Probabilistic reasoning with terms

Peter A. Flach, Elias Gyftodimos and Nicolas Lachiche

No Institute Given

Abstract. Many problems in artificial intelligence can be naturally approached by generating and manipulating probability distributions over structured objects. First-order terms such as lists, trees and tuples and nestings thereof can represent individuals with complex structure in the underlying domain, such as sequences or molecules. Higher-order terms such as sets and multisets provide additional representational flexibility. In this paper we present two Bayesian approaches that employ such probability distributions over structured objects: the first is an upgrade of the well-known naive Bayesian classifier to deal with first-order and higher-order terms, and the second is an upgrade of propositional Bayesian networks to deal with nested tuples.

1 Introduction

Attempts to combine logic and probabilities have a long history. There are, roughly speaking, two possible approaches: one is to merge logic and probabilities into a single integrated system; the other is to combine them in a complementary fashion, employing each to address issues which the other is unable to handle. Rudolf Carnap's inductive logic is an example of the first approach: he introduces a degree of confirmation $c(H,E)$ which is a number between 0 and 1 expressing the degree to which evidence E confirms hypothesis H. A degree of confirmation of 1 is equivalent to logical entailment; a lower degree of confirmation 'may be regarded as a numerical measure for a partial entailment'.

We have argued previously [1, 2] against such a tight merger of logic and probabilities, because it leads to a degenerated form of logic. In particular, in inductive logic a proof amounts to calculating a degree of confirmation, therefore any hypothesis has a 'proof' from any evidence. Probabilities do establish a semantics of some sort, but not one that is sufficient to build a fully-fledged logical system. Proof theory requires a semantics which indicates what is preserved going from evidence to hypothesis; probability theory can only offer a truth-estimating semantics.

We therefore argue in favour of the complementary approach of combining logic and probabilities. For instance, in [1] we have developed qualitative logics of induction which are able to distinguish between hypotheses that explain certain evidence and hypotheses that do not, for various definitions of what it means to explain. These logics are built on a preservation semantics in which the property to be preserved from evidence to hypothesis is explanatory power: the hypothesis explains the evidence if it has at least the same explanatory power as the evidence. What such qualitative logics of induction cannot answer is: how plausible is this hypothesis, given this evidence? For answering such questions we can employ an additional probabilistic truth-estimating semantics.

We then have both a proof theory (only hypotheses explaining the evidence can be derived from it) and a probabilistic semantics assigning degrees of belief. Notice that the same probabilistic semantics may be used in combination with different non-deductive logics.

Halpern [4] has identified the use of probability for assigning degrees of belief as one of two ways of combining logic and probability, the other being defining probability distributions over domains of first-order variables. This paper follows the latter approach. We investigate how we can define probability distributions over structured objects represented by first- and higher-order terms. First-order terms such as lists, trees and tuples and nestings thereof can represent individuals with complex structure in the underlying domain, such as sequences or molecules. Higher-order terms such as sets and multisets provide additional representational flexibility. In this paper we present two Bayesian approaches that employ such probability distributions over structured objects: the first is an upgrade of the well-known naive Bayesian classifier to deal with first-order and higher-order terms, and the second is an upgrade of propositional Bayesian networks to deal with nested tuples.

2 Probability distributions over tuples, lists and sets

We start by considering the question: how to define probability distributions over complex objects such as lists and sets? We assume that we have one or more *alphabets* (or atomic types) $A = \{x_1, \ldots, x_n\}$ of atomic objects (e.g. integers or characters – we are only dealing with categorical data analysis in this paper), as well as probability distributions P_A over elements of the alphabets. Most of the distributions will make fairly strong independence assumptions, in the style of the naive Bayesian classifier.

2.1 Probability distributions over tuples

For completeness we start with the simplest case, where our complex objects are tuples or vectors. This is the only aggregation mechanism that has routinely been considered by statisticians and probability theorists.

Consider the set of k-tuples, i.e. elements of the Cartesian product of k alphabets: $(x_1, x_2, \ldots, x_k) \in A_1 \times A_2 \times \ldots \times A_k$.

Definition 1 (Distribution over tuples). *The following defines a distribution over tuples:*

$$P_{\text{tu}}((x_1, x_2, \ldots, x_k)) = \prod_{i=1}^{k} P_{A_i}(x_i)$$

Obviously, this distribution assumes that each component of the tuple is statistically independent of the others. Bayesian networks are one way of relaxing this assumption by explicitly modelling dependencies between variables. We will return to this in Section 3.

2.2 Probability distributions over lists

We can define a uniform probability distribution over lists if we consider only finitely many of them, say, up to and including length L. There are $\frac{n^{L+1}-1}{n-1}$ of those for $n > 1$, so under a uniform distribution every list has probability $\frac{n-1}{n^{L+1}-1}$ for $n > 1$, and probability $\frac{1}{L+1}$ for $n = 1$. Clearly, such a distribution does not depend on the internal structure of the lists, treating each of them as equiprobable.

A slightly more interesting case includes a probability distribution over lengths of lists. This has the additional advantage that we can define distributions over all (infinitely many) lists over A. For instance, we can use the geometric distribution over list lengths: $P_\tau(l) = \tau(1-\tau)^l$, with parameter τ denoting the probability of the empty list. Of course, we can use other infinite distributions, or arbitrary finite distributions, as long as they sum up to 1. The geometric distribution corresponds to the head-tail representation of lists.

We then need, for each list length l, a probability distribution over lists of length l. We can again assume a uniform distribution: since there are n^l lists of length l, we would assign probability n^{-l} to each of them. Combining the two distributions over list lengths and over lists of fixed length, we assign probability $\tau(\frac{1-\tau}{n})^l$ to any list of length l. Such a distribution only depends on the length of the list, not on the elements it contains.

We can also use the probability distribution P_A over the alphabet to define a non-uniform distribution over lists of length l. For instance, among the lists of length 3, list $[a,b,c]$ would have probability $P_A(a)P_A(b)P_A(c)$, and so would its 5 permutations. Combining P_A and P_τ thus gives us a distribution over lists which depends on the length and the elements of the list, but ignores their positions or ordering.

Definition 2 (Distribution over lists). *The following defines a probability distribution over lists:*

$$P_{\text{li}}([x_{j_1},\ldots,x_{j_l}]) = \tau(1-\tau)^l \prod_{i=1}^{l} P_A(x_{j_i})$$

where $0 < \tau \leq 1$ *is a parameter determining the probability of the empty list.*

Example 1 (Order-independent distribution over lists). Consider an alphabet $A = \{a,b,c\}$, and suppose that the probability of each element occurring is estimated as $P_A(a) = .2$, $P_A(b) = .3$, and $P_A(c) = .5$. Taking $\tau = (1-.2)(1-.3)(1-.5) = .28$, i.e. using the bitvector estimate of an empty list, we have $P_{A'}(a) = (1-.28)*.2 = .14$, $P_{A'}(b) = .22$, and $P_{A'}(c) = .36$, and $P_{\text{li}}([a]) = .28*.14 = .04$, $P_{\text{li}}([b]) = .06$, $P_{\text{li}}([c]) = .10$, $P_{\text{li}}([a,b]) = .28*.14*.22 = .009$, $P_{\text{li}}([a,c]) = .014$, $P_{\text{li}}([b,c]) = .022$, and $P_{\text{li}}([a,b,c]) = .28*.14*.22*.36 = .003$. We also have, e.g., $P_{\text{li}}([a,b,b,c]) = .28*.14*.22*.22*.36 = .0007$.

Notice that this and similar distributions can easily be represented by stochastic logic programs. For instance, the distribution of Example 1 corresponds to the following SLP:

```
0.2: element(a).
0.3: element(b).
```

```
0.5: element(c).

0.28: list([]).
0.72: list([H|T]):-element(H),list(T).
```

Not surprisingly, the predicate definitions involved are range-restricted type definitions; the probability labels either define a distribution exhaustively, or else follow the recursive structure of the type.

An alternative way to see such distributions is as follows. Introducing an extended alphabet $A' = \{\varepsilon, x_1, \ldots, x_n\}$ and a renormalised distribution $P_{A'}(\varepsilon) = \tau$ and $P_{A'}(x_i) = (1-\tau)P_A(x_i)$, we have $P_{\mathrm{li}}([x_{j_1}, \ldots, x_{j_l}]) = P_{A'}(\varepsilon)\prod_{i=1}^{l} P_{A'}(x_{j_i})$. That is, under P_{li} we can view each list as an infinite tuple of finitely many independently chosen elements of the alphabet, followed by the stop symbol ε representing an infinite empty tail.

If we want to include the ordering of the list elements in the distribution, we can assume a distribution P_{A^2} over pairs of elements of the alphabet, so that $[a,b,c]$ would have probability $P_{A^2}(ab)P_{A^2}(bc)$ among the lists of length 3. Such a distribution would be calculated by the following SLP:

```
0.05: pair(a,a).
0.10: pair(a,b).
0.05: pair(a,c).
...
0.15: pair(c,c).

0.28: listp([]).
0.13: listp([X]):-element(X).
0.59: listp([X,Y|T]):-pair(X,Y),listp([Y|T]).
```

Such a distribution would take some aspects of the ordering into account, but note that $[a,a,b,a]$ and $[a,b,a,a]$ would still obtain the same probability, because they consist of the same pairs (*aa*, *ab*, and *ba*), and they start and end with *a*. Obviously we can continue this process with triples, quadruples etc., but note that this is both increasingly computationally expensive and unreliable if the probabilities must be estimated from data.

A different approach is obtained by taking not ordering but position into account. For instance, we can have three distributions $P_{A,1}$, $P_{A,2}$ and $P_{A,3+}$ over the alphabet, for positions 1, 2, and 3 and higher, respectively. Among the lists of length 4, the list $[a,b,c,d]$ would get probability $P_{A,1}(a)P_{A,2}(b)P_{A,3+}(c)P_{A,3+}(d)$; so would the list $[a,b,d,c]$. Again, this wouldn't be hard to model with a stochastic logic program.

In summary, all except the most trivial probability distributions over lists involve (i) a distribution over lengths, and (ii) distributions over lists of fixed length. The latter take the list elements, ordering and/or position into account. We do not claim any originality with respect to the above distributions, as these and similar distributions are commonplace in e.g. bioinformatics. In the next section we use list distributions to define distributions over sets and multisets. Notice that, while lists are first-order terms, sets represent predicates and therefore are higher-order terms. In this respect our work extends related work on probability distributions over first-order terms.

2.3 Probability distributions over sets and multisets

A multiset (also called a bag) differs from a list in that its elements are unordered, but multiple elements may occur. Assuming some arbitrary total ordering on the alphabet, each multiset has a unique representation such as $\{[a,b,b,c]\}$. Each multiset can be mapped to the equivalence class of lists consisting of all permutations of its elements. For instance, the previous multiset corresponds to 12 lists. Now, given any probability distribution over lists that assigns equal probability to all permutations of a given list, this provides us with a method to turn such a distribution into a distribution over multisets. In particular, we can employ P_{li} above which defines the probability of a list, among all lists with the same length, as the product of the probabilities of their elements.

Definition 3 (Distribution over multisets). *For any multiset s, let l stand for its cardinality, and let k_i stand for the number of occurrences of the i-th element of the alphabet. The following defines a probability distribution over multisets:*

$$P_{\text{ms}}(s) = \frac{l!}{k_1!\ldots k_n!} P_{\text{li}}(s) = l!\tau \prod_i \frac{P_{A'}(x_i)^{k_i}}{k_i!}$$

where τ is a parameter giving the probability of the empty multiset.

Here, $\frac{l!}{k_1!\ldots k_n!}$ stands for the number of permutations of a list with possible duplicates.

Example 2 (Distribution over multisets). Continuing Example 1, we have $P_{\text{ms}}(\{[a]\}) = .04$, $P_{\text{ms}}(\{[b]\}) = .06$, $P_{\text{ms}}(\{[c]\}) = .10$ as before. However, $P_{\text{ms}}(\{[a,b]\}) = .02$, $P_{\text{ms}}(\{[a,c]\}) = .03$, $P_{\text{ms}}(\{[b,c]\}) = .04$, $P_{\text{ms}}(\{[a,b,c]\}) = .02$, and $P_{\text{ms}}(\{[a,b,b,c]\}) = .008$.

Notice that the calculation of the multiset probabilities involves explicit manipulation of the probabilities returned by the geometric list distribution. It is therefore hard to see how this could be equivalently represented by a stochastic logic program.

The above method, of defining a probability distribution over a type by virtue of that type being isomorphic to a partition of another type for which a probability distribution is already defined, is more generally applicable. Although in the above case we assumed that the distribution over each block in the partition is uniform, so that we only have to count its number of elements, this is not a necessary condition. Indeed, blocks in the partition can be infinite, as long as we can derive an expression for its cumulative probability. We will now proceed to derive a probability distribution over sets from distribution P_{li} over lists in this manner.

Consider the set $\{a,b\}$. It can be interpreted to stand for all lists of length at least 2 which contain (i) at least a and b, and (ii) no other element of the alphabet besides a and b. The cumulative probability of lists of the second type is easily calculated.

Lemma 1. *Consider a subset S of l elements from the alphabet, with cumulative probability $P_{A'}(S) = \sum_{x_i \in S} P_{A'}(x_i)$. The cumulative probability of all lists of length at least l containing only elements from S is $f(S) = \tau \frac{(P_{A'}(S))^l}{1-P_{A'}(S)}$.*

Proof. We can delete all elements in S from the alphabet and replace them by a single element x_S with probability $P_{A'}(S)$. The lists we want consist of l or more occurrences of x_S. Their cumulative probability is

$$\sum_{j \geq l} \tau (P_{A'}(S))^j = \tau \frac{(P_{A'}(S))^l}{1 - P_{A'}(S)}$$

$f(S)$ as defined in Lemma 1 is not a probability, because the construction in the proof includes lists that do not contain all elements of S. For instance, if $S = \{a, b\}$ they include lists containing only a's or only b's. More generally, for arbitrary S the construction includes lists over every possible subset of S, which have to be excluded in the calculation of the probability of S. In other words, the calculation of the probability of a set iterates over its subsets.

Definition 4 (Subset-distribution over sets). *Let S be a non-empty subset of l elements from the alphabet, and define*

$$P_{\text{ss}}(S) = \sum_{S' \subseteq S} (-P_{A'}(S'))^{l-l'} * f(S')$$

where l' is the cardinality of S', and $f(S')$ is as defined in Lemma 1. Furthermore, define $P_{\text{ss}}(\emptyset) = \tau$. $P_{\text{ss}}(S)$ is a probability distribution over sets.

Example 3 (Subset-distribution over sets). Continuing Example 2, we have $P_{\text{ss}}(\emptyset) = \tau = .28$, $P_{\text{ss}}(\{a\}) = f(\{a\}) = \tau \frac{P_{A'}(\{a\})}{1 - P_{A'}(\{a\})} = .05$, $P_{\text{ss}}(\{b\}) = .08$, and $P_{\text{ss}}(\{c\}) = .16$. Furthermore, $P_{\text{ss}}(\{a,b\}) = f(\{a,b\}) - P_{A'}(\{a\}) * f(\{a\}) - P_{A'}(\{b\}) * f(\{b\}) = .03$, $P_{\text{ss}}(\{a,c\}) = .08$, and $P_{\text{ss}}(\{b,c\}) = .15$. Finally, $P_{\text{ss}}(\{a,b,c\}) = f(\{a,b,c\}) - P_{A'}(\{a,b\}) * f(\{a,b\}) - P_{A'}(\{a,c\}) * f(\{a,c\}) - P_{A'}(\{b,c\}) * f(\{b,c\}) + (P_{A'}(\{a\}))^2 * f(\{a\}) + (P_{A'}(\{b\}))^2 * f(\{b\}) + (P_{A'}(\{c\}))^2 * f(\{c\}) = .18$.

P_{ss} takes only the elements occurring in a set into account, and ignores the remaining elements of the alphabet. For instance, the set $\{a, b, c\}$ will have the same probability regardless whether there is one more element d in the alphabet with probability p, or 10 more elements with cumulative probability p. This situation is analogous to lists. In contrast, the usual approach of propositionalising sets by representing them as bitvectors over which a tuple distribution can be defined has the disadvantage that the probability of a set depends on the elements in its extension as well as its complement. The subset-distribution is exponential in the cardinality of the set and is therefore expensive to compute for large sets. On the other hand, it can deal with finite subsets of infinite domains. The bitvector probability calculation is independent of the size of the subset, and cannot handle infinite domains.

2.4 Naive Bayes classification of terms

It is easy to nest the above distributions: if our terms are sets of lists, the alphabet over which the sets range are the collection of all possible lists, and the list distribution can

be taken as the distribution over that alphabet. The same holds for arbitrary nestings of types.

What can we do with probability distributions over terms? The usual things: inference, learning, prediction, and the like. We can do inference, either directly or through sampling, if we have completely specified distributions over all types involved. This involves queries of the type 'what is the probability of this term?'. We would normally assume that the type structure of the terms is known, so learning would just involve estimating the parameters of the distributions as well as the probabilities of atomic terms. This is usually fairly straightforward, for instance the maximum likelihood estimate of the parameter τ giving the probability of the empty list is $\frac{1}{1+\overline{l}}$, where $\overline{l}$ is the average length of a list.

In [6] we have used this approach to implement a naive Bayes classifier for first- and higher-order terms. We assume that the type structure is given (i.e., whether to treat a sequence of elements as a list or a set), and during the training phase the algorithm estimates the parameters and the atomic distributions. Predictions are made by calculating the posterior class distribution given the term to be classified. The approach is naive because of the strong independence assumptions made: for instance, for lists we use the distribution P_{li} defined above, which treats each element of the list as independent of the others and of its position.

An interesting point to note is that in the context of the naive Bayes classifier, the list and multiset distributions behave identically (the number of permutations of a given list is independent of the class). Somewhat more surprisingly, we found that the subset and list distributions behaved nearly identically as well.

In this section, we defined fairly naive probability distributions over tuples, lists, sets and multisets, and indicated how these could be used for naive Bayes classification. A natural next question is whether the approach carries over to Bayesian networks, of which the naive Bayes classifier is a special case. In the next section we will address this question for the special case of nested tuples.

3 Hierarchical Bayesian Networks

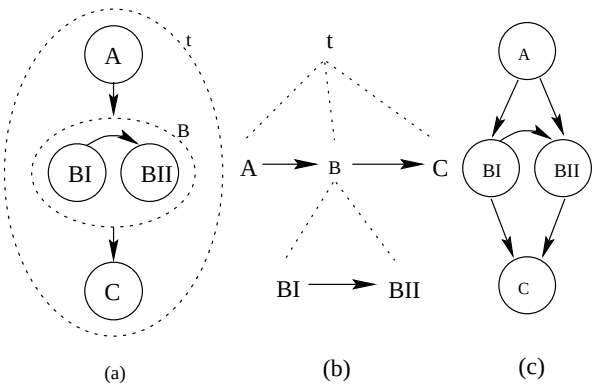
Bayesian Networks [7] are being used extensively for reasoning under uncertainty. Inference mechanisms for Bayesian Networks are compromised by the fact that they can only deal with propositional domains.

Hierarchical Bayesian Networks (HBNs) are an extension of Bayesian Networks, such that nodes in the network may correspond to (possibly nested) tuples of atomic types. Links in the network represent probabilistic dependencies the same way as in standard Bayesian Networks, the difference being that those links may lie at any level of nesting into the data structure.

3.1 Principles of Hierarchical Bayesian Networks

An HBN consists of two parts: the *structural* and the *probabilistic* part. The former describes the type hierarchy that builds the domains of the structured variables as tuples

of simpler types. It also contains the direct probabilistic dependencies, which are observed between elements of the same tuple. The latter contains the quantitative part of the conditional probabilities for the variables that are defined in the structural part.



P(A)	a1	a2
	0.3	0.7

P(BII|A,BI)	bII1	bII2	bII3
a1bI1	0.3	0.4	0.3
a1bI2	0.1	0.5	0.4
a2bI1	0.7	0.1	0.2
a2bI2	0.9	0.1	0

P(BI|A)	bI1	bI2
a1	0.4	0.6
a2	0.8	0.2

P(C|BI,BII)	c1	c2	c3
bI1bII1	0.2	0.6	0.2
bI1bII2	0.3	0.4	0.3
bI1bII3	0.2	0.2	0.6
bI2bII1	0.5	0.3	0.2
bI2bII2	0.6	0.2	0.2
bI2bII3	0.7	0.2	0.1

(d)

Fig. 1. A simple Hierarchical Bayesian Network. (a) Nested representation of the network structure. (b) Tree representation of the network structure. (c) Standard Bayesian Network expressing the same dependencies. (d) Probabilistic part.

Figure 1 presents a simple Hierarchical Bayesian Network. The structural part consists of three variables, A, B and C, where B is itself a pair (BI, BII). This may be represented either using nested nodes (a), or by a tree-like type hierarchy (b). We use the symbol t to denote a top-level composite node that includes all the variables of our world. In (c) it is shown how the probabilistic dependency links unfold if we flatten the hierarchical structure to a standard Bayesian Network.

In an HBN we observe two types of relationships between nodes: Relationships in the type structure (which we call *t-relationships*) and relationships that are formed by the probabilistic dependency links (*p-relationships*). We will make use of everyday terminology for both kinds of relationships, and refer to *parents, ancestors, siblings, spouses* etc. in the obvious meaning.

In the previous example, B has two t-children, namely BI and BII, one p-parent (A) and one p-child (C).

We also assume that a probabilistic dependency link scope "propagates" through the type structure, defining a set of *higher-level* probabilistic relationships.

Trivially, all p-parents of a node are also considered its higher-level parents. For example, we consider the higher-level parents of C to be B (as a trivial case), BI and BII (because they are t-descendants of B and there exists a p-link $B \rightarrow C$), while the higher-level parents of BII are BI (trivial) and A (because $A \rightarrow B$ and BII is a t-descendant of B).

For a more formal and complete description of the above notions, see [3].

3.2 Probability distribution decomposition

A Hierarchical Bayseian Network is a compact representation of the probabilistic independencies between the atomic types it contains, in a similar manner as in standard Bayesian Networks. Informally, we can express this as the following property:

> The value of an atomic variable is independent of all atomic variables that are not its higher-level descendants, given the value of its higher-level parents.

The independencies that an HBN describes can be exploited using the chain rule of conditional probability, to decompose the full joint probability of all the atomic types into a product of the conditional probabilities, in the following way:

$$P(x) = \begin{cases} P_{A_X}(x) & \text{if } x \in A_X \text{ is atomic} \\ \prod_{i=1}^{n} P(x_i | Par(x_i)) & \text{otherwise} \end{cases}$$

where $x_1, x_2, \ldots, x_n$ are the components of x and $Par(x_i)$ are the *direct p-parents* of x_i in the structure.

3.3 Inference in Hierarchical Bayesian Networks

Inference algorithms used for standard Bayesian Networks can also be applied to the Hierarchical Bayesian Network model. Backward reasoning algorithms [8] and message passing algorithms [7] can be used if we restrict our focus on the corresponding Bayesian Networks. The additional information in the Hierarchical Bayesian Network may serve towards a better interpretation of the resulting probability distributions.

Standard inference algorithms are not directly applicable to networks that contain loops (a loop is a closed path in the underlying undirected graph structure). One technique of coping with loops is to merge groups of variables into compound nodes, eliminating the circles in the graph. In the case of Hierarchical Bayesian Networks, knowledge of the hierarchical structure can serve as a guideline for which nodes to merge, in such a way that the resulting network would allow a more meaningful interpretation.

Message Propagation Algorithm A very efficient method of calculating beliefs in polytree-structured standard Bayesian Networks (i.e., that do not contain any undirected closed path) is the message-passing algorithm described in [7]. Every node in the network is associated to a *belief state*, that is a vector whose elements sum up to one and correspond to the proportionate beliefs that each value in the domain may be the variable's value, given all available knowledge. The belief state of every node can be directly retrieved given the belief states of its parents and children. Whenever a change in a node's belief state occurs, either forced by some direct observation or indirectly, due to a change of the state of a neighbour, the node calculates its new belief state and propagates the relevant information to its parents and children (Fig. 2). The algorithm then repeats until the network reaches an equilibrium.

Belief calculation is performed locally in three independent steps:

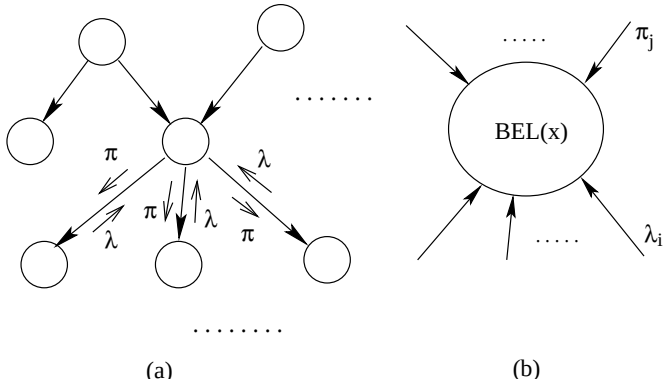


Fig. 2. Belief propagation in polytrees. (a) Message passing in polytree structures. (b) Local belief update.

Belief updating: Calculating a node's belief vector, using the latest information available from its neighbours.
Bottom-up propagation: Computing the λ *messages* that will be sent to parent nodes.
Top-down propagation: Computing the π *messages* that will be sent to children nodes.

The locality of the above algorithm is based on probabilistic independencies that result from the assumption that the network does not contain any undirected circles. If a network does contain loops, either an equilibrium cannot be reached, or, if it can be, it will not necessarily represent the actual joint probability distribution.

This algorithm may be directly applied to Hierarchical Bayesian Networks in the trivial case where the corresponding Bayesian Network does not contain undirected cycles. Even if individual composite nodes contain no loops, these will occur in the corresponding Bayesian Network in any case where some composite node participates in more than one p-relationship. The only case where loops will not occur is if we allow for polytree-like structures, where only leaf nodes may be composite, under the further restriction of not containing any p-links.

Structures containing loops In the case where probabilistic dependencies form loops in the network infrastructure (i.e., the undirected network) the above algorithm cannot be applied directly. We define a Hierarchical Bayesian Network to contain loops if its corresponding Bayesian Network contains loops. There are several approaches that can be used to perform inference on a network containing loops [7, 5]. As a trivial case, these methods can be applied to any Hierarchical Bayesian Network after flattening it to a standard Bayesian Network. Here we will restrict our discussion on an existing clustering method and show how it can be specifically adapted to the Hierarchical Bayesian Network case.

Clustering methods eliminate loops by grouping together clusters of two or more vertices into composite nodes. Different cluster selections may yield different polytrees

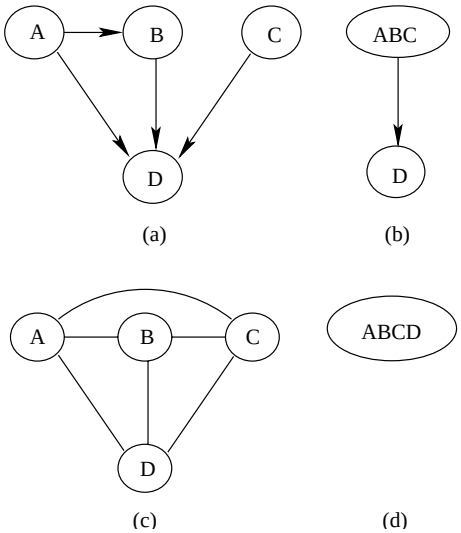


Fig. 3. Coping with loops. (a) A Bayesian Network containing the loop ABDA. (b) Grouping all non-leaf variables in a single cluster. (c) Equivalent Markov network. (d) Join tree algorithm results in a single cluster.

when applied to a given Bayesian Network. As an extreme case, all non-leaf nodes may be grouped in a single cluster (Fig. 3(b)).

One popular method, described in [7], is based on the construction of *join trees*. Briefly, the technique consists in building a triangulated undirected graph G (i.e., a Markov Network) that represents independency relations similar to the original Bayesian Network, and then linking the maximal cliques of G to form a tree structure. The advantage of this method is that the resulting directed acyclic structure is a tree, making the application of message propagation highly efficient. The trade-off is that the common parents of a node in the original Bayesian Network, along with the node itself, will be grouped into a single cluster, so information about independencies between them will be lost (Fig. 3(c,d)).

The same algorithm can be applied directly on any fully flattened Hierarchical Bayesian Network. However, we can make use of the additional information that a Hierarchical Bayesian Network structure contains to arrive to more informative structures. The method we introduce is the following algorithm:

Algorithm 1 (HBN-decycling algorithm) *To* decycle *a node v:*

- *If v is a non-leaf node:*
  - *Decycle all components of v*
  - *If v participates in two or more p-edges, prune the network structure on v.*
  - *If v has exactly one p-parent (or p-child), flatten the structure on v replacing every p-connected subset of the t-children of v by a single cluster.*
  - *If v has no p-parents or p-children, flatten the structure on the node v.*

– *If v is a leaf node, leave v unchanged*

By applying the *HBN-decycling* algorithm on a Hierarchical Bayesian Network and then retreiving its corresponding Bayesian Network, we arrive at a *polytree* Bayesian Network. The application of the inference algorithm for Bayesian Networks is not as efficient for polytrees as it is for standard trees, but this is balanced by the smaller size of individual nodes.

In figure 4(a) we see an HBN-tree structure containing a loop. The structure is similar to the Bayesian Network in figure 3(a), with nodes *A* and *B* additionally forming a composite node. The result of the decycling algorithm (Fig. 4(b)) retains much more structural information from the original structure.

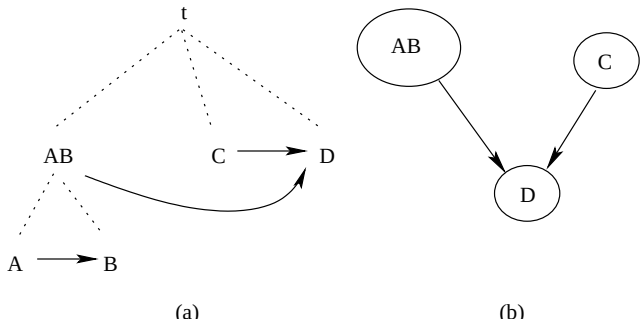


Fig. 4. (a) A Hierarchical Bayesian Network containing the loop ABDA. (b) Result of the *HBN-decycling* algorithm.

4 Concluding remarks

In this paper we have considered the issue of probabilistic reasoning with terms. Aggregation mechanisms such as lists and sets are well-understood by logicians and computer scientists and occur naturally in many domains; yet they have been mostly ignored in statistics and probability theory, where tupling seems the only aggregation mechanism. The individuals-as-terms perspective has been explored in machine learning in order to investigate the links between approaches that can deal with structured data such as inductive logic programming, and more traditional approaches which assume that all data are in a fixed attribute-value vector format. This has been very fruitful because it has paved the way for upgrading well-known propositional techniques to the first- and higher-order case.

In our work we are exploring similar upgrades of propositional probabilistic models to the first- and higher-order case. We have done this upgrade for the naive Bayes classifier but further work remains, in particular regarding an extended vocabulary of parametrised probability distributions from which to choose. In the case of Bayesian

networks, we are currently concentrating on a restricted upgrade considering nested tuples only. Our approach takes advantage of existing Bayesian Network methods, and is an elegant way of incorporating into them specific knowledge regarding the structure of the domain. Our current work is focused on implementing inference and learning methods for Hierarchical Bayesian Networks, aiming to be tested against the performance of standard Bayesian Networks. Further on, we plan to introduce more aggregation operators for types, such as lists and sets. This will demand somewhat more sophisticated probability decomposition methods than the cartesian product, and will allow the application of the model to structures of arbitrary form and length, such as web pages or DNA-sequences.

References

1. P.A. Flach. *Conjectures: An Inquiry Concerning the Logic of Induction*. PhD thesis, Katholieke Universiteit Brabant, 1995.
2. Peter A. Flach. Logical characterisations of inductive learning. In Dov M. Gabbay and Rudolf Kruse, editors, *Handbook of defeasible reasoning and uncertainty management systems, Vol. 4: Abductive reasoning and learning*, pages 155–196. Kluwer Academic Publishers, October 2000.
3. Elias Gyftodimos and Peter A. Flach. Hierarchical bayesian networks: A probabilistic reasoning model for structured domains. In Edwin de Jong and Tim Oates, editors, *Proceedings of the ICML-2002 Workshop on Development of Representations*. University of New South Wales, 2002.
4. J. Halpern. An analysis of first-order logics of probability. *Artificial Intelligence*, 46:311–350, 1990.
5. M. Henrion. Propagating uncertainty by logic sampling in bayes' networks. Technical report, Department of Engineering and Public Policy, Carnegie-Mellon University, 1986.
6. Nicolas Lachiche and Peter A. Flach. 1bc2: a true first-order bayesian classifier. In *Proceedings of the 12th International Conference on Inductive Logic Programming*, 2002.
7. Judea Pearl. *Probabilistic Reasoning in Intelligent Systems — Networks of Plausible inference*. Morgan Kaufmann, 1988.
8. Stuart J. Russell and Peter Norvig. *Artificial intelligence, a modern approach*. Prentice Hall, 2nd edition, 1995.