Local Expression Languages for Probabilistic Dependence

Bruce D’Ambrosio

Department of Computer Science
Oregon State University

ABSTRACT

A Bayesian belief net is a factored representation for a joint probability distribution over a set of variables. This factoring is made possible by the conditional independence relationships among variables made evident in the sparseness of the graphical level of the net. There is, however, another source of factoring available which cannot be directly represented in this graphical structure. This source is the micro-structure within an individual marginal or conditional distribution. We present a representation capable of making this intra-distribution structure explicit, and an extension to the SPJ algorithm capable of utilizing this structural information to improve the efficiency of inference. We discuss the expressivity of the local expression language, and present early experimental results showing the efficacy of the approach.

1. INTRODUCTION

A Bayesian belief net [16] is a compact, localized representation of a probabilistic model. The key to its locality is that, given a graphical structure representing the dependencies (and, implicitly, conditional independencies) among a set of variables, the joint probability distribution over that set can be completely described by specifying the appropriate set of marginal and conditional distributions over the variables involved. When the graph is sparse, this will involve a much smaller set of numbers than the full joint. Equally important, the graphical structure can be used to guide processing to find efficient ways to evaluate queries against the model. For more details, see [16], [19], [5], [20], [14], [13], [15].

All is not as rosy at it might seem, though. The graphical level is not capable of representing all interesting structural information which might simplify representation or inference. The only mechanism available for describing antecedent interactions in typical general purpose Bayesian belief net inference implementations is the full conditional distribution across all antecedents. However, a number of restricted interaction models have been identified which have lower space and time

Address correspondence to 303 Daraborn Hall
Corvallis, OR 97331-2602
dambrosio@cs.orst.edu
(503) 737-5563

© 1994 Elsevier Science Inc.
655 Avenue of the Americas, New York, NY 10010 0888-613X/94/$7.00 1
complexity than the full conditional. The noisy-or [16], [17], [9], [12], [1] for example, can be used to model independent causes of an event, and inference complexity is linear in both space and time in the number of antecedents for many inferences. Similarly, contingencies [21], as well as the value-dependent independence exploited in similarity nets [11] are inefficiently represented using a full conditional. In this paper we present an extension to the standard Bayesian belief-net representation which is capable of explicitly capturing much of this lower level structural detail, and which permits use of these structures within arbitrary belief nets. We further present extensions to a factoring-based [15] inference algorithm in the SPI (Symbolic Probabilistic Inference) family which capture both the space and time advantages of these structures. In the remainder of this paper we first present an extension to the Bayesian belief net representation, and show how it can be used to capture the noisy-or and various asymmetries. We then present a very brief overview of SPI in its current form. While the representation is independent of SPI and could be used within any standard algorithm for inference in Bayesian belief nets, we find SPI the most natural framework within which to discuss the issues involved in inference with the extended representation. We then discuss the actual inference algorithm extensions we have made, and compare our local expression language to other recent efforts to generalize Bayesian nets. We close with some remaining questions.

2. LOCAL EXPRESSION LANGUAGES FOR PROBABILISTIC KNOWLEDGE

In this section we present an extension to the standard representation for belief nets. This extended expression language is useful for compact representation of a number of canonical interaction models among antecedents. We demonstrate its use in capturing the noisy-or as well as several other prototypical interaction structures. Later sections will show how one inference family, SPI, can be extended to make efficient use of the information in the extended representation.

The local expression (that is, the expression which describes numerically the dependence of the values a variable can take on the values of its antecedents) in a Bayesian net is simple: it is either a marginal or conditional probability distribution. While this representation is complete (that is, capable of expressing any coherent probability model), it suffers from both space and time complexity limitations: both the space and time (for inference) required are exponential in the number of antecedents. However, computation of child probabilities using the noisy-or interaction model is linear in the number of (independent) antecedents in both space and time. When evidence is available on child variables, computation of the posterior probability of parents is exponential in the number of positive pieces of evidence, but linear in the number of pieces of negative evidence.
Heckerman [9] has developed an algorithm, called Quickscore, which provides this efficiency for two level bipartite graphs. However, I am unaware of any implemented system other than the one reported here which can efficiently incorporate a noisy-or within an arbitrary Bayesian net. If the interaction between the effects of $A$ and $B$ on $D$ in the net shown in figure 1 can be modeled as a noisy-or interaction, then we might write the following expression for the dependence of $D$ on $A$ and $B$, following Pearl [16]:

$$
P(D = t) = 1 - (1 - c_A(D))(1 - c_B(D))
$$

$$
P(D = f) = (1 - c_A(D))(1 - c_B(D))
$$

Where $c_A(D)$ is the probability that $D$ is true given that $A$ is true and $B$ is false\(^1\). We use $c$ rather than $p$ to emphasize that these are not standard conditional probabilities. We will use a slightly more compact notation. We can define

$$
\hat{c}_A(D) = (1 - c_A(D))
$$

where

$$
\hat{c}_A(D) = \begin{cases} 
1 - c_A(D), & A = t \\
1, & A = f 
\end{cases}
$$

That is, where $c_A(D)$ is a single number ($P(D = t | A = t, B = f)$), $\hat{c}_A(D)$ is a pair of numbers expressing the dependence of $P(D)$ on $A$. Now we can reexpress the above as:

$$
P(D = t) = 1 - \hat{c}_A(D) \cdot \hat{c}_B(D)
$$

$$
P(D = f) = \hat{c}_A(D) \cdot \hat{c}_B(D)
$$

This notation is intuitively appealing. It is compact (linear in the number of antecedents), captures the structure of the interaction, and, as Heckerman has shown [10], can be manually manipulated to perform efficient inference. However, it is not sufficiently formal to permit automated inference. We define a formal syntax for our expression language are follows:

\begin{align*}
\text{exp} & \rightarrow \text{term} | (+ \text{term} \text{ term-set}) | \\
& \rightarrow (- \text{term} \text{ term-set}) | \\
& \rightarrow (* \text{term} \text{ term-set}) \\
\text{term} & \rightarrow \text{exp} | \text{distribution.} \\
\text{term-set} & \rightarrow \text{term} | \text{term} \text{ term-set.} \\
\text{distribution} & \rightarrow \text{name dimensions.} \\
\text{dimensions} & \rightarrow \text{conditioned} | \text{conditioned "\|" conditioning.} \\
\text{conditioned} & \rightarrow \text{variable-name domain} \\
& \rightarrow \text{variable-name domain conditioned.} \\
\text{conditioning} & \rightarrow \text{variable-name domain} \\
& \rightarrow \text{variable-name domain } | \text{conditioning.} \\
\text{domain} & \rightarrow \text{"\" } | \text{value } | \text{value-set.} \\
\text{value-set} & \rightarrow \text{value } | \text{value, value-set.}
\end{align*}

\(^1\)In the Mycin terminology, $P(D = t)$ is the probabilistic sum of $c_A(D)$ and $c_B(D)$. 
Notice that every term eventually must reduce to one or more distributions. Each distribution, in turn, is defined over some rectangular subspace of the cartesian product of domains of its conditioned and conditioning variables. Examining the simple noisy-or example provided earlier, we discover that the informal representation obscured the fact that the two instances of $\partial_A(D)$ are in fact operating over disjoint domains. In the remainder of this paper we will use the following notation to specify expressions and distributions:

$$
\exp(D) = 1_{D_t} - \partial_{D_t|A_{1,\ldots,J}} \partial_{D_t|B_{1,\ldots,J}} \\
+ \partial_{D_t|A_{1,\ldots,J}} \partial_{D_t|B_{1,\ldots,J}}
$$

Note that in this representation there are two instances of $\partial_A(D)$. While the numeric distributions are identical, the domains over which they are defined differ.

Having specified a syntax and shown that a noisy-or can be expressed in this syntax, we will next examine the semantics for the language, and whether or not these semantics match those standardly attributed to the noisy-or structural model. Expression semantics are quite simple to specify:

An expression is equivalent to the distribution obtained by evaluating it using the standard rules of algebra for each possible combination of antecedent values.

Performing this evaluation symbolically for our simple example yields:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>t</th>
<th>D</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
<td>t</td>
<td>1 - (1 - c_D(A)) * (1 - c_D(B))</td>
<td>(1 - c_D(A)) * (1 - c_D(B))</td>
<td></td>
</tr>
<tr>
<td>t</td>
<td>f</td>
<td>1 - (1 - c_D(A)) * (1)</td>
<td>(1 - c_D(A)) * (1)</td>
<td></td>
</tr>
<tr>
<td>f</td>
<td>t</td>
<td>1 - (1) * (1 - c_D(B))</td>
<td>(1) * (1 - c_D(B))</td>
<td></td>
</tr>
<tr>
<td>f</td>
<td>f</td>
<td>1 - (1) * (1)</td>
<td>(1) * (1)</td>
<td></td>
</tr>
</tbody>
</table>

This is, in fact, exactly the standard semantics attributed to noisy-or. The next question one might ask is whether all local expressions allowed by the syntax are semantically well formed, that is, correspond to coherent implicit full conditional distributions. Unfortunately, the answer is clearly no. In fact, coherent expressions can be composed of components which individually seem to violate coherence constraints (even as extreme constraints as the requirement that values lie in the range $[0,1]$). We shall see examples of this below, in considering the use of local expressions to represent additive value models. A student has developed an algorithm for coherence checking a local expression, a technical report on this topic is in preparation.

We next consider the use of the local expression language to represent several other common occurring intra-distribution structures.

2.0.0.1. Asymmetries It has been pointed out that probabilistic relationships are often asymmetric [8]. In the example presented by Geiger and Heckerman, a variable Badge (B) is dependent on a second variable, Hypothesis (H), and also dependent on a third variable, Gender (G), only when Hypothesis is either “worker” or “executive” (Hypothesis takes four values, “worker” (w), “executive” (e), “visitor” (v), and “spy” (s)). We can capture this structure as follows:

$$
\exp(Badge) = P_{B_{i,j}|B_{i,\ldots,J}} + P_{B_{i,j}|H_{i,\ldots,J}}
$$

---

2We ignore the actual numeric values in the distribution, since they are not germane to the discussion. Our actual syntax uses prefix notation; however, for readability we will use infix notation in the remainder of the paper.
Another form of asymmetry occurs when a variable has a contingent existence\textsuperscript{3}. For example, $W$ might only be defined when $X = t$, and might depend on the values of $Y$ and $Z$ \cite{21}:

$$\exp(W) = P_{W|f}|X,Y,Z|_{t,f} + P_{X|f}$$

The second term in the above expression may be unexpected. Remember, however, that in a normal Bayesian network we must be able to recover the joint across all variables by forming the product of all distributions. The extension for local expressions is that we must be able to recover the joint by forming the product of all local expressions. One easy way to ensure this is to require that every local expression be defined for every instantiation of its parent set. For each such instantiation, the local expression for a variable must then describe the distribution of the mass for that instantiation. The second term in the above expression records that the mass associated with certain instantiations of the parents (all those in which $X = f$) is not assigned to any value in the domain of the variable $W$. Nonetheless, this mass must be accounted for so that this expression can be properly combined with others to recover the full joint.

The Dagum and Galper \cite{4} recently noted the utility of additive decomposition of conditional distributions. Their model is easily expressed in our local expression language:

$$\exp(Y) = a_1 \times P_{Y|f}|X_1|_{t,f} + a_2 \times P_{Y|f}|X_2|_{t,f}$$

Two notes about this example. First, the local expression language does not restrict the domains in the above example to \{t, f\}; we use that domain merely for concreteness. Second, as Dagum and Galper point out, the decomposition can be into conditioning subsets rather than individual variables.

Finally, decision models can be represented as Bayesian nets, as noted by Cooper \cite{3}. Value structures are often factorable. Consider, for example, the classic drilling example \cite{18}. The overall utility is the value of the oil recovered, minus the cost of drilling and the cost of testing:

$$P(V) = P_{V|f}|Oil\-recovered,\-soaking,\-wet,\-dry|_{t,f} - P_{V|f}|Drill|_{t,f} - P_{V|f}|Test|_{t,f}$$

In this case it is interesting to look at sample numeric values in the various component distributions:

\begin{verbatim}
(distribution Dvalue/or (((value t f))
 ((oil-recovered soaking wet dry))
 (soaking) 1.0 0.0)
 (wet) 0.6 0.4)
 (dry) 0.26 .74))

(distribution Dvalue/drill (((value t f))
 ((drill y n))
 (y) .2 -.2)
\end{verbatim}

\textsuperscript{3}Proper evaluation of queries on Bayesian nets including contingent variables goes beyond the scope of this paper, for now we will merely demonstrate the expressivity of the representation.

\textsuperscript{4}I hope the interpretation for these simple distributions is self evident, given the previous discussion and the formatting provided. If it isn't just skip this example, not much will be lost. The various costs and values are scaled to keep utility in the range [0, 1.0] as described by Cooper.
There seems to be something very strange in the above, namely a negative value in a distribution. The explanation is simple: the three \textit{Dvalue}/\textit{<var> distributions jointly define the full conditional. The first is the base definition, and so each row adds to 1.0 as we expect. The second and third distributions are coded as \textit{modifiers}. In order to preserve the constraint that a row in the full conditional sums to 1.0, a row in a modifier must sum to 0.0, and the sum of all modifiers for any one mass element must be such that the element remains in the range [0,1] at all times.

As a final example, and one which we will carry through the section on query evaluation, consider the network shown in figure 1. We will model both multiple-antecedent relationships using a noisy-or structural model. The local expressions for \textit{D} and \textit{E} in this network are as follows:

\[
\begin{align*}
\exp(D) & = 1_{D_i} - c_{D_i|A_{i,j}} * c_{D_i|B_{i,j}} + c_{D_i|A_{i,j}} * c_{D_i|B_{i,j}} \\
\exp(E) & = 1_{E_i} - c_{E_i|B_{i,j}} * c_{E_i|C_{i,j}} + c_{E_i|B_{i,j}} * c_{E_i|C_{i,j}}
\end{align*}
\]

3. OVERVIEW OF SPI

In this section we briefly review the essential aspects of the SPI approach to inference in Bayesian nets. This will be needed later when we show how the method can be extended to perform inference over nets defined using our local expression language. For further details, see [5] or [20].

Computation of probabilities in a Bayesian net can be done quite straightforwardly, albeit somewhat inefficiently\textsuperscript{5}. I illustrate this process with a simple network, shown in figure 1. First, the marginal and conditional distributions associated with the graph:

\textsuperscript{5}We ignore evidence for purposes of this introduction. It introduces only minor complications, see [5], [20] for details.
\[ \begin{align*}
\exp(A) &= p_{A_i,j} \\
\exp(B) &= p_{B_i,j|A_i,j} \\
\exp(C) &= p_{C_i,j|A_i,j} \\
\exp(D) &= p_{D_i,j|B_i,j,C_i,j}
\end{align*} \]

Now suppose we wish to compute \( P(D) \). The procedure is quite simple. We gather all distributions relevant to \( P(D) \) (using d-separation, for example), compose them, and sum over all variables except \( D \):

\[ P(D) = \sum_{A,B,C} p_{D_i,j|B_i,j,C_i,j} \times p_{C_i,j|A_i,j} \times p_{B_i,j|A_i,j} \times p_{A_i,j} \]

The actual computation can be optimized somewhat by retaining each dimension only until we have combined with all terms in which the dimension appears (unless that dimension is a goal of the evaluation, in which case it must be retained throughout the computation). For example, we can sum over \( A \), since it is not needed in the final result, immediately after combining with \( p_{B_i,j|A_i,j} \) and \( p_{C_i,j|A_i,j} \). Conjunctive queries are handled automatically (we compute the set of relevant variables with respect to the set of variables in the query), and the existence of evidence simply means we must include more distributions in our computation (and normalize)\(^6\). SPI essentially follows this process, but can be viewed as a heuristic procedure for developing factorings which minimize the dimension of intermediate results. The factorizing is developed incrementally, and factorizing is intermixed with expression evaluation. In the next section we briefly review our current factorizing heuristic. A full discussion of the current heuristics for constructing factorings, their theoretical basis, and experimental evaluation of their efficacy appears in a companion article recently published [15].

3.1. The set-factorizing algorithm

We have developed an efficient heuristic algorithm, called set-factorizing, for finding good factorings for probability computation. We review it here because we will later extend it for networks in which variable dependence is described using arbitrary local expressions. In the following brief review we will treat each distribution as a subset of the \( n \) variables in the network (e.g., \( p_{Y|X1,X2} \) will be represented as \( \{X1,X2,Y\} \)). We will refer to each such subset as a factor, and use the following algorithm to combine these factors. Note also that intermediate results will not, in general, be true probability distributions, but will rather be generalized distributions, as defined in [20].

1. Partition the set of factors into independent (i.e., no shared variables) subsets if possible. Evaluate each subset separately using the following method, then combine the results:
2. Construct a factor set \( A \) which contains all factors to be chosen for the next combination (initially all the relevant net distributions). Each factor in set \( A \) is represented as a set of variables. Initialize a combination candidate set \( B \) empty.
3. Add all pairwise combinations of factors of the factor set \( A \) to \( B \) which are not already in set \( B \) in which one factor of the pair contains a variable which is a child or parent of a variable in the second factor and compute \( u = (x \cup y) \) and sum(\( u \)) of each pair, where \( x \) and \( y \) are factors in the set \( A \), sum(\( u \)) is the number of variables in \( u \) which can be summed over when the conformal product corresponding to combining the two factors is carried out. A variable

\(^6\)There is also a separate phase on arrival of evidence where we sweep all references to unobserved values of the evidence variable from all distributions, but we promised we would not discuss evidence.
can be summed out when it appears in neither the set of target variables nor any of the factors not in the current pair.

4. Choose elements from set B such that $C = \{u | u : \text{minimum}_B(\{|u|-\text{sum}(u)\})\}$, where $|u|$ is the size of $u$ excluding observed variables. If $|C| = 1$, $x$ and $y$ are the factors for the next combination; otherwise, choose elements from C such that $D = \{u | u : \text{maximum}_C(\{|x|+|y|\})\}$, $x, y \in u$. If $|D| = 1$, $x$ and $y$ are the factors for the next multiplication, otherwise, choose any one of D.

5. Generate a new factor by combining the pair chosen in the above steps. Modify the factor set $A$ by deleting the two factors in the chosen pair from the factor set and adding the new factor in the set.

6. Delete any pair in set B which has non-empty intersection with the candidate pair.

7. Repeat step 3 to 6 until only one element is left in the factor set $A$ which is the final result.

Following is an example to illustrate the algorithm using the network shown in figure 1. Suppose that we want to compute the query $p(D)$ for the Bayesian network and assume that there are 2 possible values of each variable. The variables relevant to the query are $\{A, B, C, D\}$. We use the set-factoring algorithm to combine the distributions. We will omit domain subscripts on variables for simplicity.

1. Loop1:
   (a) Step 2: factor set A is $\{p_A, p_B|A, p_C|A, p_D|B, C\}$
   (c) Step 4: the current combination is $(p_A, p_B|A)$, (there was more than one candidate in this step, we chose one arbitrarily). The result is $p_{A,B}$.
   (d) Step 5: the set A is $\{p_A, B, p_C|A, p_D|B, C\}$
   (e) Step 6: set B is $\{p_C|A, p_D|B, C\}$.

2. Loop2:
   (a) Step 2: factor set A is $\{p_A, B, p_C|A, p_D|B, C\}$.
   (b) Step 3: the set B is $\{(p_A, B, p_C|A)(p_A, B, p_D|B, C)p_C|A, p_D|B, C)\}$
   (c) Step 4: the current combination is $(p_A, B, p_C|A)$. The result is $p_{B,C}$.
   (d) Step 5: the set A is $\{p_B, C, p_D|B, C\}$
   (e) Step 6: set B is empty.

3. Loop3:
   (a) Step 2: factor set A is $\{p_B, C, p_D|B, C\}$.
   (b) Step 3: The set B is $\{(p_B, C, p_D|B, C)\}$
   (c) Step 4: the current combination is $(p_B, C, p_D|B, C)$. The result is $p_D$.
   (d) Step 5: $\{p_D\}$
   (e) Step 6: the set B is empty.

The factoring result is

$$p(D) = \sum_{B,C}(p(D|B, C)(\sum_A(p(C|A)(p(B|A)p(A)))).$$

4. INFEERENCE WITH LOCAL EXPRESSIONS

Factoring heuristics for general networks containing arbitrary local expressions are an area of current research in our group. In this section we present a simple extension of the above heuristic
which works well for BN2O networks (bipartite graphs with a noisy-or interaction model). This
same heuristic has also performed well on a few general multilevel networks, where local expressions
are a mixture of noisy-or’s, additive decompositions, and asymmetric expressions, and has worked
well on a few general multilevel noisy-or networks. After a brief presentation of the basic algorithm,
we will discuss its operation and present some experimental results.

The algorithm presented above is designed to factor expressions which include only the conformal
product and marginalization operators, and takes an input with a single marginalization operator at
the outside of the expression. The extended algorithm will have to handle more general expressions,
but these will still be of a restricted form. Expressions formed by gathering all variable expressions
relevant to a query will still be a marginalization of a conformal product, but the terms of the conformal
product will now be general expressions rather than marginal or conditional distributions. The
key new decision, then, is when and what to distribute over a + or − factor in the overall conformal
product. We integrate this decision into the factoring heuristic described above by expanding the
set of candidates generated, revising the interpretation of a candidate, and extending the scoring
heuristic. Each of these is detailed below.

4.0.0.2. Extended candidate generation In addition to the pairwise candidates described
erlier, we also generate the following combination candidates:

1. The combination of an expression $e_1$ with all expressions containing conditioning variables in
   $e_1$ as conditioned variables (ie, at the first stage of combination, combination with parents). If
   $e_1$ is a distribution, then this candidate is generated only if it does not contain all expressions
   in factor set $A$.

2. The transitive closure of the above, that is, the combination of an expression $e_1$ with all
   expressions containing as conditioned variables either conditioning variables of $e_1$ or their
   parents. If $e_1$ is a distribution, then this candidate is generated only if it does not contain all
   expressions in factor set $A$.

3. if a factor $e_1$ is a $+$ or $-$ expression, the combination of $e_1$ with all remaining factors (ie, all
   but $e_1$).

The basic factoring algorithm is a greedy algorithm. The inclusion of groups of expressions as
candidates circumvents some of the limitations of greedy methods. It can be thought of as adding
a layer of abstraction to the factoring construction process, although note that “higher level” (ie,
group) candidates must compete not only with each other, but also with simple pair-wise candidates.

4.0.0.3. Extended candidate interpretation Any pairwise candidate containing a $+$ or $-$
factor as its first factor, as well as any candidate generated by the three extensions above, is inter-
preted as calling for the application of distributivity, that is, the distribution of the remaining factors
in the combination over the first factor. When the selected candidate contains more than two factors
the factoring heuristic is recursively called on the candidate factor set. When a candidate calling
for application of distributivity is selected, all factors in the candidate other than the designated $+$
or $-$ factor are distributed over the designated factor, the factoring heuristic is called recursively on
the resulting factors, and the results are combined, numerically when possible\(^7\). That is:

$$\text{Eval}((t_{11} + t_{12}) * t_2 * t_3) = \text{Eval(Eval}(t_{11} + t_2 * t_3) + \text{Eval}(t_{12} * t_2 * t_3))$$

4.0.0.4. Candidate selection The heuristic scoring function described earlier is extended as follows:

1. If a candidate $C$ begins with a $+/ -$ expression, and there exists at least one expression $x$ in
   set $A$ and not in $C$ for which $T(C) \cap C(x) \neq \emptyset$ and $T(C) \setminus C(x) \neq \emptyset$, add the number of $+/ -$
factors in $C$ to its first stage heuristic value (remember that the first stage heuristic value is just $|\mathcal{T}(C)|$, and that lower is better). $\mathcal{T}(C)$ is the set of variables to be retained in the result of evaluating candidate $C$, and $C(C)$ is the set of conditioning variables in the result.

2. When choosing among candidates with equal first stage scores:
   
   (a) If both candidates start with distribution-type factors, use the original scoring rule.
   
   (b) If one candidate starts with a distribution-type expression, and the other starts with a
       $+/\sim$ expression, select the candidate starting with the distribution-type expression
   
   (c) If both candidates start with $+/\sim$ expression, select the one with fewer expressions. If
       both have the same number of expressions, select the one with the larger number of
       conditioning variables in the first expression.

The first criterion above biases against using distributivity because in our current implementation symbolic operations are considerably slower than purely numeric ones. The second continues this bias toward combining distributions first, and adds a heuristic for selecting which $+/\sim$ expression to distribute over first. The last heuristic in 2 above is important in BN2Q networks. As shown in the experimental results presented later, there is considerable structure which can be exploited, even in networks as richly connected as the QMR-DT basic disease network (4000 findings, 600 diseases, 40,000 disease-finding links). The order in which finding expressions are distributed over is crucial to the successful exploitation of structure.

4.1. Numeric evaluation of operations on distributions

We have already discussed the semantics of overall expressions. In general evaluation of an entire expression for each possible set of variable values would be quite inefficient. Rather, we use the following equivalent method of evaluating expressions:

* Conformal product. We use the same procedure as for standard SPI. When combining distributions defined over differing subspaces of the domain for a variable, only those values in the domain for which both distributions are defined need be considered. That is, distributions are implicitly extended with 0.0 in all values for which they are not defined for variables over which they are defined, and replicated for all values of any variable over which they are not defined. Thus, $1_{D_t}$ can be seen to specify the distribution $\{1.0, 0.0\}$ over $\{D = t, D = f\}$.

$+/\sim$ sum/difference. As before, distributions are extended with zeros for values in the domain over which they are not defined. However, for addition and subtraction we must include domain values over which either of the arguments is defined.

5. EVALUATION

5.1. Complexity

5.1.1. Factoring complexity. Complexity of the base factoring algorithm is $O(n^3)$ in variables in the network. Since we are only increasing the number of candidates by a constant, 2, complexity is not affected by that change. Similarly, making the factoring algorithm recursive does not change complexity since the number of factors is still reduced by one each time through the basic loop. For further analysis see [16].
5.1.2. Evaluation Complexity  
The algorithm reproduces the essential results of Quickscore when applied to two level bipartite (BN2O) graphs: numeric evaluation is linear in the number of antecedents, linear in the number of negative findings, and exponential in the number of positive findings. However, further analysis has revealed that is a worst-case result. Often, after distributing over a few expressions, the result can be partitioned into independent subexpressions. In the following we present some preliminary results obtained by applying the above algorithm to the QMR DT network.

5.2. Inference in bipartite graphs: QMR-DT

QMR DT, at the time of this report, is a BN2O network with over 900 diseases, 4000 findings, and 40,000 disease to finding arcs. The size of the network, together with the number of findings in a typical case, makes inference a daunting task. The disease-disease interaction is modeled using a noisy-or, which reduces inference complexity to linear in the number of negative findings, linear in the number of diseases, and exponential in the number of positive findings (Quickscore, [9]). However, a typical case can have up to 50 positive findings, and findings have an average of 10 antecedents (some have hundreds, and tend to be common findings). As a result, Heckerman’s original trials of quickscore resulted in inference times of one minute for 9 positive findings [9].

5.2.0.1. Local Expression representation of Noisy OR  
Remember that the local expression language representation we use for the noisy or is as follows. Given three variables, $A$, $B$, and $B$, with $A$ and $B$ “causes” for $D$, the expression for $P(D|A, B)$ is:

$$exp(D) = 1_D - c_{D_i|A_i, f} * c_{D_i|B_i, f} + c_{D_f|A_i, f} * c_{D_f|B_i, f}$$

SPI rewrites expressions for finding variables and immediate successors eliminating all references to unobserved values of the finding variable. As a result, the expression for a positive finding is reduced to:

$$exp(D) = 1_D - c_{D_i|A_i, f} * c_{D_i|B_i, f}$$

And that for a negative finding to:

$$exp(D) = c_{D_f|A_i, f} * c_{D_f|B_i, f}$$

5.2.1. Experiments  
We ran a series of experiments to evaluate the current SPI heuristics on QMR. We ignored negative findings, since they can be handled in linear time and distorted our measurements for small numbers of positive findings. We also ordered positive findings by the number of possible causes (fewer causes first). We did this because findings with fewer causes should be easier to handle (more opportunities for partitioning), and should also be more diagnostic. We then tested each case by querying for the posterior probability of the disease provided with the case, given the first positive finding, the first two positive findings, the first three, and so on, until a threshold of 2,000,000 multiplies was exceeded (We used a limit of 2,000,000 for an individual query and the count was reset to zero as each new piece of evidence was added.). As shown in the charts which follow, we then recorded the number of findings, the number of multiplications needed (shown in thousands), and the posterior probability. We chose a limit of 2,000,000 multiplies because this
corresponded to about five minutes of compute time on a Macintosh Quadra 700 in Common Lisp. We estimate that a straightforward C or Pascal implementation would be about twenty times as fast, and so this is roughly comparable to the one minute of Pascal computation described in the Quickscore paper. The results in the following section were obtained by running the test cases in the Sci-Am test case file. The first table shows the results of running Quickscore on the ten cases, and is consistent with the results obtained by Heckerman. The second table shows the results obtained using the algorithm presented earlier.  

5.2.2. Results

<table>
<thead>
<tr>
<th>Case</th>
<th>Pos Findings</th>
<th>Mults</th>
<th>$P(D^*)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>13</td>
<td>1518</td>
<td>.076</td>
</tr>
<tr>
<td>2</td>
<td>13</td>
<td>1262</td>
<td>.999</td>
</tr>
<tr>
<td>3</td>
<td>13</td>
<td>596</td>
<td>.282</td>
</tr>
<tr>
<td>4</td>
<td>9 (Done)</td>
<td>289</td>
<td>.285</td>
</tr>
<tr>
<td>5</td>
<td>8 (Done)</td>
<td>167</td>
<td>.992</td>
</tr>
<tr>
<td>6</td>
<td>12</td>
<td>1617</td>
<td>.136</td>
</tr>
<tr>
<td>7</td>
<td>12</td>
<td>1814</td>
<td>.054</td>
</tr>
<tr>
<td>8</td>
<td>9 (Done)</td>
<td>343</td>
<td>.707</td>
</tr>
<tr>
<td>9</td>
<td>12</td>
<td>727</td>
<td>.004</td>
</tr>
<tr>
<td>10</td>
<td>8 (Done)</td>
<td>191</td>
<td>.875</td>
</tr>
</tbody>
</table>

Sci Am using extended set-factoring

<table>
<thead>
<tr>
<th>Case</th>
<th>Pos Findings</th>
<th>Mults</th>
<th>$P(D^*)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>18</td>
<td>1571</td>
<td>.916</td>
</tr>
<tr>
<td>2</td>
<td>16 (Done)</td>
<td>148</td>
<td>1.000</td>
</tr>
<tr>
<td>3</td>
<td>15 (Done)</td>
<td>321</td>
<td>.330</td>
</tr>
<tr>
<td>4</td>
<td>9 (Done)</td>
<td>11</td>
<td>.285</td>
</tr>
<tr>
<td>5</td>
<td>8 (Done)</td>
<td>17</td>
<td>.992</td>
</tr>
<tr>
<td>6</td>
<td>15</td>
<td>1527</td>
<td>.111</td>
</tr>
<tr>
<td>7</td>
<td>14</td>
<td>1602</td>
<td>.166</td>
</tr>
<tr>
<td>8</td>
<td>9 (Done)</td>
<td>38</td>
<td>.707</td>
</tr>
<tr>
<td>9</td>
<td>17 (Done)</td>
<td>1252</td>
<td>.997</td>
</tr>
<tr>
<td>10</td>
<td>8 (Done)</td>
<td>36</td>
<td>.875</td>
</tr>
</tbody>
</table>

The second table seems to indicate that the method described here is able to find and exploit considerable structure in the QMR net. Several questions are raised by these results, however. A basic question is one of credit assignment. To what extent is the improvement the result of partitioning versus the distribution heuristic? The table below separates these effects by retaining the partitioning step, but choosing randomly which finding to distribute over at each step. As we see, partitioning following each distribution, but randomly choosing the finding to distribute over, typically only increases capacity by one more finding, and doesn’t result in processing of all finding for any additional cases. Adding the distribution heuristic increases the number of positive finding substantially for all cases except six and seven, and results in handling all finding for all cases except 1, 6 and 7 within the multiplication limit. For case 1, SPI handles an additional five positive findings beyond quickscore. For cases six and seven the picture is less positive: these represent “worst-case” situations for the method described here. Almost every positive finding has a very large number of antecedents, and so there is little structure to exploit.

*For an indepth study of this particular problem see [7].
**6. DISCUSSION**

The factoring algorithm we presented is far from optimal. It is, in fact, a straightforward application of a few basic symbolic algebra techniques to the computation of Bayesian probabilities. We are continuing to refine our formulation of efficient inference using local expressions and at the same time continue our search for effective inference heuristics. What we have attempted to show is two things: (1) that fairly simple algebraic representations can be used to capture structural details not currently recorded in Bayesian nets; and (2) that fairly simple computer algebra techniques can directly and efficiently utilize this structural information. The method we present, implemented as an extension to our factoring algorithm for probabilistic inference, provides a method for performing inference using standard interaction models such as noisy-or within arbitrary Bayesian networks.

Noisy-or is traditionally considered to be of restricted applicability since standard presentations restrict to the case where all variables take only two values. However, there are generalizations to the multi-valued case which require $(d - 1)$ or $(d - 1)^n$ parameters for each antecedent, where $d$ is the number of values a variable can take. For example, the local expression language presented here can represent the Noisy Max generalization of Srinivas [22]. Prior work with Intel provides one example of the efficiency of the methods presented. In this work we explored the application of Bayesian networks to diagnosis of problems in semiconductor fabrication. One network we constructed to interpret final test results was a BN2O network consisting of 95 (mostly 2-value) variables, 64 “symptoms,” 31 “causes,” and 155 arcs. With observations on all symptoms, and 27 out of the 64 observations “abnormal,” the system processes marginal queries on the cause variables in 5 to 65 seconds, depending on the variable queried. Profiling tools reveal the computational burden is well balanced: approximately 50% of the cpu time is spent in the symbolic heuristics, and the remainder in numeric computation. This might seem surprising: regardless of whether one measures by number of “causes” or number of abnormal “symptoms,” time complexity might be expected to be on the order of $O(3^n)$, and therefore the network should be intractable, even using Quickscore. However, this worst case only occurs when every parent is connected to every child, and the Intel network only contains 155 arcs. After a few symbolic distributions of parent distributions over evidence expressions, the query expression is partitionable into several independent subfactors, and so the computation stays tractable. We have had some success in applying the algorithm to multi-level

\[ P(D^4) \]

<table>
<thead>
<tr>
<th>Case</th>
<th>Pos Findings</th>
<th>Mults</th>
<th>P(D^4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>14</td>
<td>1094</td>
<td>0.17</td>
</tr>
<tr>
<td>2</td>
<td>14</td>
<td>1455</td>
<td>0.99</td>
</tr>
<tr>
<td>3</td>
<td>13</td>
<td>506</td>
<td>0.28</td>
</tr>
<tr>
<td>4</td>
<td>9 (Done)</td>
<td>274</td>
<td>0.285</td>
</tr>
<tr>
<td>5</td>
<td>8 (Done)</td>
<td>171</td>
<td>0.992</td>
</tr>
<tr>
<td>6</td>
<td>12</td>
<td>1045</td>
<td>0.136</td>
</tr>
<tr>
<td>7</td>
<td>12</td>
<td>1071</td>
<td>0.054</td>
</tr>
<tr>
<td>8</td>
<td>9 (Done)</td>
<td>295</td>
<td>0.707</td>
</tr>
<tr>
<td>9</td>
<td>14</td>
<td>1999</td>
<td>0.279</td>
</tr>
<tr>
<td>10</td>
<td>8 (Done)</td>
<td>160</td>
<td>0.875</td>
</tr>
</tbody>
</table>
nets as well, although results are difficult to evaluate since there are no gold standards against which we can compare.

There are at least two limitations we see with the local expression language proposed here. First, the syntax admits probabilistically incoherent expressions. It would be nice to have a language (like the graphical language of Bayesian nets for large scale structure) which admitted only probabilistically coherent representations. Second, we see potential utility for a "subset" operator. That is, it is often useful to specify that a probability is constant over some subrange of a conditioning or conditioned variable. For example, the probability of variable $B$ might have one value for a particular value of parent $A$, and a second value for all other values of $A$. Currently there is no way to make this structure explicit using our representation.

We began our exploration of probabilistic inference in the context of truth maintenance systems, and at that time used symbolic representation at the level of individual probability mass elements [6]. Later, motivated by efficiency concerns, we changed to a symbolic representation at the distribution level [20]. We now seem to have come full circle: the implementation described here again performs symbolic reasoning on elements as small as individual probabilities. The difference is that we now have a choice of representation grain-size, and can select the grain-size appropriate for the dependence model being described.

Finally, there is nothing unique to SPI that enables it to be extended in the ways presented here. Any algorithm for probabilistic inference based on Bayesian nets should be extendable in the ways we have discussed. What is unique is our approach to efficient inference as essentially an algebraic problem, as opposed to a graph theoretic one. We see no easy way to extend graph-theoretic perspectives, either representationally or inferentially, to include the rich variety of local dependency structures our algebraic language can capture. The view within SPI of inference as essentially a symbolic algebra factoring problem, however, is readily extensible, as we have shown.

7. RELATED WORK

Fung and Shadter [21], have proposed a general representation for contingencies and asymmetries. Their representation is more general than the one described here in one important way: it permits representation of asymmetries which induce cycles. For example, $B$ may depend on $C$ when $A = t$, but $C$ may depend on $B$ when $A = f$. The representation presented here can accommodate such dependency structures, albeit somewhat awkwardly. One must define two contingent versions of $B$ and $C$, giving them separate names (e.g., $B1$ and $B2$). It is then up to the user to remember to query all instances of $B$ and sum the results.

Geiger and Heckerman [8] have proposed multi-nets as a general representation for asymmetries. Their representation, however, has a significant limitation: it is not usable as a component in an arbitrary Bayesian net. In particular, they assume that the hypothesis variable of a multi-net is a root variable. This restriction does not exist in our representation. However, their representation offers the strong advantage that consistency can be easily established. As we noted earlier, checking the consistency of local expressions in our language is exponential in the number of parent variables. Andreasen [2] presents a convenient method for describing interaction models such as noisy or. Dagum and Galper [4] have proposed an additive decomposition of conditional dependence that is easily captured using local expressions. Srinivas [22] has developed a generalized noisy-or model which is compatible in spirit, but might require extension of the set of operators available in our local expression language. This, together with extension to capture influence diagrams, is work in progress.
8. CONCLUSION

Bayesian nets are a compact, intuitive representation for general probabilistic models, but suffer from inability to efficiently represent low level structural details such as asymmetries and noisy-or relationships. We have presented an extension to the belief net representation for probabilistic models capable of explicitly capturing this information, and shown how the SPI framework can be extended use this information to perform efficient inference. This permits explicit capture of low level structural details within an arbitrary belief net, and provides efficient processing of arbitrary marginal and conditional queries on the resulting belief net. This facility also provides for easy experimentation on new interaction models, since there is no need to write code to perform inference using the new model: one directly describes the interaction using a simple algebraic local expression language. The full expression language has been implemented and is in use on a variety of monitoring, diagnosis, assessment and control projects.

Acknowledgements. Thanks to Bob Fung of Prevision, Inc. and Peter Raulefs and Bob Culley of Intel for many useful discussions. Thanks to NSF (IRI88-21660, IRI91-00530), AFSR, and Intel for providing the support which made this work possible.

References


