

Linköping Electronic Articles in  
Computer and Information Science  
Vol. 7(2002): nr 11

# Probabilistic reasoning with terms

Peter A. Flach

Elias Gyftodimos

University of Bristol

Nicolas Lachiche

Louis Pasteur University, Strasbourg

Linköping University Electronic Press  
Linköping, Sweden

<http://www.ep.liu.se/ea/cis/2002/011/>

*Published on September 15, 2002 by  
Linköping University Electronic Press  
581 83 Linköping, Sweden*

**Linköping Electronic Articles in  
Computer and Information Science**  
*ISSN 1401-9841  
Series editor: Erik Sandewall*

©2002 Peter A. Flach, Elias Gyftodimos and Nicolas Lachiche  
*Typeset by the author using L<sup>A</sup>T<sub>E</sub>X  
Formatted using étendu style*

**Recommended citation:**

*<Author>. <Title>. Linköping Electronic Articles in  
Computer and Information Science, Vol. 7(2002): nr 11.  
<http://www.ep.liu.se/ea/cis/2002/011/>. September 15, 2002.*

*This URL will also contain a link to the author's home page.*

*The publishers will keep this article on-line on the Internet  
(or its possible replacement network in the future)  
for a period of 25 years from the date of publication,  
barring exceptional circumstances as described separately.*

*The on-line availability of the article implies  
a permanent permission for anyone to read the article on-line,  
to print out single copies of it, and to use it unchanged  
for any non-commercial research and educational purpose,  
including making copies for classroom use.  
This permission can not be revoked by subsequent  
transfers of copyright. All other uses of the article are  
conditional on the consent of the copyright owner.*

*The publication of the article on the date stated above  
included also the production of a limited number of copies  
on paper, which were archived in Swedish university libraries  
like all other written works published in Sweden.  
The publisher has taken technical and administrative measures  
to assure that the on-line version of the article will be  
permanently accessible using the URL stated above,  
unchanged, and permanently equal to the archived printed copies  
at least until the expiration of the publication period.*

*For additional information about the Linköping University  
Electronic Press and its procedures for publication and for  
assurance of document integrity, please refer to  
its WWW home page: <http://www.ep.liu.se/>  
or by conventional mail to the address stated above.*

## **Abstract**

Many problems in artificial intelligence can be naturally approached by generating and manipulating probability distributions over structured objects. In this paper we represent structured objects by first-order logic terms (lists, trees, tuples, and nestings thereof) and higher-order terms (sets, multisets), and we study the question how to define probability distributions over such terms. 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

Halpern [7] provides a categorisation of formalisms that combine first order logics with uncertainty. A probabilistic model may be either a chance setup, i.e. a distribution over some domain, or a set of degrees of belief, which is a notion closer to a distribution over possible worlds. The statement “The probability that a randomly chosen bird can fly is greater than 0.9” describes a distribution over individuals, about which statistical information reveals that some are expected to have a particular property. Alternatively, we could say that the result of a stochastic experiment in that world is expected to give a result according to that distribution. On the other hand, the statement “The probability that Tweety (a particular bird) can fly is greater than 0.9” implies that there are several worlds, and the property of flying may be defined in such a way that Tweety satisfies it in some worlds, but not in others. That defines some probability distribution over these possible worlds. Halpern presents two extensions of first order logic that capture these above mentioned notions, (*Type-1 and Type-2 probability structures, respectively*).

This paper follows the Type-1 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.

Among various approaches aiming at combining first-order logic with probabilistic inference, our work is perhaps most closely related to Stochastic Logic Programs (SLPs). Stochastic Logic Programs [14, 3] are logic programs that have numerical values (*labels*) attached to the clauses that define some of their predicates. Predicate labels can be used to define probability distributions over *derivations* of a goal, *refutations* of a goal and *atoms* resulting from a goal. Consider the following example:

0.6:  $s(X, Y) : \neg p(X), q(Y)$ .

0.4:  $s(X, X) : \neg r(X), q(X)$ .

$p(c)$ .

0.2:  $q(b)$ .

0.8:  $q(c)$ .

0.5:  $r(a)$ .

0.5:  $r(c)$ .

The goal  $?-s(X, Y)$  yields six distinct derivations, three of which lead to failure, the remaining three being successful refutations. One of these refutations yields the atom  $s(c, b)$ , and two others  $s(c, c)$ . To compute the above mentioned distribution over derivations, we assign each derivation a probability equal to the product of all labels of clauses contained in it. For the distribution over refutations, we normalise by dividing with the sum of the probabilities of all successful refutations. For the distribution over atoms, we just add together the probabilities of refutations that yield the same atom.

In this paper we take a more restricted viewpoint, in that we are only interested in manipulating probability distributions over terms. On the other hand, our approach generalises SLPs in that we define several distributions that cannot be defined by an SLP. Our overall goal is to employ these distributions in Bayesian

approaches to dealing with structured data, in particular the naive Bayes classifier and Bayesian Networks.

The outline of the paper is as follows. In Section 2 we derive probability distributions over tuples, lists, multisets, and sets. In Section 3 we use these distributions in a naive Bayes classifier for structured data. In Section 4 we consider the more general case of Bayesian networks over structured terms. Section 5 concludes with a discussion of related work, and a summary of the main conclusions.

## 2 Probability distributions over terms

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)  $A = \{x_1, \dots, x_n\}$  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:  $(x_1, x_2, \dots, x_k) \in A_1 \times A_2 \times \dots \times A_k$ .

**Lemma 2.1 (Distribution over tuples)** *Let  $P_{\text{tu}}$  be defined as follows:*

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

$P_{\text{tu}}$  is a probability distribution over tuples.

*Proof.* We need to prove that (1)  $0 \leq P_{\text{tu}}(x) \leq 1$  for all tuples  $x$ , and (2)  $\sum_x P_{\text{tu}}(x) = 1$  when summing over all  $x \in A_1 \times A_2 \times \dots \times A_k$ . (1) follows because  $P_{\text{tu}}(x)$  is a product of probabilities; (2) follows by simple algebraic manipulation, rewriting the sum of products into a product of sums over each individual alphabet.

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 4.

A special case of the tuple distribution is obtained when each component takes on the values 0 and 1: the tuples are so-called *bitvectors*. Associating each bit in the bitvector with a particular object of a given universe of  $k$  objects, a bitvector can be seen as representing the set of objects whose bit is set to 1 in the bitvector. This provides a very simple way of defining a probability distribution over sets. The bitvector distribution has however two problems: (i) it can only be defined over a finite universe, and (ii) for small subsets of the universe the complement of the set (i.e. the bits set to 0) dominates. The subset distribution we define in Section 2.3 overcomes both problems.

## 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  $\tau$  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 a 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  $P_\tau$  thus gives us a distribution over lists which depends on the length and the elements of the list, but ignores their positions or ordering.

**Theorem 2.2 (Distribution over lists)** *Let  $P_{\text{li}}$  be defined as follows:*

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

where  $0 < \tau \leq 1$  is a parameter determining the probability of the empty list.  $P_{\text{li}}$  is a probability distribution over lists.

*Proof.* We need to prove that (1)  $0 \leq P_{\text{li}}(x) \leq 1$  for all lists  $x$ , and (2)  $\sum_x P_{\text{li}}(x) = 1$  when summing over all possible lists  $x$ . Introducing an extended alphabet  $A' = \{\epsilon, x_1, \dots, x_n\}$  and a renormalised distribution  $P_{A'}(\epsilon) = \tau$  and  $P_{A'}(x_i) = (1-\tau)P_A(x_i)$ , we have  $P_{\text{li}}([x_{j_1}, \dots, x_{j_l}]) = P_{A'}(\epsilon) \prod_{i=1}^l P_{A'}(x_{j_i})$ . That is, under  $P_{\text{li}}$  we can view each list as an infinite tuple of finitely many independently chosen elements of the alphabet, followed by the stop symbol  $\epsilon$  representing an infinite empty tail. (1) now follows because  $P_{\text{li}}(x)$  is again a product of probabilities.

For (2), let  $k_i$  stand for the number of occurrences of the  $i$ -th element of the alphabet in a given list  $x$ ; we then have  $P_{\text{li}}(x) = \tau \prod_i P_{A'}(x_i)^{k_i}$ . There are  $\frac{l!}{k_1! \dots k_n!}$  distinct permutations of this list, each with the same probability ( $l = \sum_i k_i$  is the number of elements in the list), with cumulative probability  $\frac{l!}{k_1! \dots k_n!} \tau \prod_i P_{A'}(x_i)^{k_i}$ , i.e., a term in a multinomial series. Summing over all non-negative  $k_i$  gives  $\tau(\sum_i P_{A'}(x_i))^l = \tau(1-\tau)^l$ ; summing over all non-negative  $l$  yields 1 as required.

*Example 2.1 (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 bivector estimate of an empty list), we have  $P_{A'}(a) = (1-.28) \cdot .2 = .14$ ,  $P_{A'}(b) = .22$ , and  $P_{A'}(c) = .36$ , and  $P_{\text{li}}([a]) = .28 \cdot .14 = .04$ ,  $P_{\text{li}}([b]) = .06$ ,

$P_i([c]) = .10$ ,  $P_i([a, b]) = .28 * .14 * .22 = .009$ ,  $P_i([a, c]) = .014$ ,  $P_i([b, c]) = .022$ , and  $P_i([a, b, c]) = .28 * .14 * .22 * .36 = .003$ . We also have, e.g.,  $P_i([a, b, b, c]) = .28 * .14 * .22 * .22 * .36 = .0007$ .

Notice that this and similar distributions can easily be represented by stochastic logic programs [14, 3]. For instance, the distribution of Example 2.1 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.

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  $\{\{a, b, b, c\}\}$ .<sup>1</sup> 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  $P_{li}$  from the previous section which defines the probability of a list, among all lists with the same length, as the product of the probabilities of their elements.

**Theorem 2.3 (Distribution over multisets)** *For any multiset  $x$ , let  $k_i$  stand for the number of occurrences of the  $i$ -th element of the alphabet, and let  $l = \sum_i k_i$  denote the cardinality of the multiset. Define  $P_{ms}$  as follows:*

$$P_{ms}(x) = \tau l! \prod_i \frac{P_{A'}(x_i)^{k_i}}{k_i!}$$

where  $0 < \tau \leq 1$  is a parameter determining the probability of the empty multiset.  $P_{ms}$  is a probability distribution over multisets.

*Proof.* The number of permutations of the elements of  $x$  is  $\frac{l!}{k_1! \dots k_n!}$ , and the cumulative probability of these permutations under the list distribution is  $\frac{l!}{k_1! \dots k_n!} P_{li}(x) = P_{ms}(x)$ .

*Example 2.2 (Distribution over multisets)* *Continuing Example 2.1, we have  $P_{ms}(\{\{a\}\}) = .04$ ,  $P_{ms}(\{\{b\}\}) = .06$ ,  $P_{ms}(\{\{c\}\}) = .10$  as before. However,  $P_{ms}(\{\{a, b\}\}) = .02$ ,  $P_{ms}(\{\{a, c\}\}) = .03$ ,  $P_{ms}(\{\{b, c\}\}) = .04$ ,  $P_{ms}(\{\{a, b, c\}\}) = .02$ , and  $P_{ms}(\{\{a, b, b, c\}\}) = .008$ .*

Notice that the calculation of the multiset probabilities involves explicit manipulation of the probabilities returned by the geometric list distribution. As a consequence, it is not directly expressible as a stochastic logic program (this would require the SLP-equivalent of the set of /3 predicate, calculating all solutions to a query and the associated probability mass).

Similarly, a set can be seen as the equivalence class of all lists (or multisets) containing each element of the set at least once, and no other elements. For instance, the set  $\{a, b\}$  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 satisfying condition (ii) is easily calculated, but the first condition is conditionally expensive as it requires iterating over all subsets of the set.

**Theorem 2.4 (Subset distribution over sets)** *Let  $S$  be a non-empty subset of  $l$  elements from the alphabet, and define*

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

where  $l'$  is the cardinality of  $S'$ ,  $P_{A'}(S') = \sum_{x \in S'} P_{A'}(x)$ , and  $0 < \tau \leq 1$  is a parameter determining the probability of the empty set.  $P_{ss}(S)$  is a probability distribution over sets.

<sup>1</sup>In this paper, we use the notation  $\{\dots\}$  to denote multisets.

*Proof.* This follows by an application of the Inclusion-Exclusion Principle (a special case of which is  $|A_1 \cup \dots \cup A_n| = \sum_i |A_i| - \sum_{i,j} |A_i \cap A_j| + \sum_{i,j,k} |A_i \cap A_j \cap A_k| - \dots + (-1)^{n-1} |A_1 \cap \dots \cap A_n|$ ). For any set  $S$ , the cumulative probability of all lists with at least  $l$  elements all of which are in  $S$  is

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

However, we have to discount all lists which do not contain all elements from  $S$ , for which we use the Inclusion-Exclusion Principle to obtain the required expression.

*Example 2.3 (Subset distribution over sets)* Continuing Example 2.2, we have  $P_{ss}(\emptyset) = \tau = .28$ ,  $P_{ss}(\{a\}) = \tau \frac{P_{A'}(\{a\})}{1 - P_{A'}(\{a\})} = .05$ ,  $P_{ss}(\{b\}) = .08$ , and  $P_{ss}(\{c\}) = .16$ . Furthermore,  $P_{ss}(\{a, b\}) = \tau \left( \frac{P_{A'}(\{a, b\})^2}{1 - P_{A'}(\{a, b\})} - \frac{P_{A'}(\{a\})^2}{1 - P_{A'}(\{a\})} - \frac{P_{A'}(\{b\})^2}{1 - P_{A'}(\{b\})} \right) = 0.3$ ,  $P_{ss}(\{a, c\}) = .08$ , and  $P_{ss}(\{b, c\}) = .15$ . Finally,  $P_{ss}(\{a, b, c\}) = \tau \left( \frac{P_{A'}(\{a, b, c\})^3}{1 - P_{A'}(\{a, b, c\})} - \frac{P_{A'}(\{a, b\})^3}{1 - P_{A'}(\{a, b\})} - \frac{P_{A'}(\{a, c\})^3}{1 - P_{A'}(\{a, c\})} - \frac{P_{A'}(\{b, c\})^3}{1 - P_{A'}(\{b, c\})} + \frac{P_{A'}(\{a\})^3}{1 - P_{A'}(\{a\})} + \frac{P_{A'}(\{b\})^3}{1 - P_{A'}(\{b\})} + \frac{P_{A'}(\{c\})^3}{1 - P_{A'}(\{c\})} \right) = .18$ .

$P_{ss}$  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. Another disadvantage of the bitvector representation is that it is impossible to handle infinite domains, whereas the subset distribution is applicable to finite subsets of infinite domains.

An obvious disadvantage of the subset distribution is that it is exponential in the cardinality of the set and therefore expensive to compute for large sets. As it turns out, the list distribution can be used to approximate the subset distribution for any set  $S$  such that  $P_{A'}(S)$  is close to 0, which is likely in the case of large alphabets (e.g., sets ranging over structured objects, see Section 3).

**Theorem 2.5 (List distribution approximates subset distribution)** *For any set  $S$ , if  $P_{A'}(S)$  is close to 0, then  $P_{ss}(S)$  is close to  $l!P_{li}(L_S)$ , where  $L_S$  is a list constructed from  $S$  by choosing an arbitrary order for its elements.*

*Proof.* If  $P_{A'}(S)$  is close to 0 then  $1 - P_{A'}(S') \approx 1$  for all  $S' \subseteq S$ , hence  $P_{ss}(S) \approx \sum_{S' \subseteq S} \tau(-1)^{|S|-|S'|} P_{A'}(S')^{|S'|}$ . Each term  $\tau P_{A'}(S')^{|S'|}$  represents all possible lists of  $l$  elements (not necessarily distinct) chosen from  $S'$ . The alternating sum is again an application of the Inclusion-Exclusion Principle, resulting in the probability of all lists of  $l$  distinct elements from  $S$ , i.e.  $\tau! \prod_{x \in S} P_{A'}(x) = l!P_{li}(L_S)$ .

*Example 2.4 (Subset distribution vs. list distribution)* Suppose now that  $P_{A'}(a) = .014$ ,  $P_{A'}(b) = .022$ , and  $P_{A'}(c) = .036$ , i.e., each probability has decreased to 0.1 of its original value; the distribution of the probability mass over the remainder of the alphabet is left unspecified. This modifies the subset distribution as follows:  $P_{ss}(\{a\}) = .0040$ ,  $P_{ss}(\{b\}) = .0059$ , and  $P_{ss}(\{c\}) = .0097$ ;  $P_{ss}(\{a, b\}) = .00018$ ,  $P_{ss}(\{a, c\}) = .00031$ , and  $P_{ss}(\{b, c\}) = .00048$ ; and  $P_{ss}(\{a, b, c\}) = .000022$ . On the other hand,  $P_{li}([a]) = .0044$ ,  $P_{li}([b]) = .0066$ , and  $P_{li}([c]) = .0101$ ;  $2P_{li}([a, b]) = .00017$ ,  $2P_{li}([a, c]) = .00029$ , and  $2P_{li}([b, c]) = .00044$ ; and  $6P_{li}([a, b, c]) = .000019$ .

In all our experiments with the first-order naive Bayesian classifier 1BC2 (see Section 3), we found indeed that – whenever it was computationally feasible to use the subset distribution – it yielded virtually identical results to the list distribution. This reinforces the importance of  $P_i$  (which admittedly is not very good for lists in that it disregards the order of their elements) as a useful and computationally feasible approximation to the subset distribution.

## 2.4 Probability distributions over nested 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 terms. We just need to define the structure of these terms, i.e., their type.

Types can be defined in various ways, some of which we consider here by way of examples. Mutagenesis is a well-known ILP benchmark problem in which organic molecules need to be classified as mutagenic or not. Molecules consist of atoms and bonds between atoms; each of these entities have properties (e.g., an atom has an electric charge, and a bond has a type). Here is a type definition for mutagenesis written in the Escher language, which is a higher-order logic and functional programming language [13]:

```
type Molecule = (Atoms,LUMO,LogP);
type Atoms = {Atom};
type LUMO = Float;
type LogP = Float;
type Atom = (Element,AtomType,Charge,FromBonds,ToBonds);
data Element = Br | C | Cl | F | H | I | N | O | S;
type AtomType = Int;
type Charge = Float;
type FromBonds = {Bond};
type ToBonds = {Bond};
type Bond = (BondType);
type BondType = Int;
```

A molecule is a tuple consisting of a set of atoms, and two attributes LUMO and LogP, which represent measurements of certain chemical properties of molecules. An atom is a tuple consisting of attributes Element (an enumerated datatype), AtomType (an integer), and Charge (a floating point number). The remainder of the type structure deals with bonds between atoms.

The presence of bonds means that molecules are graphs rather than trees. In pure term representations, graph structures such as molecules always pose representational problems.<sup>2</sup> In this simplified example we obtain a tree representation from a molecule by cutting each bond in the middle. In order to maintain consistency with the benchmark dataset, bonds are directed. Consequently, each atom has two associated sets of bonds: the ones which depart from that atom, and the ones which arrive at that atom. Bonds can have several attributes; for simplicity we have represented it as a 1-tuple with a single attribute BondType. Figure 1 gives a graphical representation of this type structure.

Given such a type structure, we can compute a probability distribution over molecules in a bottom-up fashion. For instance, once we know the distribution  $P_{\text{BondType}}$  over the alphabet of bond types, we can calculate  $P_{\text{FromBonds}}$  and  $P_{\text{ToBonds}}$

<sup>2</sup>In the flattened representation used by 1BC2, we have more flexibility because subterms are named (see Section 3).

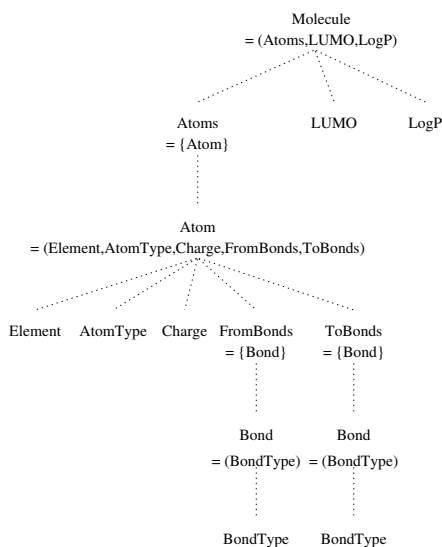


Figure 1: Type structure used in the mutagenesis domain.

using the subset distribution or its approximation by the list distribution (provided we know the parameter of this distribution). We can then combine this with the distributions  $P_{\text{Element}}$ ,  $P_{\text{AtomType}}$  and  $P_{\text{Charge}}$  using the tuple distribution, yielding  $P_{\text{Atom}}$ . Using the subset distribution once again, and combining this with  $P_{\text{LUMO}}$  and  $P_{\text{LogP}}$  using the tuple distribution, we finally obtain  $P_{\text{Molecule}}$ .

In summary, in this section we defined probability distributions over tuples, lists, sets and multisets, that can be composed to obtain distributions over arbitrarily nested terms. In the next sections we show how these distributions can be used in a Bayesian context. In Section 3 we discuss naive Bayes classification of terms. In Section 4 we consider the more general case of Bayesian networks over structured terms.

### 3 Naive Bayes classification of terms

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.

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,  $P_{\text{li}}$  treats each element of the list as independent of the others and of its position, while  $P_{\text{tu}}$  treats each component of the tuple as independent of the others.

In its general form, a Bayesian classifier predicts the most likely class value  $c$  of an object given its description  $d$ :  $\text{argmax}_c P(c|d)$ . Applying Bayes’ theo-

rem, this formula can be rewritten as  $\operatorname{argmax}_c \frac{P(d|c)P(c)}{P(d)} = \operatorname{argmax}_c P(d|c)P(c)$ . In an attribute-value representation, the object is described by its values  $a_1, \dots, a_n$  for a fixed set of attributes  $A_1, \dots, A_n$ . Since it is difficult to get a good estimate of  $P(a_1, \dots, a_n|c)$ , the naive Bayes assumption consists in decomposing this probability distribution into the product of the probability of each attribute value:  $P(a_1|c) \times \dots \times P(a_n|c)$ , assuming that the probability of attribute  $A_i$  taking on value  $a_i$  is independent of the probabilities of the values taken by the other attributes. While this independence assumption is rarely met in practice, naive Bayes classification is often surprisingly accurate. This can be explained by the fact that we use the estimated probabilities only for classification – e.g., if we rank all instances  $i$  by their predicted  $P(i|c_0)$  for some class  $c_0$ , we want all instances of that class to appear before any instances of another class, but we do not care about the ranking among instances of the same class.

The 1BC system we developed earlier [4] generates a set of first-order features that are used as attributes in a classical attribute-value naive Bayesian classifier. It can thus be classified as a propositionalisation approach (although the propositionalisation is done dynamically, not in a pre-processing step as in LINUS [12]). In this section we propose a different approach that is less ‘propositional’ and directly considers probability distributions on structured individuals made of sets, tuples and multisets: the 1BC2 first-order Bayesian classifier.

This section is structured as follows. In Section 3.1 we define the flattened first-order language the 1BC2 employs. In Section 3.2 we describe 1BC2’s learning and classification algorithms. In Section 3.3 we give some experimental results.

### 3.1 1BC2 language bias

1BC2 makes use of a data representation similar to the one of 1BC [4]. Specifically, 1BC2 uses a flattened language (i.e., Datalog) where all subterms (except components of a tuple) of a term are named, and the structure of individuals is represented by so-called structural predicates.

**Definition 3.1 (Structural predicate)** *A structural predicate is a binary predicate representing the relation between one type and another. Given an individual of one type, a functional structural predicate, or structural function, refers to a unique individual of the other type, while a non-determinate structural predicate is non-functional.*

**Definition 3.2 (Property)** *A property is a predicate characterising an individual. A parameter is an argument of a property which is always instantiated. If a property has no parameter (or only one instantiation of its parameters), it is boolean, otherwise it is multivalued.*

1BC2 makes use of an ISP (Individual-Structural-Properties) declaration where all structural predicates and properties are declared. Here is the ISP declaration for mutagenesis (see Section 2.4):

```
--INDIVIDUAL
molecule 1 mol
--STRUCTURAL
mol2atom 2 1:mol *:atom 1 li
fr_atom2bond 2 1:atom *:bond 1 li
to_atom2bond 2 1:atom *:bond 1 li
--PROPERTIES
class 2 mol #classP 1
lumo 2 mol #lumoP 1
```

```

logp 2 mol #logpP 1
element 2 atom #elementP 1
atomtype 2 atom #atomtypeP 1
charge 2 atom #chargeP 1
bondtype 2 bond #bondtypeP 1

```

Parameters are denoted by a #, e.g., classP, elementP, and bondtypeP. Other domains (mol, atom, and bond) are considered as individuals. Each line defines one predicate, listing the predicate name, its arity, the domain of each argument, and the number of times this predicate can be used (useful either to disable some predicates, or to limit loops with structural predicates – see below). Structural predicates require one more column: the kind of probability distribution that should be used (li, ms, ss). Non-determinate structural predicates are denoted with a star when several individuals (e.g. atoms) can be linked to a single individual (e.g. molecule). Comparing with Figure 1, we see that properties correspond to atomic components of tuples, while structural predicates correspond to non-atomic components of tuples (e.g. sets).

### 3.2 1BC2 classification and learning algorithms

In order to perform classification, 1BC2 is given a target predicate on the command line (e.g. class). We will now explain the classification and learning phases. For classification, the main function is `getClassLikelihood(individual i)`. It returns a table with an estimate of the class likelihoods of individual *i*. We will use *P* to denote probability estimates (Laplace estimates from counts collected in the learning phase).

**Algorithm 1 (1BC2 classification algorithm)** *To obtain the class likelihoods  $P(i|c)$  for an individual *i*:*

```

getClassLikelihood(i) {
FOR all properties prop of i
  Find parameter value v such that prop(i,v)
  FOR each class c
    CL[c] = CL[c] x P(v|c)
FOR all structural predicates struc involving i
  IF struc is functional given i THEN
    Find the related individual j
    CL' = getClassLikelihood(j)
    FOR each class c
      CL[c] = CL[c] x CL'[c]
  ELSE /* non-determinate struc. pred. li/ms/ss */
    Find all related individuals J
    FOR each individual j of J
      CL''[j] = getClassLikelihood(j)
    FOR each class c
      CL[c] = CL[c] x estimateP(li/ms/ss,CL'',c)
Return CL}

```

In order to deal with lists, multisets and sets, `estimateP(li/ms/ss,CL'',c)` applies the appropriate formula (Theorems 2.2, 2.3, 2.4) using `CL''[j][c]` as the probability of each element *j*.

*Example 3.1 (Classification of a molecule)* Here is a partial description of molecule d1:

```

lumo(d1,-1.246).
logp(d1,4.23).
mol2atom(d1,d1_1).
charge(d1_1,-0.117).
element(d1_1,c).
atomtype(d1_1,22).
fr_atom2bond(d1_1,d1_1d1_2).
fr_atom2bond(d1_1,d1_1d1_7).
...
to_atom2bond(d1_1,d1_6d1_1).
...
mol2atom(d1,d1_2).
...
bondtype(d1_1d1_2,7).
...

```

*In order to classify molecule d1, 1BC2 estimates its probability given each class using the properties of that individual, and the probabilities of individuals it is linked to. For molecules, we retrieve the probabilities of a LUMO value of -1.246 and a logP value of 4.23 (these are estimated during the training phase, see below). Next, we consider the structural predicate mol2atom. A molecule can have several atoms (\*:atom in the ISP declaration of mol2atom) and the  $P_i$  estimate (li at the end of the line) will be used. In order to apply  $P_i$ , 1BC2 requires the probability of each atom of molecule d1: d1\_1, d1\_2, ... given a class c. This is obtained by applying the algorithm recursively.*

*The probability of atom d1\_1 is estimated using its properties: element(d1\_1,c), atomtype(d1\_1,22), and charge(d1\_1,-0.117), and the probabilities of the individuals it is linked to: fr\_atom2bond(d1\_1,d1\_1d1\_2), to\_atom2bond(d1\_1,d1\_6d1\_1), fr\_atom2bond(d1\_1,d1\_1d1\_7), etc. First for each class we multiply the estimates of the probability of a carbon atom, the probability of an atomic number of 22, and the probability of a -0.117 charge for that class. Then we need to estimate the probabilities of each bond by applying the same algorithm again. The probability of bond d1\_1d1\_2 is estimated using its property: bondtype(d1\_1d1\_2,7). The same is done for all bonds in which atom d1 occurs. Then, given those probabilities, the  $P_i$  formula is used to get the probability of both sets of bonds, and  $P_w$  is used to obtain the probability of atom d1\_1 for each class. The same is done for all atoms of molecule d1. Finally, the  $P_i$  formula is used again to get the probability of the molecule given a list of its atoms.*

The flattened language used by 1BC2 allows more representational flexibility than a pure term language because subterms are named. This means that graph structures such as molecules can be modelled, and we need to take care to avoid cyclic computations. ISP declarations allow to limit the number of times a structural predicate is used (set to 1 in the above example). In addition, a global limit on the number of structural predicates can be set. In practice, this means that an individual is represented by the set of paths to its parts satisfying these length constraints.

The learning phase has to estimate the probabilities of empty lists or sets, and the probabilities of each property, for each class. This involves counting how many individuals satisfy each value of the property, taking into account the target predicate. For instance, 1BC2 counts how many carbon atoms belong to mutagenic molecules, and how many carbon atoms belong to non-mutagenic molecules. It does the same for all parameters (o, h, ...) of the element, and for all parameters of all other properties (atomic type, charge, type of bond). The Laplace estimate is

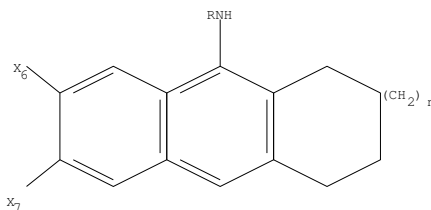


Figure 2: Template of the tacrine molecule.

used given those countings. The probability  $\tau$  of empty lists and sets is determined as follows.

**Lemma 3.3 (Estimating the probability of empty lists and sets)** *Assuming that the cardinalities of lists and sets are governed by a geometric distribution  $P(l) = \tau(1 - \tau)^l$ , the maximum likelihood estimate for  $\tau$  is  $\frac{1}{1 + \bar{l}}$ , where  $\bar{l}$  is the average of the observed cardinalities.*

*Proof.* Assuming the observed values  $l_i$  are identically and independently distributed, the maximum likelihood estimate of  $\tau$  is the value that maximises the probabilities  $p = \prod_i P(l_i) = \prod_i \tau(1 - \tau)^{l_i}$ . This is equivalent to maximising  $\log(p) = \sum_i (\log(\tau) + l_i \times \log(1 - \tau))$  or  $\frac{\log(p)}{k} = \log(\tau) + \bar{l} \times \log(1 - \tau)$ , where  $\bar{l}$  is the average of  $l_i$  and  $k$  is the number of lists. Taking the derivative and setting to 0 yields  $\frac{1}{\tau} - \bar{l} \times \frac{1}{1 - \tau} = 0$  and thus  $\tau = \frac{1}{1 + \bar{l}}$ .

### 3.3 Experimental results

We have obtained experimental results on several domains. Here we describe one of them, concerning a dataset involving drugs against Alzheimer’s disease [1]; further experimental results can be found in [11]. Such drugs can have the following four desirable properties:

- inhibit amine reuptake
- low toxicity
- high acetyl cholinesterase inhibition
- good reversal of scopolamine-induced memory deficiency

The aim is not to predict whether a molecule is good or bad, but rather which of a given pair of molecules is best for each of the four properties above. All molecules considered in this dataset have the structure of the tacrine molecule and they differ only by some substitutions on the ring structures (Figure 2).

New compounds are created by substituting chemicals for  $R$ ,  $X_6$  and  $X_7$ . The  $X$  substitution is represented by a substituent and its position. The  $R$  substitution consists of one or more alkyl groups linked to one or more benzene rings. The linkage can either be direct, or through  $N$  or  $O$  atoms, or through a  $CH$  bond. The  $R$  substitution is represented by its number of alkyl groups, a set of pairs of position and substituent, a number of ring substitutions, and a set of pairs of ring position and substituent.

Experiments have been carried out with the list distribution (Theorem 2.2). The multiset distribution (Theorem 2.3) has not been used since, in the context of the naive Bayes classifier, it gives the same results as the list distribution ( $P_{ms}(s) =$

Table 1: Accuracy in the Alzheimer’s disease domain. For 1BC and 1BC2 two numbers are given: the first corresponds to the default probability threshold of 0.5, and the second corresponds to the best point on the ROC curve (see text).

Target	1BC	1BC2	Best rule inducer
Inhibit amine reuptake	67.7% / 79.5%	75.3% / 79.0%	86.1%
Low toxicity	74.4% / 73.9%	73.8% / 73.5%	81.9%
High acetyl cholinesterase inhibition	69.1% / 70.4%	68.7% / 68.3%	75.5%
Reversal of memory deficiency	62.2% / 76.2%	76.6% / 76.9%	61.0%

$\frac{l!}{k_1! \dots k_n!} P_{li}(s)$  and  $\frac{l!}{k_1! \dots k_n!}$  does not depend on the class, so  $\operatorname{argmax}_c P_{ms}(s|c) = \operatorname{argmax}_c P_{li}(s|c)$ . The list distribution can be viewed as an appropriate approximation of the subset distribution (Theorem 2.5), which has been experimentally verified [11].

Table 1 compares the accuracies of 1BC and 1BC2 on each of the four desirable properties. All experiments in this domain are carried out using a 10-fold cross validation. The last column shows the best accuracies reported by Boström and Asker using up-to-date rule inducers [1]. It should be noted that Boström and Asker applied several rule inducers, and that none of them got the best accuracy on all four target. Therefore those accuracies are surely an overestimate of the best overall rule inducer.

Probabilistic classifications such as predicted by 1BC2 can be visualised in an interesting way using ROC curves. If we order the test instances from most certain to be positive to most certain to be negative, the predicted ordering is not so important as long as the actual positives come before the actual negatives. We can visualise this in a Cartesian square as follows: if the next instance is an actual positive make a step upwards, otherwise make a step to the right.<sup>3</sup> The resulting curve moves monotonically from (0,0) to (1,1) – the ideal curve is a step function through (1,0), meaning that all the positives come before the negatives in the ordering given by the classifier.

In order to turn a probabilistic classifier into a categorical one, the decision rule used is usually ‘if the predicted probability of being positive is larger than the predicted probability of being negative then predict positive else negative’. Equivalently, this corresponds to setting a fixed threshold 0.5 on the positive probability: if the positive probability is larger than this threshold we predict positive, else negative. In terms of ROC curves, this means fixing a particular point on the curve, yielding a particular true positive rate (on the y-axis) and false positive rate (on the x-axis). Our experiments show that this is often not the optimal choice: therefore we use the ROC curve to obtain the probability threshold with optimal classification accuracy (second row of numbers in Table 1). This almost always improves the obtained accuracy.

Figure 3 shows the corresponding ROC curves. Also indicated are: the points with probability threshold 0.5 corresponding to the accuracy results in Table 1 (the crosshairs), and the iso-accuracy lines through these points (the diagonal lines). Notice that for the fourth target the optimal point happens to be the majority class classifier – i.e., 1BC (with default 0.5 threshold) and all rule inducers performed

<sup>3</sup>In the case of ties, we make the appropriate number of steps up and to the right at once, drawing a diagonal line segment.

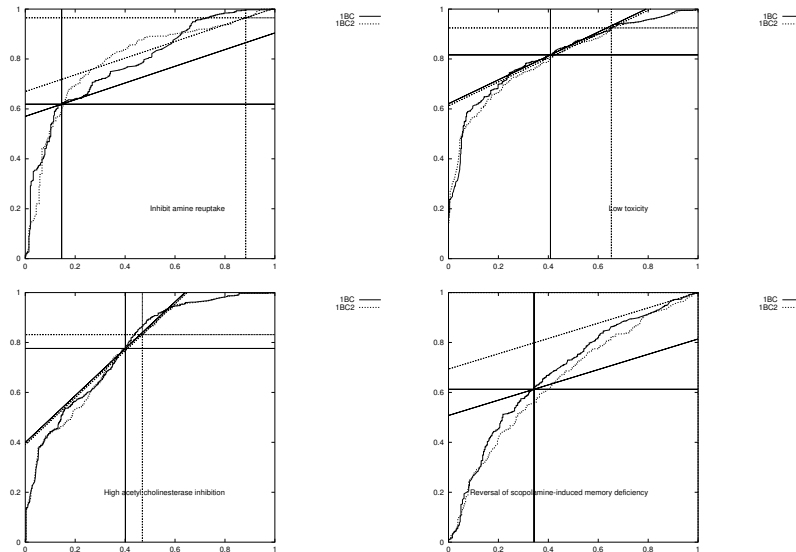


Figure 3: ROC curves in the Alzheimer's disease domain for 1BC and 1BC2. The crosshairs denote the point chosen by the default probability threshold of 0.5, and the diagonal lines indicate the iso-accuracy lines at those points (higher lines are better).

worse than default!

In this section we have shown how the probability distributions over terms defined in Section 2 can be used to implement a naive Bayes classifier. In the next section we consider the more general setting of Bayesian networks over terms.

## 4 Bayesian networks over terms

Bayesian Networks [15] are being used extensively for reasoning under uncertainty. Bayesian Networks are directed acyclic graphs, where the nodes represent propositional variables or attributes. The joint probability distribution encoded by a Bayesian Network can be obtained by multiplying the conditional probability distributions for each node given its parents. The graph structure thus encodes certain independence assumptions.

From the term perspective, Bayesian Networks define probability distributions over tuples of constants. In this section we consider how this can be upgraded to richer type structures. We consider the naive Bayes classifier as a case in point. It is well-known that the independence assumptions encoded in a propositional naive Bayes classifier can be represented as a Bayesian Network, with arcs from the class node to all the attribute nodes (and no other arcs). This can be equivalently modelled as a network consisting of two nodes, one representing the class and the other representing the tuple of conditionally independent attributes (Figure 4). Viewed this way, a Bayesian Network for terms becomes a merger of a type structure and probabilistic dependencies between subterms. By explicitly including the type structure we can see how probabilistic dependencies propagate over subtypes. In particular, by including a probabilistic dependency from Class to Desc we state that all components of Desc should be conditioned on Class; by not including any dependencies between A1, A2 and A3 we state that, thus con-

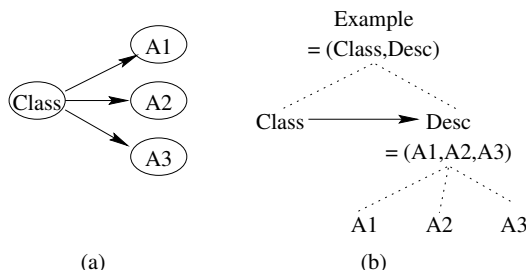


Figure 4: Dependencies in a simple naive Bayes classifier as (a) a standard BN (b) a BN over terms.

ditioned, these attributes are independent. As another example, the naive Bayes classifier for mutagenesis can be depicted as in Figure 5.

In the remainder of this section we consider Hierarchical Bayesian Networks (HBNs), an extension of Bayesian Networks, dealing with 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.

The outline of this section is as follows. In Section 4.1 we give an intuitive description of how HBNs upgrade Bayesian Networks. Section 4.2 gives the main definitions pertaining to HBNs. In Section 4.3 we discuss various inference algorithms for HBNs. In Section 4.4 we describe our implementation of a K2-style learning algorithm for HBNs.

#### 4.1 From Bayesian Networks to Hierarchical Bayesian Networks

A Bayesian network is a directed acyclic graph where nodes correspond to random variables and arcs between nodes represent probabilistic dependencies. Intuitively, an arc pointing from node  $A$  to node  $B$  can be perceived as  $A$  causing or influencing  $B$ . Each node in the network is annotated with a conditional probability table, that represents the conditional probability of the variable given the values of its parents in the graph. For nodes that have no parents, the corresponding table will simply contain the prior probabilities for that variable.

A Bayesian Network is a compact representation of the full joint probability of the random variables in the graph. The joint probability of  $n$  variables can be expressed using a table whose size is of order  $O(2^n)$ . In a Bayesian Network, the joint probability is expressed in factorised form, and conditional independencies are exploited in order to simplify the posterior probability expressions. Hence, it is enough to store  $n$  separate tables of size  $O(2^k)$ , where  $k$  is the maximum number of incoming arrows in a node.

The main property of Bayesian Networks can be loosely expressed in the following way: A node is conditionally independent of its non-descendants, given the values of its immediate parents. A strict definition of the independencies that a Bayesian Network encodes uses the criterion of d-separation [15].

Hierarchical Bayesian Networks are a generalisation of standard Bayesian Networks, where a node in the network may be an aggregate data type. This allows the random variables of the network to represent arbitrarily structured types. Within a single node, there may also be links between components, representing probabilistic dependencies among parts of the structure. Hierarchical Bayesian Networks encode conditional probability dependencies the same way as standard

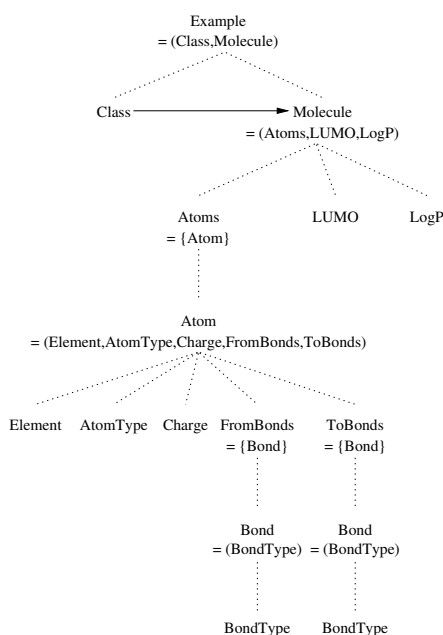


Figure 5: Bayesian Network for the mutagenesis naive Bayes classifier.

Bayesian Networks. Hierarchical Bayesian Networks can express further knowledge about variable structure and use that knowledge to build more realistic probabilistic models.

A Hierarchical Bayesian Network consists of two parts. The *structural part* contains the variables of the network and describes the *part-of* relationships and the *probabilistic dependencies* between them. The *part-of* relationships in a structural part may be illustrated either as nested nodes (Figure 6(a)) or as a tree hierarchy (Figure 6(b)). The second part of a Hierarchical Bayesian Network, the *probabilistic part*, contains the conditional probability tables that quantify the links introduced at the structural part.

We will demonstrate how a Hierarchical Bayesian Network can express dependencies in structured domains. Consider a random variable *PlayGolf* that expresses the probability that a particular day is appropriate for a given individual to exercise her hobby. That decision is independently influenced by the weather and by the individual's mood according to business affairs. Furthermore, assume that *Weather* is a triple of random variables (*Outlook*, *Temperature*, *Wind*) and *Business* is a pair of random variables (*Market*, *Meeting*) where *Market* consists itself of the pair (*Currency*, *Shares*). A hypothetical configuration of the probabilistic dependencies between those variables is illustrated as a Hierarchical Bayesian Network (structural part only shown) in Figure 7(a).

We can observe that the Hierarchical Bayesian Network structure is much more informative than a standard Bayesian Network mapping the same independencies, shown in Figure 7(b). Additionally, the assumption that components of *Weather* and *Business* are independent to each other is shown explicitly the structure. The structure can easily be extended (adding more components inside *Business* composite node) or refined (transforming a leaf node into a composite one) without having to examine dependencies with the separate *Weather* components (as would be the case in a standard Bayesian Network without any domain infor-

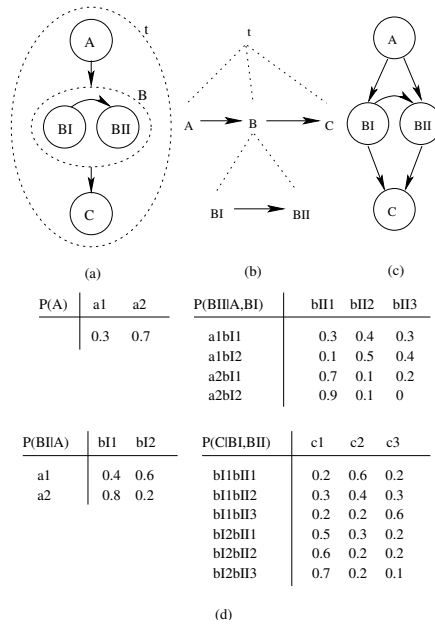


Figure 6: A simple Hierarchical Bayesian Network. (a) Nested representation of the network structure. (b) Tree representation of the network structure. (c) Standard Bayesian Network expressing the same dependencies. (d) Probabilistic part.

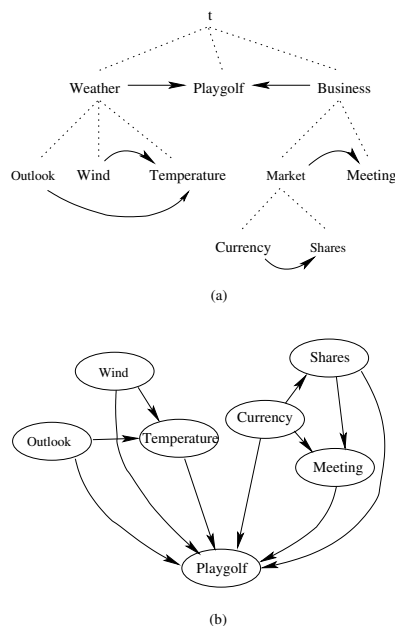


Figure 7: The Playgolf example structure. (a) Hierarchical Bayesian Network structure. (b) Standard Bayesian Network mapping the same independencies

mation).

## 4.2 Main definitions

We will now provide more formal definitions of the notion of Hierarchical Bayesian Networks. We begin by introducing hierarchical type aggregations, over which Hierarchical Bayesian Networks are defined. Currently, the only aggregation operator that we allow for composite types is the cartesian product, but we plan to extend composite types to include aggregations such as lists and sets, using the probability distributions such as the ones defined in Section 2.

**Definition 4.1 (Composite type)** Let  $\{\tau_1, \tau_2, \dots, \tau_n\}$  be a set of atomic types (domains).

The cartesian product

$$\tau = \tau_1 \times \tau_2 \times \dots \times \tau_n$$

is a composite type. The types  $\tau_1, \tau_2, \dots, \tau_n$  are called the component types of  $\tau$ .

**Definition 4.2 (Type structure)** The type structure corresponding to a type  $\tau$  is a tree  $t$  such that:

- If  $\tau$  is an atomic type,  $t$  is a single leaf labelled  $\tau$ .
- If  $\tau$  is composite,  $t$  has root  $\tau$  and as children the type structures that correspond to the components of  $\tau$ .

**Definition 4.3 (HBN-tree structure)** Let  $\tau$  be an atomic or composite type, and  $t$  its corresponding type structure. An HBN-tree structure  $T$  over the type structure  $t$ , is a triple  $\langle R, \mathcal{V}, \mathcal{E} \rangle$  where

- $R$  is the root of the structure, and corresponds to a random variable of type  $\tau$ .
- $\mathcal{V}$  is a set of HBN-tree structures called the t-children of  $R$ . If  $\tau$  is a simple type then this set is empty, otherwise it is the set of HBN-tree structures over the component-types of  $\tau$ .  $R$  is also called the t-parent of the elements of  $\mathcal{V}$ .
- $\mathcal{E} \subset \mathcal{V}^2$  is a set of directed edges between elements of  $\mathcal{V}$  such that the resulting graph contains no directed cycles. For  $(v, v') \in \mathcal{E}$  we say that  $v$  and  $v'$  participate in a p-relationship, or more specifically that  $v$  is a p-parent of  $v'$  and  $v'$  is a p-child of  $v$ .

If  $\tau$  is an atomic type, an HBN-tree structure over  $t$  will be called an HBN-variable. We will use the term HBN-variable to refer also to the random variable of type  $\tau$  that the root of the structure is associated to. Referring to the Hierarchical Bayesian Network of Figure 6, there are four HBN-variables:  $A, BI, BII$  and  $C$ . The t-parent of  $BII$  is  $B$ , and its only p-parent is  $BI$ .

Given an HBN-tree structure  $T = \langle R, \mathcal{V}, \mathcal{E} \rangle$  and a t-child of  $R$ ,  $\langle R', \mathcal{V}', \mathcal{E}' \rangle \in \mathcal{V}$ , then for any  $v_P, v_C, v_i$  such that  $(v_P, v) \in \mathcal{E}, (v, v_C) \in \mathcal{E}, v_i \in \mathcal{V}'$ , we say that  $v_P$  is an higher-level parent of  $v_i$ , and that  $v_i$  is an higher-level parent of  $v_C$ . Furthermore, if  $v_{HLP}$  is a higher-level parent of  $v$ , then  $v_{HLP}$  is also a higher-level parent of  $v_i$ , and if  $v$  is a higher-level parent of  $v_{HLC}$  then  $v_i$  is also a higher-level parent of  $v_{HLC}$ . Back to the example in Figure 6, the higher-level parents of  $BII$  are  $A$  and  $BI$ , and its only higher-level child is  $C$ .

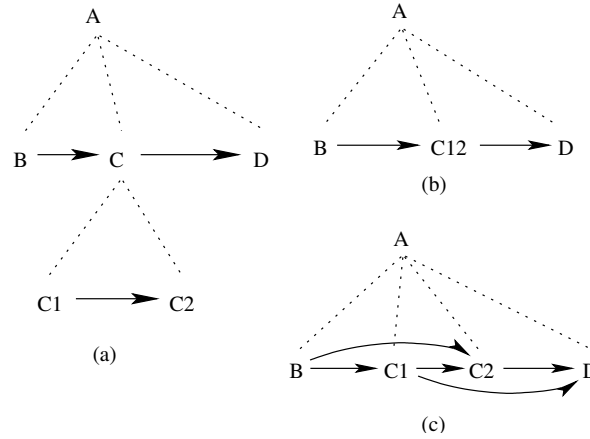


Figure 8: (a) A part of an HBN-tree structure. (b) Result of pruning the structure on node C. (c) Result of flattening the structure on C.

**Definition 4.4** The HBN-Probabilistic Part related to an HBN-structure  $T$  consists of:

- A probability table for each HBN-variable in  $T$  that does not have any  $p$ -parents or higher-level parents.
- A conditional probability table for each other HBN-variable, given the values of all HBN-variables that are its  $p$ -parents or higher-level parents.

**Definition 4.5** A Hierarchical Bayesian Network is a triple  $\langle T, \mathcal{P}, t \rangle$  where

- $t$  is a type structure
- $T = \langle R, \mathcal{V}, \mathcal{E} \rangle$  is an HBN-tree structure over  $t$
- $\mathcal{P}$  is the HBN-Probabilistic Part related to  $T$

The key property underlying a Hierarchical Bayesian Network is that the value of a variable is conditionally independent of the nodes that are not its descendants, given the values of its direct or higher-level parents.

We now define the operations of *pruning* and *flattening* a Hierarchical Bayesian Network on a given node  $v$  (Figure 8). Informally, pruning a structure on a composite node disregards the additional structure underneath, replacing the whole sub-tree with a single node, that will be an aggregation of all the atomic types contained in it. Flattening also disregards the structure, but retains the individual nodes (and the  $p$ -relationships between them). Using pruning and flattening we can employ various useful transformations on a Hierarchical Bayesian Network, such as deriving a standard Bayesian Network that maps the same probabilistic independencies between the atomic types.

**Definition 4.6 (Pruning)** Let  $t$  be a type structure with root  $\tau$  and  $T = \langle R, \mathcal{V}, \mathcal{E} \rangle$  an HBN-tree structure over  $t$ .  $P_t$  is a closed operator on type structures, defined as follows:

- If  $\tau$  is an atomic type,  $P_t(t) = t$ .
- Otherwise,  $P_t(t) = t'$ , where  $t'$  is a leaf type structure, corresponding to  $\tau$ .

That is, pruning a composite type simply yields the equivalent atomic type, ignoring the internal structure.

Assuming  $\mathcal{V}$  is not empty, let  $v \in \mathcal{V}$  be a  $t$ -child of  $R$ , i.e. an HBN-tree structure over  $t'$ , where  $t'$  is a child of  $t$ .  $P_h$  is a closed operator on HBN-tree structures, defined as follows:  $P_h(v)$  is an HBN-tree structure over  $P_t(t')$ , i.e. an HBN-variable.

The operation of pruning an HBN-tree structure  $T$  under a node  $v$ , which we notate  $P_{HBN}(T, v)$ , results in an HBN-tree structure  $T' = \langle R, \mathcal{V}', \mathcal{E}' \rangle$ , where  $P_h(v)$  replaces  $v$  in  $\mathcal{V}, \mathcal{E}$  to give  $\mathcal{V}', \mathcal{E}'$  respectively.

**Definition 4.7 (Type structure flattening)** Let  $t$  be a type structure with root  $\tau$  and  $t_i, i = 1, 2, \dots, n$  the children of  $\tau$ . The operation of flattening the type structure  $t$  under  $t_i$ , denoted  $F_t(t, t_i)$ , is defined as follows:

- If  $t_i$  is a leaf,  $F_t(t, t_i) = t$ .
- Otherwise, if  $t_i$  has children  $t'_j, j = 1, 2, \dots, m$ , and  $t'' = F_t(\dots(F_t(F_t(t_i, t'_1), t'_2), \dots), t'_m)$  (i.e.,  $t''$  is  $t'$  flattened under all its children), and  $t''_k, k = 1, 2, \dots, r$  are the children of  $t''$ , then  $F_t(t, t_i)$  is a type structure with root  $t$  and children

$$\{t_1, t_2, \dots, t_n\} \setminus \{t_i\} \cup \{t''_1, t''_2, \dots, t''_r\}$$

(i.e.,  $t_i$  is replaced by the children of  $t''$ ).

**Definition 4.8 (HBN-tree structure flattening)** Let  $T = \langle R, \mathcal{V}, \mathcal{E} \rangle$  be an HBN-tree structure over a type structure  $t$ ,  $T' = \langle R', \mathcal{V}', \mathcal{E}' \rangle$  an HBN-tree structure over a type structure  $t'$  with  $T' \in \mathcal{V}$ , and  $\tau, \tau'$  the roots of  $t, t'$  respectively. The operation of flattening the HBN-tree structure  $T$  under  $T'$ , denoted  $F_h(T, T')$ , is an HBN-tree structure over the type structure  $F_t(t, t')$  defined as follows:

- If  $T'$  is an HBN-variable,  $F_h(T, T') = T$ .
- Otherwise, if  $\mathcal{V}' = \{v'_1, v'_2, \dots, v'_n\}$ ,  $T'' = F_h(\dots(F_h(F_h(T', v'_1), v'_2), \dots), v'_n)$  is the result of flattening  $T$  under all its  $t$ -children and  $\mathcal{V}'' = \{v''_1, v''_2, \dots, v''_m\}$  are the  $t$ -children of  $T''$ , then  $F_h(T, T') = \langle R, \mathcal{V}_1, \mathcal{E}_1 \rangle$  such that
  - $\mathcal{V}_1 = \mathcal{V} \setminus \{T'\} \cup \mathcal{V}''$ , i.e.  $T'$  is replaced by the children of  $T''$  in the HBN-tree structure
  - $\mathcal{E}_1$  is similar to  $\mathcal{E}$ , except for each  $(T', v) \in \mathcal{E}$  being replaced by  $\{(v''_i, v) | i = 1, 2, \dots, m\}$  in  $\mathcal{E}_1$ , and each  $(v, T') \in \mathcal{E}$  being replaced by  $\{(v, v''_i) | i = 1, 2, \dots, m\}$  in  $\mathcal{E}_1$ . I.e., all  $v_i$  participate in the former  $p$ -relationships of  $T'$ .

**Definition 4.9 (Corresponding Bayesian Network)** Let  $H$  be a Hierarchical Bayesian Network structure with HBN-probabilistic part  $P$  associated to it, and  $H' = \langle R, \mathcal{V}, \mathcal{E} \rangle$  the Hierarchical Bayesian Network that results after flattening  $H$  successively under all its  $t$ -children. The corresponding Bayesian Network of  $H$  is a Bayesian Network whose graphical part is the graph  $\langle \mathcal{V}, \mathcal{E} \rangle$  and probabilistic part is  $P$ .

### 4.3 Inference in Hierarchical Bayesian Networks

Hierarchical Bayesian Networks provide a means for probabilistic inference on the variables they contain. The aim is, given a valuation for a subset of the variables (the evidence), to compute the probability of a second variable set (the query variables) having some specific values. Inference algorithms used for standard Bayesian Networks can also be applied to the Hierarchical Bayesian Network

model. We will investigate the applicability of both exact and approximate inference methods [16, 15]. Backward reasoning algorithms and message passing algorithms 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. Those 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 cycles 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.

The main problem with exact inference algorithms lies in the fact that they generally present exponential complexity to the size of the network. Although we illustrate applicability to the HBN case, we will focus more closely into approximate methods. Such a way of performing inference consists in sampling data that are generated stochastically using the network as a model. This approach is straightforwardly applicable on the HBN case, and experiments show that it is an efficient inference method.

### 4.3.1 Inference by message propagation

A 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 [15]. 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. The algorithm then repeats until the network reaches an equilibrium.

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.

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 [15, 8]. 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 when applied to a given Bayesian Network. As an extreme solution,

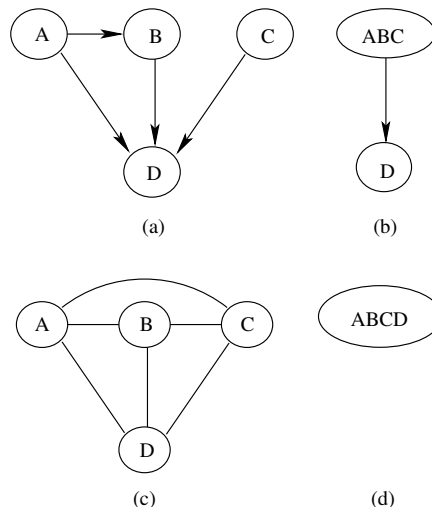


Figure 9: Coping with loops. (a) A Bayesian Network containing the loop ABDA. (b) Grouping all non-leaf variables in a single cluster. (c) Equivalent Markov network. (d) Join tree algorithm results in a single cluster.

all non-leaf nodes may be grouped in a single cluster (Figure 9(b)).

One popular method, described in [15], 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 (Figure 9(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 2 (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 retrieving its corresponding Bayesian Network, we arrive at a *polytree*

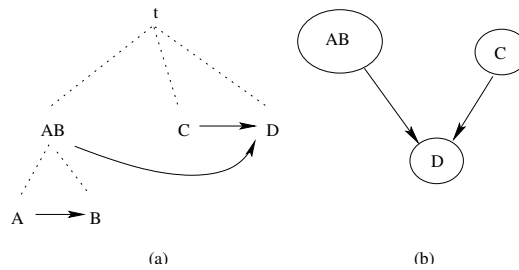


Figure 10: (a) A Hierarchical Bayesian Network containing the loop ABDA. (b) Result of the *HBN-decycling* algorithm.

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, i.e. the smaller cardinality of the variable domains.

In Figure 10(a) we see an HBN-tree structure containing a loop. The structure is similar to the Bayesian Network in Figure 9(a), with nodes *A* and *B* additionally forming a composite node. The result of the decycling algorithm (Figure 10(b)) retains much more structural information from the original structure.

#### 4.3.2 Inference by sampling

Another way of estimating the probability  $P(Q|E)$  for given valuations of sets of query and evidence variables  $Q$  and  $E$  is by sampling data from the Hierarchical Bayesian Network. Our approach is a straightforward extension of existing algorithms for standard Bayesian Networks, as described e.g. in [16, 15]. The HBN serves as a model that describes the joint probability of all the variables. In order to derive stochastically a value for one variable, all we need is the values of its higher-level parents. Since the graph resulting from higher-level relationships is always acyclic, there exist a total ordering of the variables such that every variable is preceded by its higher-level parents. We can then assign values to the variables in that order, according for each variable to the conditional probability given its higher-level parents, which will have already been given a value. Obviously, the first variables instantiated will be the ones that have no higher-level parents, and this will be according to their prior probability distribution. If we generate a large number of such instantiations, the relative frequency of the cases where  $Q$  holds divided by the relative frequency of the cases where  $E$  holds will converge to  $P(Q|E)$ .

One shortcoming of the above method is that when the likelihood of the evidence is small, convergence will be very slow. We address this problem using the method described as *logic sampling* in [16]: We associate each instance we generate to a weight, initially set to 1. When the instantiation procedure reaches an evidence node, instead of stochastically choosing a value, we deterministically assign to it the value it has as evidence, and multiply the weight for that run by the corresponding conditional probability of the node given the values of its parents (which will have already been instantiated). In order to compute the final relative frequency of the query being satisfied given the evidence, we take into account the final weight of each instance.

## 4.4 Learning Hierarchical Bayesian Networks

One significant topic we want to address is how we can construct a Hierarchical Bayesian Network (structural and probabilistic part) given a database of instantiations of the network's variables and the type structure that the variables are organised in. Our approach is an application of the method described in [2]. At first we use a Bayesian method to compute the likelihood of a structure given the data, and search for the structure that maximises that likelihood. We then use the relative frequencies of the events in the database to estimate the conditional probabilities for each variable given the values of its higher-level parents. If we assume an initial ordering for the nodes and that all possible HBN structures have equal prior probabilities, the derivation of the most likely structure is computationally feasible.

### 4.4.1 Computation of the maximum likelihood structure

Let us denote the given database as  $D$ , and let  $B_{HS}$  be an HBN-structure and  $B_S$  the corresponding Bayesian Network structure, both containing the same set of variables that appear in the database. In [2] a formula is derived to compute  $P(B_S, D)$ , depending on the prior  $P(B_S)$ . That result is based on the assumptions that (a) the variables in the database are discrete, (b) different instances occur independently given the structure, (c) there are no missing values, and (d) that before seeing the database, we consider all the possible conditional probability values setups for a given structure equally likely.

**Theorem 4.10 (Cooper and Herskovits)** *Let  $B_S$  be a Bayesian Network structure containing  $n$  discrete variables  $x_i$ , each associated to a domain  $(v_i1, \dots, v_{ir_i})$ , and  $\pi_i$  be the set of parents of  $x_i$  in  $B_S$ . Suppose  $D$  is a database of  $m$  instantiations of the variables  $x_i$ , and let  $(w_i1, \dots, w_{iq_i})$  be all the unique instantiations of  $\pi_i$  in  $D$ . Let  $N_{ijk}$  be the number of cases where  $x_i = v_{ik}$  and  $\pi_i$  is instantiated to  $w_{ij}$ . Let*

$$N_{ij} = \sum_{k=1}^{r_i} N_{ijk}$$

*The joint probability of having the structure  $B_S$  and the database  $D$  is given by:*

$$P(B_S, D) = P(B_S) \prod_{i=1}^n \prod_{j=1}^{q_i} \frac{(r_i - 1)!}{(N_{ij} + r_i - 1)!} \prod_{k=1}^{r_i} N_{ijk}!$$

To adapt this method to HBNs, we set  $P(B_{HS}, D) = \alpha P(B_S, D)$ , where  $S$  is the corresponding Bayesian Network of  $HS$  and  $\alpha$  is a normalising constant such that  $\sum_{HS} P(B_{HS}, D) = 1$ . Therefore, in order to compute the above joint probability, instead of parents (in the standard Bayesian Network case) we consider higher-level parents. The constant  $\alpha$  is needed because there are less possible HBN structures than standard BN structures containing the same variables, given the type structure. We also assume equal prior probabilities among different structures. So, in order to compute the likelihood for an HBN structure, we simply need to extract the corresponding BN structure from it and apply the above formula. The next step is to find the HBN structure that maximises that expression. Since probabilistic links occur only between siblings in the type structure, this presents a limit to the number of possible parents we need to consider for each node. The search space of the possible structures is significantly reduced compared to the standard Bayesian Network case. Furthermore, if we require an ordering on the nodes of the type structure (in fact, we only need an ordering for each subset of siblings) the

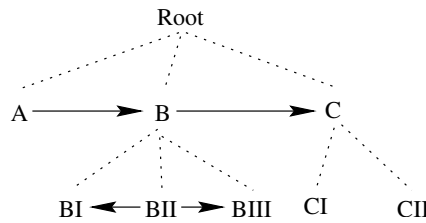


Figure 11: A Hierarchical Bayesian Network structure.

number of possible structures decreases dramatically. In [2] it is shown that with an ordering on the nodes and a sufficiently tight limit on the number of parents for each node, the derivation of the structure that maximises  $P(B_S, D)$  is computationally feasible. Our approach is a clear step towards that direction.

#### 4.4.2 Implementation and experiments

A software library has been implemented for generating HBNs, computing the results of probabilistic inference queries, and reconstructing an HBN structure given a set of variables, their relations in a type structure and a database of cases. For our experiments, we sampled the HBN in Figure 11 to obtain databases of various sizes. All the variables in consideration were boolean.

For the inference part, we implemented and tested the sampling methods for Hierarchical Bayesian Networks discussed earlier. Generation of samples (valuations for all the variables) from the HBN is done in  $O(n)$  time, so in general the value of a query  $P(Q|E)$  can be approximated efficiently, though more samples are needed for unlikely events. We queried the network of Figure 11 with several different query and evidence variable sets, and achieved convergence in all cases. In our implementation we did not consider exact inference methods, since the exponential nature of the problem would render their application infeasible even in moderate size domains.

To illustrate how learning can be achieved in Hierarchical Bayesian Networks, we applied the analysis presented in the previous section on an artificial example, which gives promising results for future tackling of real data. Our experiment consisted in defining an HBN which served as a model to generate a database of valuations for its variable nodes. Then, considering given the database and the type structure of the network, we seek the HBN structure that maximises the likelihood measure discussed in the above section. This was done by exhaustively searching through all possible HBN structures, and we considered both cases where a prior ordering on the nodes is and is not given. The algorithm was able to reconstruct the original structure with minor discrepancies, even when the database was relatively small. When the ordering of the nodes was taken into consideration, there was a dramatic improvement on the runtime required by the program, and the output was exactly the original structure. Even in a small example, we can note the speedup of the process compared to the standard Bayesian Network case. For this case where we have six variables, the number of possible BN structures is in the order of  $10^6$ , whereas the total number of possible HBN structures is 1875. If we introduce an ordering on the nodes, we have 32768 possibilities for BNs against 128 for the HBN case.

## 5 Discussion

We finish the paper with a brief review of some related work, and a summary of the main conclusions of our work.

### 5.1 Related research

We briefly discuss related work on the integration of probabilistic models with first-order logic.

#### 5.1.1 Bayesian Logic Programs

Bayesian Logic Programs [9] are a combination of Bayesian Networks and definite clause logic. A Bayesian Logic Program can be seen as having a logic part and a probabilistic part. The logic part is a (finite) set of *Bayesian definite clauses*, i.e. expressions of the form

$$A|A_1, A_2, \dots, A_n$$

Bayesian definite clauses are similar to Prolog clauses. The main differences are that atoms and predicates in BLPs are associated to a *domain*, and that the “|” symbol is used to signify probabilistic dependency. The probabilistic part includes a conditional probability density for each clause, defining the conditional probability for the values of the head given the values of the body variables, and a combining rule for each predicate that is defined in more than one clauses, that defines one combined conditional probability density. Intuitively, a BLP defines a Bayesian Network over the variables (ground atoms) contained in the least Herbrand model of the logic program. An edge from variable *A* to variable *B* is introduced whenever there is a refutation of *B* relying on *A*.

In contrast to Stochastic Logic Programs and Hierarchical Bayesian Networks, BLPs are defined to represent degrees of belief rather than statistical knowledge on the domain, and therefore match more closely to Halpern Type-2 representations.

#### 5.1.2 Probabilistic Relational Models

Probabilistic Relational Models (PRMs) [10] combine the expressiveness of Relational Models with the probability decomposition mechanisms defined on Bayesian networks. In a similar way as BNs define a probability distribution over possible valuations for the variables of the network, PRMs can be seen as defining a probability distribution over possible instantiations of a relational data base schema. The semantics of PRMs are very close to those of Bayesian Logic Programs, in that they represent a way for probabilistic reasoning over possible worlds. The main characteristic of this formalism is its close correspondence to relational logic.

#### 5.1.3 Hierarchical Probabilistic Diagnosis

[17] presents a mechanism for system fault diagnosis using nested Bayesian Networks. A *system model* is a set of components, each one having a number of inputs and outputs that link to other components of the system. For example, a digital circuit consists of a number of logical gates, where each gate input is connected to another’s output. A way of performing fault diagnosis in such a system is to construct a Bayesian Network where nodes and arcs correspond to components and their interconnections. Additionally, each component is accompanied by a discrete valued *state* node, stating whether the component is functioning properly. A fault in the system may then be tracked using some kind of Bayesian Network

diagnostic inference. Furthermore, components in a complex system may often themselves be built from sub-components. The BN representation of the model in this case will become more fine-grained.

There are several differences between this kind of representation and HBNs. On the technical level, our approach extends Bayesian Networks, and is a way of mapping the conditional independencies between variables in the network, rather than expressing operational dependencies as is the case in system models representation. Hierarchy is a part of the semantics of HBNs, and introduces a bias for reasoning and learning methods by restricting the total number of possible structures. In the case of system models, hierarchy rather emerges from the specific interpretation of a given model: given a system built from a set of components, every possible subset of these components could potentially be interpreted as a subsystem. From a more conceptual point of view, the obvious difference between the two works is that HBNs are a method for defining probability distributions over structured data, which is neither the intention nor a potential application of system models.

## 5.2 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 [5, 6] 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 allow the application of the model to structures of arbitrary form and length, such as web pages or DNA-sequences.

## References

- [1] H. Boström and L. Asker. Combining divide-and-conquer and separate-and-conquer for efficient and effective rule induction. In S. Džeroski and P. Flach, editors, *Proceedings of the 9th International Workshop on Inductive Logic Programming*, volume 1634 of *Lecture Notes in Artificial Intelligence*, pages 33–43. Springer-Verlag, 1999.
- [2] Gregory F. Cooper and Edward Herskovits. A bayesian method for the induction of probabilistic networks from data. *Machine Learning*, 9:309–347, 1992.

- [3] James Cussens. Parameter estimation in stochastic logic programs. *Machine Learning*, 43(3):245–271, September 2001.
- [4] P. Flach and N. Lachiche. 1BC: A first-order Bayesian classifier. In S. Džeroski and P. Flach, editors, *Proceedings of the 9th International Workshop on Inductive Logic Programming*, volume 1634 of *Lecture Notes in Artificial Intelligence*, pages 92–103. Springer-Verlag, 1999.
- [5] P.A. Flach, C. Giraud-Carrier, and J.W. Lloyd. Strongly typed inductive concept learning. In D. Page, editor, *Proceedings of the 8th International Conference on Inductive Logic Programming*, volume 1446 of *Lecture Notes in Artificial Intelligence*, pages 185–194. Springer-Verlag, 1998.
- [6] Peter A. Flach. Knowledge representation for inductive learning. In Anthony Hunter and Simon Parsons, editors, *Symbolic and Quantitative Approaches to Reasoning and Uncertainty (ECSQARU'99)*, volume 1638 of *Lecture Notes in Artificial Intelligence*, pages 160–167. Springer-Verlag, July 1999.
- [7] J. Halpern. An analysis of first-order logics of probability. *Artificial Intelligence*, 46:311–350, 1990.
- [8] M. Henrion. Propagating uncertainty by logic sampling in bayes' networks. Technical report, Department of Engineering and Public Policy, Carnegie-Mellon University, 1986.
- [9] Kristian Kersting and Luc De Raedt. Bayesian logic programs. Technical report, Institute for Computer Science, Machine Learning Lab, University of Freiburg, Germany, 2000.
- [10] Daphne Koller. Probabilistic relational models. In Sašo Džeroski and Peter A. Flach, editors, *Inductive Logic Programming, 9th International Workshop (ILP-99)*. Springer Verlag, 1999.
- [11] 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.
- [12] N. Lavrač, S. Džeroski, and M. Grobelnik. Learning nonrecursive definitions of relations with LINUS. In Y. Kodratoff, editor, *Proceedings of the 5th European Working Session on Learning*, volume 482 of *Lecture Notes in Artificial Intelligence*, pages 265–281. Springer-Verlag, 1991.
- [13] J.W. Lloyd. Programming in an integrated functional and logic language. *Journal of Functional and Logic Programming*, 1999(3), March 1999.
- [14] S. Muggleton. Stochastic logic programs. In L. De Raedt, editor, *Advances in Inductive Logic Programming*, pages 254–264. IOS Press, 1996.
- [15] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems — Networks of Plausible Inference*. Morgan Kaufmann, 1988.
- [16] Stuart J. Russell and Peter Norvig. *Artificial intelligence, a modern approach*. Prentice Hall, 2nd edition, 1995.
- [17] Sampath Srinivas. *Modeling Techniques and Algorithms for Probabilistic Model-Based Diagnosis and Repair*. PhD thesis, Stanford University, 1995.