis of these algorithms reveals different complexity characteristics. Roughly, if the number of levels, ℓ, is small, say at most 3, then the state-wise algorithm is efficient, irrespective of the state space size $q = |\mathcal{X}|$. If ℓ is larger, then one can use the level-wise algorithm, provided that q is not too large, say $q < 20$. Furthermore, for monotone partitions exact computations are feasible when the state space $\mathcal{X}$ is a Cartesian product of a few totally ordered sets. These characterizations facilitate decision making about which algorithm to use in a particular problem setting, or whether exact computations are intractable.

We also measured the speed of the algorithms and illustrated their use in variable selection on two synthetic data sets. Our results agree with earlier observations (Friedman & Goldszmidt, 1996; Chickering et al., 1997) in that local models can be more powerful than the plain Dirichlet model. In addition, we observed that exact Bayesian reasoning can be computationally feasible in settings that are of practical interest. (Certainly there are also important cases where the presented algorithms are far too slow.)

Besides the computational challenges, finding the best partitioning constraints for a data analysis problem at hand is domain specific and a matter of background knowledge. If no single configuration of the constraints can be fixed, it is reasonable to average over different choices, whenever computationally feasible.

There are various directions for future research. E.g., it might be fruitful to consider cases where x, y, or both are continuous. For continuous x, existing methods rely on Markov chain Monte Carlo (Denison et al., 2002); it is not clear whether exact techniques apply. However, when just y is continuous the level-wise algorithm readily applies. Finally, we plan to apply the presented algorithms to genotype-phenotype analysis which aims at locating genes for complex diseases.

Acknowledgments

The authors are grateful to Heikki Mannila and to the anonymous reviewers for their helpful comments and suggestions. This work was partly supported and inspired by the Morgam (www.ktl.fi/morgam) and GenomeEUTwin (www.genomeutwin.org) projects.

References

Beinlich, I., Suermondt, G., Chavez, R., & Cooper, G. F. (1989). The ALARM monitoring system. *Proc. of the Second European Conference on Artificial Intelligence and Medicine* (pp. 247–256).

Chickering, D. M., Heckerman, D., & Meek, C. (1997). A Bayesian approach to learning Bayesian networks with local structure. *Proc. of the Thirteenth Conference on Uncertainty in Artificial Intelligence* (pp. 80–89).

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

Dechter, R. (1999). Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, *113*, 41–85.

Denison, D. G. T., Adams, N. M., Holmes, C. C., & Hand, D. J. (2002). Bayesian partition modelling. *Computational Statistics & Data Analysis*, *38*.

Friedman, N., & Goldszmidt, M. (1996). Learning Bayesian networks with local structure. *Proc. of the Twelfth Conference on Uncertainty in Artificial Intelligence* (pp. 252–262).

Geiger, D., & Heckerman, D. (2002). Parameter priors for directed acyclic graphical models and the characterization of several probability distributions. *The Annals of Statistics*, *30*, 1412–1440.

Golinelli, D., Madigan, D., & Consonni, G. (1999). Relaxing the local independence assumption for quantitative learning in acyclic directed graphical models through hierarchical partition models. *Proc. of the Seventh Internat. Workshop on Artificial Intelligence and Statistics.*

Heckerman, D., Geiger, D., & Chickering, D. M. (1995). Learning Bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, *20*, 197–243.

Kahn, J. (2002). Entropy, independent sets and antichains: A new approach to Dedekind's problem. *Proceedings of the American Mathematical Society*, *130*, 371–378.

Koivisto, M., & Sood, K. (2004). Exact Bayesian structure discovery in Bayesian networks. *Journal of Machine Learning Research*, *5*, 549–573.

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

Poole, D. (1997). Probabilistic Partial Evaluation: Exploiting rule structure in probabilistic inference. *Proc. of the Fifteenth International Joint Conference on Artificial Intelligence* (pp. 1284–1291).

Stearns, R. E., & Hunt III, H. B. (1996). An algebraic model for combinatorial problems. *SIAM Journal on Computing*, *25*, 448–476.