Performing decision-theoretic inference in Bayesian network ensemble models

Michael Ashcroft

I. Introduction

Bayesian networks are a popular and powerful tool in artificial intelligence. In certain circumstances, rather than using a single network, 'meta-models' are constructed and inference proceeds using a (fitness-)weighted average of the individual networks.

Individual networks can be extended for use in decision theoretic applications. The point of this paper is to explain how to generalize a common decision theoretic inference algorithms to work on such a meta-model. We will proceed by giving a mathematical overview of what Bayesian networks are and a brief explanation of how they are learnt and the situations that lead to the use of meta-models. We then examine the workhorse of exact Bayesian inference algorithms, the Junction Tree algorithm and looking at how to generalize it for use in decision theoretic applications for single networks. Finally, we see how to generalize the algorithm further for its use in decision theoretic applications for meta-models.

The junction tree algorithm presented follows that given by [1]. The decision theoretic extension of the algorithm is a simplification of that given in [2]. The generalization to meta-models is original work.

II. Bayesian Networks

Recall from probability theory that two random variables, X and Y, are independent if and only if $P(X,Y) = P(X)P(Y)$. Analogously, X and Y are conditionally independent given a third random variable Z if and only if $P(X,Y|Z) = P(X|Z)P(Y|Z)$, which is equivalent to:

$$P(X|Z) = P(X|Y,Z) \tag{1}$$

Also recall that the chain rule for random variables says that for n random variables, $X_1, X_2, ...X_n$, defined on the same sample space S:

$$\begin{aligned} P(X_1, X_2, ...X_n) &= P(X_n|X_{n-1}, X_{n-2}, ...X_1) \\ &\quad P(X_{n-1}|X_{n-2}, ...X_1) \\ &\quad ...P(X_2|X_1)P(X_1) \end{aligned} \tag{2}$$

Imagine we have five random variables: $\{A, B, C, D, E\}$. From the chain rule, we know that:

$$\begin{aligned} P(A, B, C, D, E) &= P(E|A, B, C, D) \\ &\quad P(D|A, B, C)P(C|A, B) \\ &\quad P(B|A)P(A) \end{aligned} \tag{3}$$

We can represent these five conditional independencies by means of a directed acyclic graph (DAG) and a set of conditional distributions, where:

- Each random variable is mapped to a node of the DAG
- Each node has associated with it the conditional distribution for its variable
- Each node has incoming edges from the nodes associated with the variables on which the node's conditional distribution is conditional

Such a representation is a Bayesian network. It satisfies the Markov conditions:

Definition A direct acyclic graph (DAG), G, with nodes N_G, a joint probability distribution, P, of random variables D_P, and a bijective mapping $f : D_P \Rightarrow N_G$ satisfies the Markov Condition if and only if for all $v \in D_P$, where $n = f(v)$, v is conditionally independent given P of the variables that are mapped to the non-descendants of n given the variables that are mapped to the parents n.

Node	Conditional Independencies
A	-
B	C and E, given A
C	B, given A
D	A and E, given B and C
E	A, B and D, given C

TABLE I: Conditional independencies required of random variables the DAG in Figure 1 to be a Bayesian Network

If we know no more than the decomposition given to us by the chain rule in equation 3, the associated Bayesian network's DAG will be complete (since each variable is conditional on all those prior to it in the decomposition order). However, imagine that we know that certain conditional independencies exist as specified in table I. From the definition of conditional independence, we know that:

- $P(C|B,A) = P(C|A)$
- $P(D|C,B,A) = P(D|C,B)$
- $P(E|D,C,B,A) = P(E|C)$

Accordingly:

$$P(A,B,C,D,E) = P(E|C)P(D|C,B)P(C|A)P(B|A)P(A) \quad (4)$$

Whenever we simply the conditional distributions in virtue of a known conditional independence relation, we remove an edge on the DAG of our Bayesian network representation. In this case, the resulting network is given by figure 1.

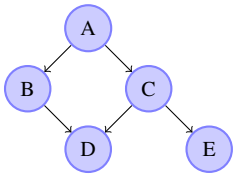

Fig. 1: A DAG with five nodes

Loosely speaking, what we have done is pull the joint probability distribution P apart by its conditional independencies. A Bayesian network is an encoding of these conditional independencies in the DAG topology coupled with the simplified conditional distributions. Note that the conditional independencies are encoded by the *absence* of edges.

III. Learning Bayesian Networks from Data

A Bayesian network can be specified from expert knowledge. Alternatively, we can learn the conditional independencies encoded in the network from data. The basic procedure is to perform a heuristic search on the space of possible sets of conditional independencies in order to obtain the best such set. This is complicated by the fact that multiple topologies can encode the same set of conditional independencies. To overcome this, we instead search equivalence classes of topologies/conditional independence sets [3].

In the discrete case, an algorithm exists that includes an optimality guarantee: As the size of our learning data approaches infinity, the probability of learning the globally optimal model (with a single iteration of the algorithm) approaches 1 [4]. Where the conditions are not met, a more general hill climb algorithm produces better results. The result is that the most robust learning algorithm utilizes the inclusion boundary algorithm for its first search and then repeatedly restarts the general hill climb algorithm.

The most principled and popular fitness function is the Bayesian Dirichlet score. This calculates the aposteriori probability of a set of conditional independencies given the learning data. Accordingly the network we obtain is that which represents the most probable set of conditional independencies.

IV. Meta-Models

Often a single network structure dominates alternatives. Where this is not the case, we can collect multiple high scoring networks by, for example, collecting all networks that are at least $\frac{1}{x}$ as probable as the best network, for some x. These networks can be weighted by their relative probability and inference can be performed over the entire set. Effectively, we now reason using not just our best hypothesis of the system structure, but a set of plausible hypotheses, weighted for their plausibility. This can be a very powerful method.

V. The Junction Tree Algorithm

This algorithm utilizes a secondary structure formed from the Bayesian Network called a Junction Tree or Join Tree. We first show how to create this structure.

Some Definitions:

- A cluster is a maximally connected sub-graph.
- The weight of a node is the number of values its associated random variable has.
- The weight of a cluster is the product of the weight of its constituent nodes.

Create (an Optimal) Junction Tree Algorithm

1) Take a copy, G, of the DAG, join all unconnected parents and undirect all edges.
2) While there are still nodes left in G:
   a) Select a node, n, from G, such that n causes the least number of edges to be added in step 2b, breaking ties by choosing the node which induces the cluster with the least weight.
   b) Form a cluster, C, from n and its neighbors, adding edges as required.
   c) If C is not a sub-graph of a previously stored cluster, store C as a clique.
   d) Remove n from G.
3) Create n trees, each consisting of a single stored clique. Also create a set, S. Repeat until $n-1$ sepsets have been inserted into the forest:
   a) Select from S the sepset, s, that has the largest number of variables in it, breaking ties by calculating the product of the number of values of the random variables in the

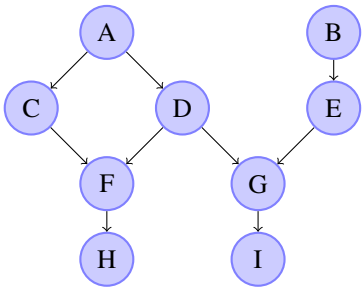

Fig. 2: A simple Bayesian Network

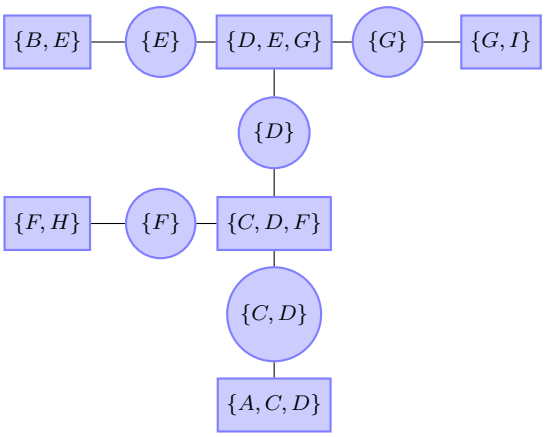

Fig. 3: The Junction Tree constructed from Figure 2

sets, and choosing the set with the lowest. Further ties can be broken arbitrarily.

b) Delete s from S.

c) Insert s between the cliques X and Y only if X and Y are on different trees in the forest. (This merges their two trees into a larger tree, until you are left with a single tree: The Junction Tree.)

Before explaining how to perform inference using a Junction Tree, we define a potential over a set of variables X is a function that maps each instantiation x into a real number. Each clique and sepset in the junction tree has a potential associated with it.

We define operations on potentials in the following way. Let the variables of potential p_1 are a superset of the variables of p_2. Let f be a function $p_2 \Rightarrow p_1$ such that it assigns each instantiation of the variables of p_2 the unique instantiation of the variables of p_1 where the shared variables take the same value. Likewise let g be a function $p_1 \Rightarrow p_2$ such that it assigns each instantiation of the variables of p_1 the set of instantiations of the variables of p_2 where the shared variables take the same value. Since we are dealing with discrete cases, we will talk of each instantiation being a row.

- Multiplying p_1 into p_2: Multiply each row, r, in p_2 by $f(r)$.
- Marginalize out of p_2 into p_1: Assign each row, r, in p_1 the sum of all rows in $g(r)$.
- Adding p_1 into p_2: Add to each row, r, in p_2 the value of $f(r)$.
- Maximize the variables not present in p_1 out of p_2 into p_1: Assign to each row, r, in p_1 the maximum value of $g(r)$.

Division is defined as expected given multiplication, except that we define the division of a rows value by zero to be zero. If you are unfamiliar, an more detailed explanation of potentials and operations on them is available at [1].

An evidence potential has a singleton set of random variables, and maps real numbers to the random variable's values. If working with hard evidence, it will map 0 to values which evidence has ruled out, and 1 to all other values (where at least one value must be mapped to 1). Where all values are mapped to 1, nothing is know about the random variables. Where all values except one are mapped to 1, it is known that the random variable takes the specified value. If working with soft evidence, values can be mapped to any non-negative real number, but the sum of these must be non-zero. Such a potential assigns values probabilities as specified by the its normalization.

We now require some definitions:

Message Pass

We pass a message from one clique, c_1, to another, c_2, via the intervening sepset, s, by:

1) Save the potential associated with s.
2) Marginalize a new potential for s, containing only those variables in s, out of c_1.
3) Assign a new potential to c_2, such that $pot(c_2)_{new} = pot(c_2)_{old}(\frac{pot(s)_{new}}{pot(s)_{old}})$.

Collect Evidence

When called on a clique, c, Collect Evidence does the following:

1) Marks c.
2) Calls Collect Evidence recursively on the unmarked neighbors of c, if any.
3) Passes a message from c to the clique that called collect evidence, if any.

Disperse Evidence

Variable	Value 1	Value 2	Value 3	Notes
A	1	1	1	Nothing known
B	1	0	0	Observed to be value 1.
C	1	0	1	Observed to *not* be value 2
D	0.75	0.2	0.05	Soft evidence, with actual probabilities
E	150	40	10	Soft evidence, assigns same probabilities as D

TABLE II: Evidence Potentials

When called on a clique, c, Disperse Evidence does the following:

1) Marks c.
2) Passes a message to each of the unmarked neighbors of c, if any.
3) Calls Disperse Evidence recursively on the unmarked neighbors of c, if any.

After creating the Junction Tree, we must initialize it. We can then perform inference on it.

Initialize Junction Tree Algorithm

1) Associate with each clique and sepset a probability potential, whose random variables are those of the clique/subset, and which associates with all value combinations of these random variables the value 1.
2) Associate with each variable an evidence potential representing current knowledge.
3) For each variable:
   a) Assign the variable a clique, c, containing the variable and its parents in the original Bayesian network (it is certain to exist).
   b) Multiply in the node's conditional probability distribution from the Bayesian network to the probability potential associated with c. (By 'multiply in' is meant: multiply the node's conditional probability distribution and the clique's probability potential, and replace the clique's probability potential with the result.)

Junction Tree Perform Inference Algorithm

1) For each variable:
   a) Set its associated evidence potential to represent current knowledge.
   b) Multiply this evidence potential into the probability potential of the variable's assigned clique.
2) Pick an arbitrary root clique, and call collect evidence and then disperse evidence on this clique:
3) For each node you wish to obtain *a posteriori* probabilities for:
   a) Select the smallest clique containing this node.
   b) Create a copy of the potential associated with this clique.
   c) Marginalize all other nodes out of the clique.
   d) Normalize the resulting potential. This is the random variable's *a posteriori* probability distribution.

Note that the complexity of the algorithm is dominated by the largest potential associated a clique. Note also that a junction tree can be formed from the smallest sub-graph containing the variables whose *a posteriori* probabilities we wish to find that is 'd-separated' (see [5] for a detailed definition of d-separation) from the remainder of the network by variables whose values we know. We will term this 'network pruning'. It is a common efficiency enhancement and, in complex networks, can be necessary for tractability.

VI. The Decision Theoretic Junction Tree Algorithm

When generalizing Bayesian networks for decision theoretic purposes, we introduce three new types of variables. Firstly, utility variables which specify the value to the user of the system being in particular states. Secondly, decision variables which are under the user's control. Finally, information variables which are variables not under the user's control that, if they are not currently known, will be known before the performance of a particular set of decisions. The last are no more than ordinary chance variables which have a particular relation with decision variables, but treating them as a specific type of variable eases explanation. We will term an information node that is known before a decision d an information parent of d.

Extending the junction tree algorithm decision-theoretically requires the stipulation of an information order, which is a partial order which we will refer to as the n-tuple I of sets of variables. We will talk of appending a set of variables to I. Where I is a tuple of n places, this means adding a new set of variables to I in place $n+1$ such that I is now a $n+1$-tuple. I is generated from a specification of an order in which the decisions must be made as well as the specification of information variables, such that:

Information Order Generation Algorithm

1) Specify a decision order. This should be a total order. In general, the actual order the decisions must be made in will be partial, but an arbitrary linear extension will transform it to a total order.
2) Make a copy of the domain (the set of decision, chance and information variables - not utilities), $\mathfrak{D}$.
3) For each decision node, d, in the order specified in the preceding step:
   a) If there are information parents of d, append these to I and remove them from $\mathfrak{D}$.

b) If I is empty or the highest ordered variable set in I contains non-decision variables, append the singleton $\{d\}$ to I.
c) Otherwise the highest ordered variable set in I contains decision variables, and we add d to that set.
d) Remove d from $\mathfrak{D}$.

4) Append $\mathfrak{D}$ to I.

I will be an n-tuple where, assuming the existence of at least one decision, the set of variables in places 1 to $n-1$ will be alternating sets of information parents and decision variables, and the set in place n will be the remaining chance nodes.

We now make the following emendations to the algorithms of the previous section. First, in the Create a Junction Tree algorithm, we replace steps 1 and 2(a) with:

1′ Take a copy, G, of the DAG except for utility nodes, join all unconnected parents including those of the utility nodes, and undirect all edges.

2 (a)′ Select a node, n, from G, such that:
(i) n is a lowest ordered remaining nodes, as given by the information order.
(ii) n causes the least number of edges to be added in step 2b, breaking ties by choosing the node which induces the cluster with the least weight.

One consequence of this is that the collect evidence portion of the algorithm now possesses an elimination order that respects the information order.

Secondly, we supplement the Initialize Junction Tree algorithm with two additional steps, the first between steps (1) and (2), and the second at the end:

1.5 Associate with each clique and sepset a utility potential, whose random variables are those of the clique/subset, and which associates with all value combinations of these random variables the value 0.

4 For each utility node, u, in the Bayesian network, multiply in the node's utility potential to the utility potential associated the clique containing all parents of u (it is certain to exist).

We also redefine a message pass, separating it into two cases:

Marginal Message Pass

We pass a message from one clique, c_1, to another, c_2, via the intervening sepset, s, by:

1) Save the probability potential associated with s.
2) Marginalize a new probability potential for s, containing only those variables in s, out of c_1.
3) Let U_s, U_c, P_s and P_c be the utility and probability potentials of s and c_1 respectively. Let X^y be the yth row of potential X. Assign values to U_s such that for each row, r: $U_s^r = P_s^r(\sum_{s\in g(r)} \frac{U_c^s}{P_c^s})$.
4) Assign a new probability potential to c_2, such that $pot(c_2)_{new} = pot(c_2)_{old}(\frac{pot(s)_{new}}{pot(s)_{old}})$.
5) Add the utility potential of s to the utility potential of c_2.

Maximal Message Pass

As for a marginal message pass, except that we replace step 2 with:

2. Create a new probability potential for s by maximizing out of c_1 those variables not present in s.

We must also redefine collect and disperse evidence:

Collect Evidence′

When called on a clique, c, Collect Evidence does the following:

1) Marks c.
2) Calls Collect Evidence recursively on the unmarked neighbors of c, if any.
3) Specify a target sepset, s, which is that which separates c from the clique that called collect evidence if such a clique exists, and $\emptyset$ otherwise.
4) Divides the clique into a number of subcliques, c_1 to c_n, where $c_1 = c$ and c_{n+1} lacks the lowest nodes present in c_n but not in s, as given by the information order.
5) Proceeds to pass messages from c_m to c_{m+1}, for $1 \leq m < n$, and from c_n to s, where a maximal message is passed if c_m/c_{m+1} (or c_n/s) consists of decision variables and a marginal message otherwise.
   - We collect the results of the maximization steps in order to obtain decision policies for the decision variables. A decision policy consists all variables in c_m, and takes rows take 1 when they correspond to the maximum value of the decision variables being maximized out, and 0 otherwise. Note that multiplying c_m by the resulting decision policy and then marginalizing out the decision variables results is equivalent to maximizing out the decision variables.

Disperse Evidence′

Disperse evidence is as before except that we now make marginal message passes.

Again we should note that pruning is possible and sometimes necessary. We now seek the smallest sub-graph containing the variables whose *a posteriori* probabilities we wish to find and the parents of all utility nodes that is d-separated from the remainder of the network by variables whose values we know. Also note that we impose more restrictions on the generation of the junction tree, which can result in larger cliques and hence, since complexity is dominated by the largest clique, increased complexity. The more decisions and information parents specified the more restrictions are imposed.

VII. Using Decision Theoretic Junction Trees with Meta-Models

We can now explain what alterations must be made to permit the above algorithms to function when using a meta-model composed of multiple networks, weighted by their relative fitness. Firstly, we focus on an issue that was only briefly noted above: Bayesian networks are pruned to obtain the smallest sub-network capable of generating a junction tree suitable for the case at hand. Since we wish for decisions to be made given the information available in the entire meta-model rather than particular networks two conditions must be met:

- If a decision node is present in junction tree, it must be present in all junction trees.
- If an information node is relevant to a decision variable in any network, we include it in all networks.

Accordingly, when creating multiple junctions trees for use with a meta-model, we must, in pruning the networks, follow an iterative algorithm that takes into account the variables that will be present in all junctions trees:

1) Create a set of nodes, Γ, consisting of the variables which are utility parents and any unknown variables we wish to predict the aposteriori distributions of.
2) Repeat until stability achieved:
   - For each network:
     a) Find the pruned sub-network, N, that will be required to generate the smallest junction tree capable of calculating the aposteriori distributions of the variables in Γ.
     b) Insert into Γ any decision variables present in N.
     c) Insert into Γ any information parent of any decision variable, d, in Γ that is present in N and not D-Separated from d.

Secondly, in formulating decision policies for decision variables (or, equivalently, when maximizing decision variables out during the collect information phase of the inference algorithm) we must incorporate the weighted information of all networks. To permit this, we require that all junction trees respect a total ordering on decision variables. Accordingly, we must alter 3 (b) and delete 3 (c) of the Information Order Generation algorithm, giving us:

3 b′ Append the singleton $\{d\}$ to I.

3 c′ -

Finally, we perform inference of the junction trees concurrently such that at each point when a decision variable is to be maximized out, the relevant potentials of all networks are weighted by their network weight and summed together to form a new potential. The result will be that the decision is made given the weighted information of *all* networks. To do so, we replace 2 in the Maximize Message Pass algorithm with:

2′. Create a new probability potential for s by maximizing out of c_1 those variables not present in s by:
   a) Wait for all junction trees to get to this step.
   b) Create a new potential over the decision variable, d, and all its information parents, with all rows taking the value 0.
   c) For the junction tree associated with each Bayesian network, n, (with c_1^n being the conditional probability table):
      i) Marginalize out a new potential over the decision variable and all its information parents, t^n, from c_1^n.
      ii) Multiply all values in t^n by the weight of network n.
      iii) Add t^n into d.
   d) Maximize the decision variable out of d to produce its decision policy.
   e) For the junction tree associated with each Bayesian network, n:
      i) Multiply c_1^d by d, creating a potential e.
      ii) Marginalize a new probability potential for s, containing those variables in e except for d, out of e.

Note that we ensure that all junction trees may have more (and will have at least as many) decision and information parents as would be the case were they being created for a single network model. This can result in the imposition of more restrictions on the generation of the junction tree, and hence larger cliques and greater complexity. It can also make pruning less effective. Interesting work remains to

be done on establishing fall back methods when such the algorithms given here are intractable. One possibility is to calculate all junction trees individually in the hope of finding decision policies that hold in all cases.

REFERENCES

[1] C. Huang and A. Darwiche, "Inference in belief networks: A procedural guide," *International Journal of Approximate Reasoning*, no. 11, pp. 1–158, 1994.

[2] F. V. Jensen and T. D. Nielsen, *Bayesian Networks and Decision Graphs*. Springer, 2nd ed., 2007.

[3] D. Chickering, "Learning equivalence classes of bayesian-network structures," *Journal of Machine Learning Research*, no. 2, pp. 445–498, 2002.

[4] D. Chickering, "Optimal structure identification with greedy search," *Journal of Machine Learning Research*, no. 3, pp. 507–554, 2002.

[5] R. Neopolitan, *Learning Bayesian Networks*. Pearson Prentice Hall, 2004.