On multiple spline approximations for Bayesian computations*

Eugene Santos, Jr.

Department of Electrical and Computer Engineering, Air Force Institute of Technology, Wright-Patterson AFB, OH 45433–7765,USA
E-mail: esantos@afit.af.mil

Probabilistic reasoning typically suffers from the explosive amount of information it must maintain. There are a variety of methods available for curbing this explosion. However, in doing so, it is important to avoid oversimplifying the given domain through injudicious use of assumptions such as independence. Multiple splining is an approach for compressing and approximating the probabilistic information. Instead of positing additional independence conditions, it attempts to identify patterns in the information. While the data explosion is multiplicative in nature, $O(n_1 n_2 \cdots n_k)$, multiple splines reduces it to an additive one, $O(n_1 + n_2 + \cdots + n_k)$. We consider how these splines can be found and used. Since splines exploit patterns in the data, we can also use them to help in filling in missing data. As it turns out, our splining method is quite general and may be applied to other domains besides probabilistic reasoning which can benefit from data compression.

1. Introduction

Managing large amounts of probabilistic information has been of particular interest to researchers modeling uncertainty. We can capture a lot of detail for any given domain by representing knowledge in terms of correlations. Combined with the strong semantics of probability theory, we can manipulate it in a sound and rigorous fashion. Unfortunately, as we often find, nearly everything cross-correlates in the strictest sense causing a tremendous data explosion.

We look in particular at a class of probabilistic models called *Bayesian networks* [6]. Other models such as influence diagrams [13], Markov random fields [3], similarity networks [4], and Markov networks [6] can also be considered since they are very closely related. However, for our purposes, Bayesian networks will be sufficient since our discussion will be readily applicable to the other models as we shall see.

* This paper was supported in part by AFOSR Project #940006 and by Rome Labs Project Number 55812769 USAF. Thanks to the anonymous reviewers who helped improve this paper.

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
1997		**00-00-1997 to 00-00-1997**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
On multiple spline approximations for Bayesian computations	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Air Force Institute of Technology,Department of Electrical and Computer Engineering,Wright-Patterson AFB,OH,45433-7765**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**	**Same as Report (SAR)**	**34**	

In Bayesian networks, objects and/or events are represented by random variables.[1] Correlations between these objects are modeled as conditional probabilities. In order to avoid the correlations explosion, simplifying assumptions concerning independence conditions are made about the target domain. In particular, conditional independencies are asserted between collections of r.v.s. Assume that each r.v. is designated by some unique node in a directed graph. Given a r.v. A, let W_1 be the immediate parents of A. For any set of r.v.s W_2 such that A is not an ancestor of any of those in W_2, $P(A \mid W_1, W_2) = P(A \mid W_1)$. Furthermore, such a graph must be acyclic. These networks are also called causal networks since we can interpret the conditionals as: If W_1, then A with probability $P(A \mid W_1)$.

To complete the formulation of Bayesian networks, each node has an associated table of conditional probabilities. For example, node A will have the table consisting of $P(A \mid W_1)$ for all possible assignments to A and W_1. The power of Bayesian networks lies in the fact that because of the above conditional independence structure, it is relatively straightforward to compute the joint probabilities of any complete assignment to all the r.v.s. We simply look up the corresponding probabilities in each of the conditional tables and multiply them together.[2] For example, consider the network in Fig. 1. The joint probability for the assignment $\{A = F, B = T, C = F, D = T, E = F\}$ is

$$\begin{aligned} &P(A = F, B = T, C = F, D = T, E = F) \\ &\quad = P(E = F \mid C = F) P(D = T \mid A = F, C = F) \\ &\qquad \times P(C = F \mid B = T) P(B = T) P(A = F), \end{aligned}$$

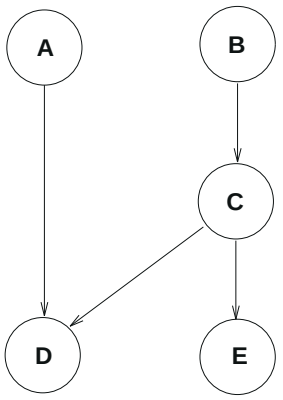
which computes to 0.21168.

The total number of conditional probabilities we must maintain in a Bayesian network is simply the sum of the number of entries in each table. The size of the table is governed by the number of r.v.s involved, the node and all its parents, and by the number of possible assignments to each r.v. Clearly, the table size is a multiplicative factor but nonetheless is an improvement over the completely cross-correlated case without independence assumptions.

Organizing r.v.s in an effort to curb the probabilities explosion has been applied to domains such as vision and pattern recognition. For example, hierarchically grouping sensors into vertical, horizontal, etc. line detectors reduces the overall connectivity of the graph. However, we must keep in mind that these tend to be overly-simplistic assumptions resulting in unrealistic solutions. Furthermore, connectivity is only one of the factors in table size. Independence does not impact on the number of possible assignments for a given r.v.

[1] We abbreviate "random variables" to "r.v." throughout the rest of this paper. We also assume that the r.v.s are discrete.

[2] For more detailed information, see Bayes' theorem in [6].




```
P(A) = .3

P(B) = .8

P(C|B = T) = .1
P(C|B = F) = .1

P(D|A = T, C = T) = .6
P(D|A = T, C = F) = .4
P(D|A = F, C = T) = .4
P(D|A = F, C = F) = .6

P(E|C = T) = .7
P(E|C = F) = .3
```

Fig. 1. Simple Bayesian network.

Consider the case where we have designated six sonars located around some mobile robot for use in determining its present location. Treat each sonar as a r.v. with assignments based on the possible different sonar readings. If we discretize our readings too much, we lose a great deal of accuracy. On the other hand, with only 10 values each, we still have a minimum of 10^6 possible combinations.

Aside from space considerations, how does this all impact our reasoning algorithms? Reasoning with Bayesian networks has been shown to be NP-hard [1, 2, 15] except for special classes such as polytress which are polynomial with respect to the number of nodes. Unfortunately, even for the polytrees, each node visit entails a complete search through the associated table [6, 18]. Hence, large tables clearly compound our computational problems.

Recently, alternative approaches to the table size problem have been developed [8, 12, 14]. Instead of relying on independence assumptions, a search is made for possible patterns in the conditional tables. Once some regular pattern is identified, we hope to encode it into some compact representation allowing us to dispense with explicitly storing the table.

One method is founded on something called *independence-based assignments* (abbrev. IBMAPs) [12, 14]. In this approach, we attempt to find a specific set of entries in the table which share the same probability. Let A be some r.v. and $B_1, \ldots, B_n, C$ be the immediate parents of A. If for some set of assignment values for $A, B_1, \ldots, B_n$, say $a, b_1, \ldots, b_n$,

$$P(A = a \mid B_1 = b_1, \ldots, B_n = b_n, C = c)$$

are identical for all possible assignments c to C, then we can conclude that under the assignments $a, b_1, \ldots, b_n$, any assignment to C will have no effect on the conditional probability. Hence, we can throw away all those redundant probabilities and replace them by the single probability

$$P(A = a \mid B_1 = b_1, \ldots, B_n = b_n).$$

Note, C can be generalized to a set of r.v.s as well.

Another method called "Noisy-OR" models [6, 7, 17] also exploits domain dependent information by attempting to identify disjunctive interactions (and its generalizations) between r.v.s. Once identified, we can reduce the amount of information necessary for computing the probabilities.

A third method is based on the idea of finding a real function called a *linear potential function* (abbrev. LPF) which can ideally interpolate through all the points in the table [8]. The arguments to the function would be the assignments to the r.v.s. Since assignments to r.v.s are often abstract objects or concepts, we must map them to real values. At the same time, the choice of mappings will directly affect the quality of the approximation. For example, consider the simple case where we have a table consisting of only one r.v., say A, representing the color of a ball. Let $\{\mathsf{red}, \mathsf{green}, \mathsf{yellow}, \mathsf{blue}, \mathsf{purple}\}$ be the possible colors and let the table be

$$\begin{aligned} P(A = \mathsf{red}) &= 0.02, \\ P(A = \mathsf{green}) &= 0.50, \\ P(A = \mathsf{yellow}) &= 0.00, \\ P(A = \mathsf{blue}) &= 0.37, \\ P(A = \mathsf{purple}) &= 0.11. \end{aligned}$$

First, let's simply map red to 1, green to 2, yellow to 3, etc. We get the resulting plot in Fig. 2. Now, let's choose a more intelligent mapping scheme as in Fig. 3. The approximation function for our second mapping is simply a straight line through all the points as opposed to the awful polynomial interpolation curve we would need for the first mapping. Thus, the problem also involves choosing the right mapping to help insure that we can find a simple enough function that can reasonably approximate the table. Once a suitable function is found, we can again throw away the table.

Clearly, the independence-based method is much simpler and easier to implement. In fact, we only need to make a very minor modification to the reasoning computations [14]. There is no loss of precision with this compaction scheme. However, requiring identical probabilities in the entries is very restrictive. Hence, this will generally result in relatively minor savings. With the "Noisy-OR" models, the problem lies in successfully identifying a significant number of interactions.

LPFs on the other hand can provide tremendous reductions. Furthermore, there is a closed form solution for computing the best one. However, we trade off these savings with reduced accuracy since LPFs are generally only an approximation.[3] Yet, with the LPF as well, little change to the reasoning algorithms is necessary. Much can be gained especially when searching for the optimal value in the table. As we mentioned above, most of the algorithms such as [18] must spend time in searching an unordered table. Since LPFs are real functions, we could potentially use simple techniques such as taking its first derivative to search for the optimal value even

[3] Although we can generally interpolate all the points perfectly given a high enough degree polynomial, such a function would be much too complex to find and use.

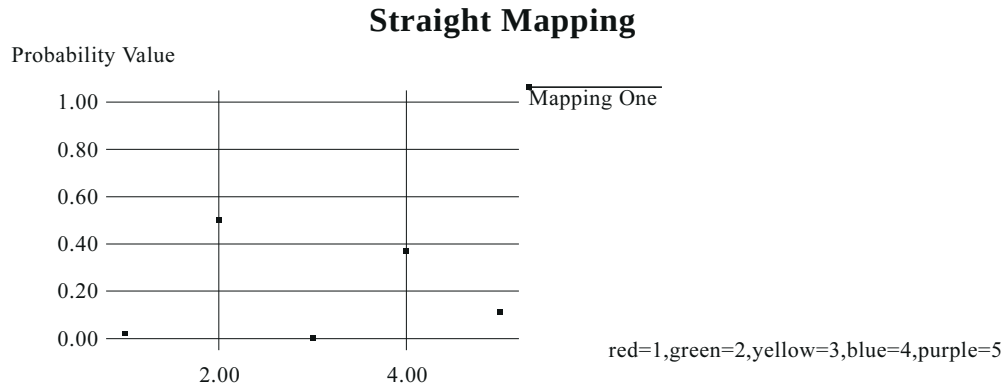


Fig. 2. Encoding 1 for A.

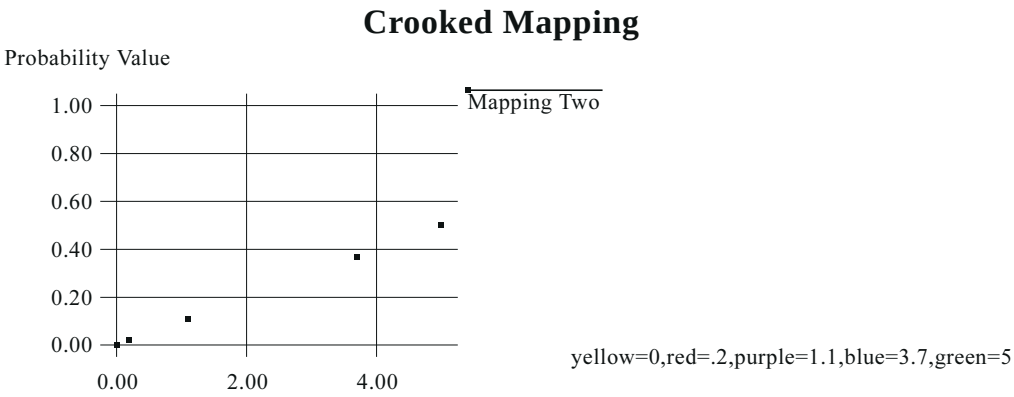


Fig. 3. Encoding 2 for A.

under various r.v. assignment restrictions. Furthermore, they can be directly used in an integer linear programming algorithm for reasoning with Bayesian networks [8].

Aside from the space and computational aspects of the methods above, the pattern matching approaches of IBMAPs and LPFs can also help deal with incomplete information. One of the drawbacks of using Bayesian networks is the requirement that the conditional tables be complete, i.e., all combinations of conditionals must be available. Lacking completeness, computations such as determining the most probable joint distribution cannot be performed. However, we might make intelligent estimates for these missing values via the patterns we've discovered using the above techniques.

In this paper, we look further at pattern matching techniques to help reduce the table explosions. In particular, we look more closely at LPFs since the other methods seems somewhat restrictive. In [8], only a single LPF is used for a given table. Here, we extend the results to splining a table with multiple LPFs to increase precision. We look at the complexity issues of determining these multiple LPFs as well as various special algorithms.

We begin in section 2 by studying single LPFs. In section 3, we present a natural extension of single LPFs for estimating missing data. In section 4, we discuss the problem of applying multiple LPFs and its complexity issues. And, in section 5, we consider some special approaches to multiple LPFs called bisecting splines and merging splines which are easier to work with.

2. Single LPFs

Given a Bayesian network, we can construct an ordered pair (V, P) where V is the set of r.v.s in the network and P is a set of conditional probabilities associated with the network. $P(A = a \mid C_1 = c_1, \ldots, C_n = c_n) \in P$ iff $C_1, \ldots, C_n$ are all the immediate parents of A and there is an edge from C_i to A for $i = 1, \ldots, n$ in the network. We can clearly see that (V, P) completely describes the Bayesian network.

Notation. $\Re$ denotes the real numbers. $\Re^n$ denotes the cross product of $\Re$ n times.

Let $\mathcal{B}$ be a Bayesian network and let C_0 be some r.v. in $\mathcal{B}$ with associated conditional table T. Assume T are probabilities of the form

$$P(C_0 = c_0 \mid C_1 = c_1, \ldots, C_n = c_n)$$

and we would like to replace it by an approximation function S. S should be a real-valued function from $\Re^{n+1}$ to $\Re$ with each argument corresponding to one of the c_i's. Unfortunately, as we pointed out earlier, instantiations for r.v.s need not be numeric in nature. Hence, it is necessary to map them into real values for our purposes.

Notation. Given a r.v. A, the set of possible values for A called the *range* of A will be denoted by $R(A)$.

Definition 2.1. Given a Bayesian network $\mathcal{B} = (V, P)$, an *instantiation* is an ordered pair (A, a) where $A \in V$ and $a \in R(A)$. (An instantiation (A, a) is also denoted by $A = a$ and $\mathsf{A_a}$.) A collection of instantiations w is called an *instantiation-set* iff $(A, a), (A, a')$ in w implies $a = a'$.

An instantiation represents the event when a r.v. takes on a value from its range.[4]

Definition 2.2. Given a r.v. A in V, a one-to-one and onto mapping E_A from $R(A)$ to $\Re$ is called an *encoder* for A.

[4] We can use instantiations and assignments interchangeably, however, whereas instantiation refers strictly to one r.v., assignment might refer to a collection of r.v.s.

Intuitively, encoders provide a total ordering on the possible instantiations for a r.v. As we saw earlier in Figs. 2 and 3, there are good encoders and bad encoders. Our problem is certainly easier if we just arbitrarily choose an encoder to work with. However, we cannot guarantee finding the best LPF.

The heart of the formulation lies in the effective identification of encoders and an approximation function simultaneously. In particular, we are interested in approximation functions of the form

$$S(E_A(C_0), E_{C_1}(c_1), \ldots, E_{C_n}(c_n)) = e^{k_0 E_{C_0}(c_0)+k_1 E_{C_1}(c_1)+\cdots+k_n E_{C_n}(c_n)+k}, \qquad (1)$$

which are our *linear potential functions* (LPFs) [8]. Clearly LPFs are simple continuous real functions but their appeal lies in the following observation: We compute joint probabilities by multiplying the various conditional probabilities together such as $p_1 p_2 \ldots p_k$ via the chain-rule. If we take the natural logarithm $\ln p_1 p_2 \ldots p_k = \ln p_1 + \ln p_2 + \cdots + \ln p_k$ and then replacing the p_i's by LPFs, this reduces our computation to a simple linear summation.

Ideally, we would like S to satisfy

$$e^{k_0 E_{C_0}(c_0)+k_1 E_{C_1}(c_1)+\cdots+k_n E_{C_n}(c_n)+k} \equiv P(C_0 = c_0 \mid C_1 = c_1, \ldots, C_n = c_n),$$

which can be rewritten in a simpler form as

$$\begin{aligned} &k_0 E_{C_0}(c_0) + k_1 E_{C_1}(c_1) + \cdots + k_n E_{C_n}(c_n) + k \\ &\equiv \ln P(C_0 = c_0 \mid C_1 = c_1, \ldots, C_n = c_n). \end{aligned}$$

As we can easily see, our goal is to determine the constants $k, k_0, k_1, \ldots, k_n$ and to determine the mapping of each particular instantiation of a r.v. to some real number. Hence, we have the following variables we must solve for:

- The constants $k, k_0, k_1, \ldots, k_n$.
- For each r.v. C_i, for each $c_i \in R(C_i)$, the encoding $E_{C_i}(c_i)$.

This can be accomplished by minimizing the following sum over the entries in table T:

$$\sum_{\substack{c_0 \in R(C_0) \\ \cdots \\ c_n \in R(C_n)}} \left[\sum_{j=0}^{n} k_j E_{C_j}(c_j) + k - \ln P(C_0 = c_0 \mid \ldots, C_l = c_l, \ldots) \right]^2. \qquad (2)$$

Obviously, this equation is some form of *least-squares fit*. We can find the minimum by taking the partial derivatives over (3) with respect to all the variables we must solve for; and then set these derivatives to 0. Unfortunately, the resulting system of equations is quadratic making it quite difficult to solve. However, there are certain

key observations about the minimization (see [8]) which allows the absorption of the constants $k, k_0, \ldots, k_n$ resulting in a simpler least-squares fit as follows:

$$\sum_{\substack{c_0 \in R(C_0) \\ \cdots \\ c_n \in R(C_n)}} \left[\sum_{j=0}^{n} E_{C_j}(c_j) - \ln P(C_0 = c_0 \mid \ldots, C_l = c_l, \ldots) \right]^2. \tag{3}$$

With this reduction, we have the following closed form solution:

Theorem 2.1. The minimal solution to (3) will be for each r.v. C_k in T, and for each $c_k \in R(C_k)$,

$$E_{C_k}(c_k) = \frac{1}{\prod_{\substack{i=0 \\ i \neq k}}^{n} m_i} \left\{ \sum_{\substack{d_0 \in R(C_0) \\ \cdots \\ d_n \in R(C_n) \\ d_k = c_k}} \ln P(C_0 = d_0 \mid \ldots, C_l = d_l, \ldots) \right\}$$

$$- \frac{n\xi(T)}{(n+1)\prod_{i=0}^{n} m_i}, \tag{4}$$

where $m_i = |R(C_i)|$ for $i = 0, \ldots, n$ and

$$\xi(T) = \sum_{\substack{d_0 \in R(C_0) \\ \cdots \\ d_n \in R(C_n)}} \ln P(C_0 = d_0 \mid \ldots, C_l = d_l, \ldots).$$

Proof. This follows from substituting (4) into the partial first derivatives of (3) and proving they are equal to 0. □

Furthermore, we can determine necessary and sufficient conditions for a perfect LPF. Our best approximation function will be a perfect fit if and only if

$$\sum_{j=0}^{n} E_{C_j}(c_j) = \ln P(C_0 = c_0 \mid \ldots, C_l = c_l, \ldots) \tag{5}$$

for all $c_j \in R(C_j)$ for $j = 0, \ldots, n$.

Theorem 2.2. There is a perfect fit if and only if

$$\sum_{j=0}^{n} m_j \left[\sum_{\substack{c_0' \in R(C_0) \\ \cdots \\ c_{j-1}' \in R(C_{j-1}) \\ c_0' \in R(C_0) \\ \cdots \\ c_{j-1}' \in R(C_{j-1})}} \ln P(C_0 = c_0' \mid C_1 = c_1', \ldots, C_j = c_j, \ldots, C_n = c_n') \right]$$

$$- \ln P(C_0 = c_0 \mid \ldots, C_l = c_l, \ldots) \prod_{i=0}^{n} m_i = \frac{n^2 \xi(T)}{(n+1)} \tag{6}$$

holds for all $c_i \in R(C_i)$ for $i = 0, \ldots, n$.

Proof. This follows from substituting (6) into (5). □

From this theorem, we can prove the following:

Corollary 2.3. Given a table T consisting of only one r.v. and either all entry values are distinct or identical, the best LPF S over T will be a perfect fit.

Although this corollary posits that entries must be all distinct or identical, we can still satisfy this condition by simply perturbing the values slightly.

As we can easily see, Theorems 2.1 and 2.2 allows us to compute LPFs easily and effectively via the closed form solutions.

Finally, let's assume that some set $\{A_1, A_2, \ldots, A_n\}$ of r.v.s in our given Bayesian network are now replaced by their respective LPFs $\{S_1, S_2, \ldots, S_n\}$. We now consider the overall expected error of computing a joint probability when using our LPFs. Again, since multiplying the conditional probabilities is equivalent to taking the sum of the logarithms, the expected error is as follows:

$$\sum_{i=1}^{n} \sum_{w \in T_i} v_i \big(\ln P_i(w) - \ln \widehat{P}_i(w) \big), \tag{7}$$

where $v_i = 1/|T_i|$, $|T_i|$ is the number of entries in the table associated with A_i, P_i is the conditional probability for A_i, and $\widehat{P}_i(w)$ is the approximation via S_i.

Theorem 2.4.

$$\sum_{i=1}^{n} \sum_{w \in T_i} v_i \big(\ln P_i(w) - \ln \widehat{P}_i(w) \big) = 0.$$

Proof. This follows from substituting in the optimal encoder values into (7). □

3. Incompleteness

As we mentioned earlier, Bayesian networks require that the conditional tables be complete with respect to all the possible combinations of assignments. Without this condition, it becomes impossible to compute various joint probabilities. Unfortunately, there are cases when some of the conditional probabilities may not be obtainable due to lack of information.

Now, let's briefly consider how we might deal with incompleteness using our LPFs approach. The problem then becomes, how do we approximate through a conditional table with missing entries. Clearly, the notion of approximating through the existing probabilities is easy to see. Once we have determined such a LPF, the pattern it captures will allow us to estimate the missing value by substituting corresponding r.v. assignments into the LPF.

Our new problem formulation is as follows: Let T' be an incomplete conditional table for C_0. We say that $(c_0, c_1, \ldots, c_n) \in T'$ if the associated conditional $P(C_0 = c_0, \ldots, C_n = c_n)$ is available in T'. Our goal is to minimize the following sum over the available entries in T':

$$\sum_{(c_0,\ldots,c_n)\in T'} \left[\sum_{j=0}^{n} E_{C_j}(c_j) - \ln P(C_0 = c_0 \mid \ldots, C_l = c_l, \ldots) \right]^2. \tag{8}$$

The optimal solution to (8) can be determined by solving the following system of linear equations:

$$m_{C_k}(c_k)E_{C_k}(c_k) + \xi_{C_k}(T', c_k) + \sum_{\substack{j=0 \\ j\neq k}}^{n} \sum_{d_j \in R(C_j)} m_{C_k,C_j}(c_k, c_j)E_{C_j}(d_j) = 0,$$

where $m_{C_k}(c_k) = |\{(d_0, \ldots, d_n) \in T' \mid d_k = c_k\}|$,

$$m_{C_k,C_j}(c_k, c_j) = \big|\{(d_0, \ldots, d_n) \in T' \mid d_k = c_k \text{ and } d_j = c_j\}\big|, \quad \text{and}$$

$$\xi_{C_k}(T', c_k) = - \sum_{\substack{(d_0,\ldots,d_n)\in T' \\ d_k = c_k}} \sum_{j=0}^{n} \ln P(C_0 = d_0 \mid \ldots, C_l = d_l, \ldots)$$

for all $c_k \in R(C_k)$, $k = 0, \ldots, n$.

Instead of performing the above computations, we can modify our closed form solution (4) for the complete table to get encodings for T'. The basic idea is to drop terms in the summations involving the missing entries. Also, we must modify our constants a little to accurately reflect the number of entries being used.

For each r.v. C_k in T', and for each $c_k \in R(C_k)$,

$$E_{C_k}(c_k) = \frac{1}{m_{C_k}(c_k)} \left\{ \sum_{\substack{(d_0,\ldots,d_n)\in T' \\ d_k = c_k}} \ln P(C_0 = d_0 \mid \ldots, C_l = d_l, \ldots) \right\} - \frac{n\xi(T')}{(n+1)|T'|}, \tag{9}$$

where

$$m_{C_k}(c_k) = \big|\{(d_0, \ldots, d_n) \in T' \mid d_k = c_k\}\big|$$

and

$$\xi(T') = \sum_{(c_0,\ldots,c_n)\in T'} \ln P(C_0 = c_0 \mid \ldots, C_l = c_l, \ldots).$$

These encodings are not necessarily optimal, however, they can still provide an adequate approximation for our problem.

To get an optimal solution with a closed form answer, we consider the following optimization:

$$\sum_{(c_0,\ldots,c_n)\in T}\left[\sum_{j=0}^{n} E_{C_j}(c_j) - g(c_0,\ldots,c_n)\right]^2, \tag{10}$$

where $g(c_0,\ldots,c_n) = \ln P(C_0 = c_0 \mid \ldots, C_l = c_l, \ldots)$ if $(c_0,\ldots,c_n) \in T'$, otherwise $g(c_0,\ldots,c_n)$ is a function on some set of neighboring entries of $(c_0,\ldots,c_n)$.

Theorem 3.1. The minimal solution to (10) will be for each r.v. C_k in T, and for each $c_k \in R(C_k)$,

$$E_{C_k}(c_k) = \frac{1}{\prod_{\substack{i=0\\ i\neq k}}^{n} m_i}\left\{\sum_{\substack{(d_0,\ldots,d_n)\in T\\ d_k=c_k}} g(d_0,\ldots,d_n)\right\} - \frac{n\overline{\xi}(T)}{(n+1)\prod_{i=0}^{n} m_i}, \tag{11}$$

where $m_i = |R(C_i)|$ for $i = 0,\ldots,n$ and

$$\overline{\xi}(T) = \sum_{(d_0,\ldots,d_n)\in T} g(d_0,\ldots,d_n).$$

(Proofs can be found in Appendix 8.)

Also, we can prove the following theorem about perfect fits under incompleteness for (10):

Theorem 3.2. There is a perfect fit if and only if

$$\sum_{j=0}^{n} m_j\left[\sum_{\substack{(d_0,\ldots,d_n)\in T\\ d_j=c_j}} g(d_0,\ldots,d_n)\right] - g(c_0,\ldots,c_n)\prod_{i=0}^{n} m_i = \frac{n^2\overline{\xi}(T)}{n+1} \tag{12}$$

holds for all $(c_0,\ldots,c_n) \in T$.

Proof. This immediately follows from (10) and Theorem 3.1. □

As we mentioned earlier, we can directly apply our LPFs into an integer linear programming formulation for reasoning with Bayesian networks. This particular approach also works naturally with the incomplete LPFs requiring absolutely no change in the formulation. (See [8] and section 7 for a description of the method.)

The methods for finding the optimal LPF as well as estimating incomplete information is not restricted to Bayesian networks and their conditional probability tables. In fact, it can be applied to any table of values in any domain. Hence, we have a general scheme which can be used in many domains needing data compression and/or dealing with incompleteness.

4. Multiple LPFs

Single LPFs work best when there is an overall single pattern on the conditional table. Unfortunately, people who design and develop the knowledge bases (networks) can axiomatize the domain in a strange fashion resulting in somewhat disjointed tables made up of multiple patterns. For example, take the simple case where we have two r.v. A and B with A conditionally dependent on B:

$$\begin{array}{ll} P(A = a_1 \mid B = b_1) = 0.10, & P(A = a_1 \mid B = b_2) = 0.45, \\ P(A = a_2 \mid B = b_1) = 0.15, & P(A = a_2 \mid B = b_2) = 0.30, \\ P(A = a_3 \mid B = b_1) = 0.30, & P(A = a_3 \mid B = b_2) = 0.15, \\ P(A = a_4 \mid B = b_1) = 0.45, & P(A = a_4 \mid B = b_2) = 0.10. \end{array}$$

In the case of b_1, our values are ascending. For b_2, they are descending. Here, we have two distinct patterns which will be difficult to unify into a single LPF.

One possible solution is to use a simple process called *clustering* [6]. We can always merge a set of r.v.s into a single r.v. and still preserve the probability distribution of the given domain. Assume $P(B = b_1) = 0.2$ and $P(B = b_2) = 0.8$. We can create a new r.v. C as follows:

$$\begin{array}{ll} P(C = a_1b_1) = 0.02, & P(C = a_1b_2) = 0.36, \\ P(C = a_2b_1) = 0.03, & P(C = a_2b_2) = 0.24, \\ P(C = a_3b_1) = 0.06, & P(C = a_3b_2) = 0.12, \\ P(C = a_4b_1) = 0.09, & P(C = a_4b_2) = 0.08. \end{array}$$

Replacing A and B by C and modifying the appropriate conditional links to point to C will not change the distribution of the domain. Clustering is used to reduce the number of nodes in the network as well as reducing the graph into simpler graphs such as polytress. However, as we can easily see, our table size has increased by a multiplicative factor which makes clustering expensive and is especially so for the algorithms which need to search through this unordered table.

On the positive side, since we have reduced ourselves to a table with a single r.v., we are guaranteed by Corollary 2.3 that we will have a perfect fit for our LPF. Although we have reduced ourselves to a single perfect LPF, we must still maintain the encoding information for the r.v. assignment. Space-wise, we have only traded one set of numbers – the probabilities in the table for another set – the encodings. However, this can still be an improvement over the old methods when combined with an integer linear programming algorithm (see section 7). Mainly, we would avoid performing a complete search through an unordered table. The transformation via the encoding provides such an ordering. In the case where we may have fixed or restricted the possible values to one or more of the original r.v.s before clustering (such as evidence r.v.s), determining which encodings are consistent with these restrictions can be easily accomplished with a simple cross-indexing scheme for storing the encodings.

Still, one of our goals is to reduce space consumption. Clustering is an incredibly efficient way of swallowing up memory. Instead of clustering to solve our above

example, we can use two separate LPFs over the original table. One LPF will range over the table with B fixed to b_1 and the other fixed to b_2. In particular, for this example, the individual subtables really consist of only one r.v. whose value changes across the entries. Thus, it follows from Corollary 2.3 that the LPFs will be perfect fits for their respective subtables. As opposed to storing a multiplicative number of encodings via the clustering method, we only need to keep an additive number. Clearly, this is a major point for using multiple LPFs especially when we generalize to larger tables involving several r.v.s. Also, they naturally give us better approximations compared to single LPFs. In effect, we are splining the points in the table with multiple LPFs.

We can obviously benefit a great deal from multiple LPFs but we must consider how to use them appropriately. We realize that there is a trade off to using more and more LPFs. While we get improved approximations, we also increase space consumption (although this is additive as opposed to multiplicative) and will likely increase the problems of trying to find such LPFs. Another critical problem will involve determining exactly which LPF we are supposed to use in retrieving a particular conditional probability.

Let's first address the critical issue of choosing the right LPF which is supposed to "cover" a given entry of interest. If we simply allow our LPFs to arbitrarily approximate collections of table entries, we are back to our original problem of not saving any space. We would essentially have to keep a complete table of which entries belong to which LPF regardless of how many LPFs are being used. However, by intelligently partitioning the table and then using exactly one LPF on each partition, we can develop an inexpensive method for determining which partition a particular entry resides which of course also tells us which LPF to use.

Again, let T be a conditional table with probabilities $P(C_0 = c_0 \mid C_1 = c_1, \ldots, C_n = c_n)$. Also, let E_{C_k} be some encoder for C_k for $k = 0, \ldots, n$. We call the set $E(T) = \{E_{C_0}, \ldots, E_{C_n}\}$ an *encoder-set* for T.

Definition 4.1. Let $\rho(T) = \{\sigma_1, \ldots, \sigma_s\}$ be a partition on T and $\mathcal{E}(T) = \{E_1(T), \ldots, E_s(T)\}$ be encoder-sets for T. We say that $\varrho(T) = (\rho(T), \mathcal{E}(T))$ is *contiguous*[5] if and only if for each cell σ_l in $\rho(T)$, there exists constants α_i and β_i for $i = 0, \ldots, n$ such that

$$\sigma_l = \big\{P(C_0 = c_0 \mid C_1 = c_1, \ldots, C_n = c_n) \in T \mid \alpha_i \leqslant E^l_{C_i}(c_i) \leqslant \beta_i \text{ for } i = 0, \ldots, n\big\}, \tag{13}$$

where $E_l(T) = \{E^l_{C_0}, \ldots, E^l_{C_n}\}$. We call each σ_l in $\rho(T)$ a *sub-table* of T.

Intuitively, we can view T as a multi-dimensional hypercube as follows: For the sake of simplicity, let's assume that the encoder-sets $E(T)$ are all identical, that

[5] A preliminary version of contiguity appears in [8]. This new definition generalizes [8] by incorporating multiple encoders as well as multiple LPFs.

is, $E_i(T) = E_j(T)$ for all i and j. For each C_i, we have uniquely numbered the instantiations $R(C_i)$ via E_{C_i}. We now construct a hypercube whose axes correspond to each r.v. C_i. We associate entry $P(C_0 = c_0 \mid C_1 = c_1, \ldots, C_n = c_n)$ as the point in this hypercube at coordinates $(E_{C_0}(c_0), \ldots, E_{C_n}(c_n))$. The hypercube will be bounded by the minimum and maximum values of the encoding on each $R(C_i)$. A contiguous partition of T will simply cut up the hypercube into a set of smaller hypercubes each bounded by the α_i's and β_i's.

Since each mini-hypercube $\sigma_k \in \rho(T)$ will be bounded by a unique collection of α_{k_i}'s and β_{k_i}'s, these constants will serve as our mechanism to determine which LPF an entry belongs to. Furthermore, this will correspond to a simple satisfiability test on the collections of inequalities from (13) for each sub-table. Clearly, the number of constants and equations involved are a small factor linear in n and s.

Associating different encoder-sets to each σ increases our flexibility. All that happens now is that we still guarantee the bounding properties for each σ_l via $E_l(T)$. However, applying $E_l(T)$ to σ_j when $l \neq j$ does not require this property. In order to determine which hypercube an entry now belongs to, we simply make the appropriate encodings from each $E_l(T)$ and run the inequalities test for σ_l in particular. We are guaranteed that the appropriate hypercube containing this entry can be determined uniquely in this way. [6]

Now, with a partitioning scheme available, we next consider how to determine these multiple LPFs. The fundamental difficulty we encountered in formulating single LPFs naturally occurs for multiple LPFs, namely, before we can partition a table, we must have an encoding and vice versa. Again, we must simultaneously determine the two in order to find the best multiple LPFs.

Multiple LPFs can be of immediate benefit when the designer can identify patterns in the tables already. We would then simply try to find the best LPFs using the single LPF technique for each pattern/hypercube identified. Clearly, this sort of situation is the best we could hope for reducing our overall problem.

When such an identification is not readily available, we must then consider the following question: What is the minimum number of LPFs that best fits a given table? We know that if we decided to associate one LPF with each entry individually, these $|R(C_0)||R(C_1)| \ldots |R(C_n)|$ LPFs will perfectly fit the table. Clearly though, this answer is unsatisfactory. We can get a second slightly tighter bound as follows:

Theorem 4.1. Let

$$\big|R(C_m)\big| = \max_{i=0,\ldots,n} \big|R(C_i)\big|.$$

There exists $|R(C_0)| \ldots |R(C_{m-1})||R(C_{m+1})| \ldots |R(C_n)|$ LPFs that perfectly fit table T with minor perturbations to T if necessary.

[6] At the end of section 7, we provide a brief description of how we can model this task as integer linear programming.

Proof. For each possible assignment $\{C_0 = c_0, \ldots, C_{m-1} = c_{m-1}, C_{m+1} = c_{m+1}, \ldots, C_n = c_n\}$, construct a cell whose entries are consistent with the assignment. Hence, this cell will only have entries which vary over C_m. Since each cell now really consists of one r.v., Corollary 2.3 guarantees a perfect LPF for the cell. □

Basically, we can construct a whole collection of LPFs that effectively only have to deal with one r.v. C_m varying its assignment through the cell. Hence, we can easily construct a perfect LPF for each partition.

Returning to the general question of determining the minimum number required, we find that the major difficulty will be in attempting to examine all the different possible partitionings of the table and whether or not a given partitioning permits a set of good LPFs.

More formally, we have the following least-squares optimization problem with constraints: Let each possible r.v.s assignment in T be represented by the n-tuple $(c_0, c_1, \ldots, c_n)$.

Definition 4.2. A *characteristic function* χ on T is a function mapping the n-tuple assignments of T to $\{0, 1\}$ such that χ can be decomposed into n *projection functions* χ_{C_i} which map $R(C_i)$ to $\{0, 1\}$ and

$$\chi(c_0, c_1, \ldots, c_n) = \chi_{C_0}(c_0)\chi_{C_1}(c_1)\ldots\chi_{C_n}(c_n).$$

Intuitively, a characteristic function for T can completely describe any hypercube (sub-table) arising from a contiguous partitioning of T. A value of 1, indicates that an entry belongs to the sub-table.

Our goal is to minimize the following sum:

$$\sum_{t=1}^{s} \sum_{(c_0,\ldots,c_n)\in T} \chi^t(c_0, \ldots, c_n) \times \left[\sum_{j=0}^{n} k_j^t E_{C_j}^t(c_j) + k^t - \ln P(C_0 = c_0 \mid \ldots, C_l = c_l, \ldots) \right]^2, \tag{14}$$

under the constraint:

$$\sum_{t=1}^{s} \chi^t(c_0, \ldots, c_n) = 1 \quad \text{for all } (c_0, \ldots, c_n) \in T, \tag{15}$$

where χ^t is the characteristic function for hypercube t. Note that this minimization actually corresponds to the problem of finding the best s or fewer LPFs. A hypercube with no entries is still a partition.

The variables we minimize over in (14) are

- The constants $k_0^t, \ldots, k_n^t, k^t$ for $t = 1, \ldots, m$.
- The encodings $E_{C_i}^t(c_i)$ for $i = 0, \ldots, n$ and $t = 1, \ldots, s$.
- The projections $\chi_{C_0}^t(c_0), \chi_{C_1}^t(c_1), \ldots, \chi_{C_n}^t(c_n)$ associated with characteristic functions χ^t for $t = 1, \ldots, s$.

In order to find the minimal solution, we can fold into the objective function the additional constraints of (15) and the projection functions restriction to $\{0, 1\}$ by using *Lagrange multipliers*. We can rewrite the $\{0, 1\}$ restriction mathematically as the constraint:

$$\chi_{C_i}(c_i)^2 - \chi_{C_i}(c_i) = 0.$$

We begin by taking the partial derivatives of the new combined objective function with respect to the above variables and Lagrange multipliers and setting them to 0. Through Lagrange's method, the minimal solution must satisfy the new equations. Unfortunately, unlike our single LPF minimization, our objective function is no longer quadratic in nature. The space of possible solutions includes all sorts of extreme points such as local minimas and saddlepoints. In fact, this space is extremely huge and covers about any possible combination of partitions and encodings as demonstrated by the next theorem.

Theorem 4.2. Given any hypercube partition $\rho(t) = \{\sigma_1, \ldots, \sigma_s\}$ and the following encodings for $c_k \in R^t(C_k)$,

$$E_{C_k}^t(c_k) = \frac{1}{\prod_{\substack{i=0 \\ i \neq k}}^n m_i^t} \left\{ \sum_{\substack{d_0 \in R^t(C_0) \\ \cdots \\ d_n \in R^t(C_n) \\ d_k = c_k}} \ln P(C_0 = d_0 \mid \ldots, C_l = d_l, \ldots) \right\}$$

$$- \frac{n \xi^t(T)}{(n+1) \prod_{i=0}^n m_i^t}, \tag{16}$$

where $R^t(C_i) = \{c_i \in R(C_i) \mid \chi_{C_i}^t(c_i) = 1\}$, $m_i^t = |R^t(C_i)|$, and

$$\xi^t(T) = \sum_{(c_0, \ldots, c_n) \in \sigma_t} \ln P(C_0 = d_0 \mid \ldots, C_l = d_l, \ldots),$$

this is an extreme point for (14).

To complete Theorem 4.2, we must consider those assignments in $R(C_i) - R^t(C_i)$ for which we have not computed an encoding. To guarantee that our solution is contiguous, all we need to do is make sure that these encodings are outside the bounds imposed by the hypercube σ_t.

Hence, the traditional methods for solving this least-squares problem will be difficult to apply successfully. In fact, with all these local minimas, maximas and saddlepoints, this strongly suggests that this problem is combinatorial in nature.

In Appendix 5, we present a possible approach based on integer linear programming for multiple LPFs.

5. Bisecting splines

As we have seen from the previous section, determining the globally optimal partitioning seems to be quite hard. However, we must realize that obtaining the global optimum is not necessarily required. Our original goal with LPFs is to ease the computational bottleneck from having to deal with large amounts of data. The global optimum simply gives us our ideal approximation.

Methods which give us good solutions that may not necessarily be the best can be of great benefit especially if they are relatively easy to compute. In this section, we consider a method based on *bisecting splines*.

Clearly, the problem of determining the best contiguous partitioning rests on the combinatorial number of partitions possible. Hence, we would like to restrict the number of partitions we search through. Our approach is an iterative scheme which continuously refines our splining approximations.

The bisecting splines method begins as follows: First, compute the best single LPF over T as in section 2. Once we have this LPF, we then determine the entry in T which is worst approximated by the single LPF. From the single LPF's encoders, we can totally order the assignments in each $R(C_i)$. This ordering will help us to break the table in $n+1$ different ways into 2 sub-tables each. Basically, assume $(d_0, \ldots, d_n)$ is the entry worst approximated. Now, for each r.v. C_i, partition $R(C_i)$ into two disjoint subsets: One subset containing all those assignments which precede d_i in the ordering, and the other subset containing those succeeding d_i including d_i.

We construct our first partitioning using $R(C_0)$ via its subset split. The remaining partitioning use $R(C_i)$, respectively. Next, taking each 2 sub-table partitioning, we consider the individual sub-tables and compute the best LPF spline over it. Hence, each 2 sub-table partitioning will have 2 LPFs giving us a contiguous partition.

We now choose the best contiguous partition from among the $n+1$ created. We are guaranteed that this new 2-LPF approximation will be at least as good as the original single LPF. Basically, if the single LPF had been the best possible, then the partitioning into 2 sub-tables would not have had any effect.

We continue to refine our approximations iteratively by now determining which new entry is worst approximated by the 2 new splines. We then take the sub-table containing this entry and break it further in n different ways into 2 sub-sub-tables so to speak. Again, we choose the best partitioning among these n and get a new 3-LPF approximation. As well, this 3-LPF approximation will be at least as good as the 2-LPF. We can continue this process until we reach the extreme case of one LPF for each entry in T. In essence, we are making successively finer and finer partitions.

Clearly, the bisecting splines method is relatively easy to perform. Furthermore, it guarantees better approximations as we continue to partition the tables.

An alternative approach to bisecting splines is to start from the finest partition and merge into successively larger partitions. In essence, we begin by associating a LPF with every entry and construct coarser and coarser approximations. With each iteration, we attempt to merge partitions in a manner which incurs the least amount

of error. We can stop when either we have a sufficiently small number of partitions or some error threshold is exceeded.

Notation. Give a cell σ in some partition of T, we define

$$\mathsf{span}_{C_i}(\sigma) = \{c_i \in R(C_i) \mid \exists (d_0, \ldots, d_n) \in \sigma \text{ such that } c_i = d_i\}.$$

We say that a cell σ is *complete* if

$$\forall (c_0, \ldots, c_n) \in \mathsf{span}_{C_0}(\sigma) \times \cdots \times \mathsf{span}_{C_n}(\sigma), \quad (c_0, \ldots, c_n) \in \sigma.$$

Given two cells σ_1 and σ_2, we say that they are *mergeable* if $\sigma_1 \cup \sigma_2$ is complete.

Our method begins as follows: Partition T so that each cell has exactly one entry and let σ_i^0 represent such a cell. Next, arbitrarily choose some cell σ_i^0 and K other cells $\{\sigma_{j_1}^0, \ldots, \sigma_{j_K}^0\}$ for some fixed constant K. If there does not exist a mergeable $\sigma_{j_l}^0$ with σ_i^0, then try choosing a different set of K until one is found. For each mergeable cell $\sigma_{j_l}^0$ with σ_i^0, compute the best single spline over newly formed hypercube. Choose the merging which introduces the least amount of error and form a new partition on T consisting of the untouched cells along with the new merged cell. Denote the cells in this new partition by σ_k^1.

In an iterative process, we now attempt to merge the σ_k^1's. We start again by arbitrarily choosing some cell σ_i^1.[7] Choose K other cells such that there is at least one mergeable cell with σ_i^1. Again, perform the merging which introduces the least error creating a new partition σ_k^2.

We are guaranteed that we will always generate contiguous hypercubes using this method. As well, we will have some control over the error involved with coarser partitions.

Like the bisecting splines method, merging splines are relatively easy to perform. We can also merge the two techniques together giving us a mix of bottom-up and top-down processing. Together, these methods provide us with viable approaches for computing multiple splines.

6. Results

In this section, we now perform some experiments on computing multiple LPFs. In particular, we consider the bisecting splines algorithm described in the previous section as compared to the single LPF.

There are several measures one can choose to determine the quality of the LPF approximations. For our experiments, we measured the approximations against the following two metrics:

[7] Instead of randomly choosing a cell, we could weight them according to the number of entries in each cell. Our goal is to merge smaller cells together first in somewhat more of a bottom up fashion.

- Absolute worst fit –
$$\max_{c \in T} |P(c) - \overline{P}(c)|. \tag{17}$$
- Relative average fit –
$$\frac{1}{|T|} \sum_{c \in T} \frac{|P(c) - \overline{P}(c)|}{P(c)}, \tag{18}$$

where T is the table, P is the actual probability, and $\overline{P}$ is our approximated probability. The relative fit will give us a percentage difference from the original table entry value.

We randomly generated conditional probability tables which varied in size from 100,000 entries to 3,200,000 entries and also varied the number of r.v.s found in the tables. In addition, we also generated tables ranging from uniform distributions to highly skewed distributions.

Clearly, our approximation approach should fare worst when faced with completely random tables. By their very nature, we do not expect to find any patterns or structures. We do realize though that most of the tables used in practice will by necessity be structured in some form.

Instead of generating completely random tables, we modify it slightly to permit some structure. We began by arbitrarily ordering the possible assignments for each r.v. and then generated random values in the table with the following property: Let a_1 and a_2 be some instantiations for r.v. A. For any two entries in the table which share exactly the same instantiations to all the r.v.s except for A and A is either a_1 or a_2, then the value in the entry with a_1 should have a value less than that for a_2. Note, however, that after we normalize the table, this property may no longer hold.

We began by taking these tables and fitting them with a single LPF. Our results measured using both absolute worst fit and relative average fit are summarized in Tables 1 and 2, respectively.

With these tables, we performed the bisecting splines algorithm to generate a 2-LPF approximation. Our results are summarized in Tables 3 and 4.

Table 1
Single LPF absolute worst fit.

	100,000	200,000	400,000	800,000	1,600,000	3,200,000
# r.v.s	3–12	3–12	3–12	3–12	3–12	3–12
Min	1.13×10^{-3}	1.21×10^{-3}	7.97×10^{-4}	6.54×10^{-4}	5.17×10^{-4}	4.52×10^{-4}
Max	4.98×10^{-3}	4.13×10^{-3}	3.46×10^{-3}	3.49×10^{-3}	3.35×10^{-3}	3.59×10^{-3}
Avg.	3.00×10^{-3}	2.89×10^{-3}	2.52×10^{-3}	2.51×10^{-3}	2.40×10^{-3}	2.34×10^{-3}

Table 2
Single LPF relative average fit.

	100,000	200,000	400,000	800,000	1,600,000	3,200,000
# r.v.s	3–12	3–12	3–12	3–12	3–12	3–12
Min	4.25×10^{-4}	4.08×10^{-4}	2.70×10^{-4}	2.46×10^{-4}	2.08×10^{-4}	1.85×10^{-4}
Max	1.67×10^{-3}	1.43×10^{-3}	1.16×10^{-3}	1.08×10^{-3}	1.00×10^{-3}	9.94×10^{-4}
Avg.	9.22×10^{-4}	8.22×10^{-4}	7.83×10^{-4}	7.25×10^{-4}	6.96×10^{-4}	6.96×10^{-4}

Table 3
Bisecting spline algorithm absolute worst fit improvement.

	100,000	200,000	400,000	800,000	1,600,000	3,200,000
Min	90.19%	44.22%	83.83%	34.80%	56.24%	20.65%
Max	31772.00%	39570.20%	47397.10%	29167.50%	28701.60%	19788.30%
Avg.	2591.38%	1838.02%	1886.17%	1651.69%	1578.91%	1146.99%

Table 4
Bisecting spline algorithm relative average fit improvement.

	100,000	200,000	400,000	800,000	1,600,000	3,200,000
Min	16.24%	11.28%	11.65%	12.42%	8.02%	5.86%
Max	3205.82%	4257.28%	5882.37%	3689.36%	5194.37%	6075.96%
Avg.	562.08%	600.57%	567.58%	382.36%	508.41%	497.53%

7. LPFs, belief revision, and ILP

We now present how LPFs (single as well as multiple) can be merged with integer linear programming for reasoning with Bayesian networks.[8] In particular, we consider *belief revision* which can be used for modeling explanatory/diagnostic tasks.

In belief revision, some evidence or observation is given to us, and our task is to come up with a set of hypothesis that together constitute the most satisfactory explanation/interpretation of the evidence at hand. More formally, if W is the set of all r.v.s in our given Bayesian network and e is our given evidence,[9] any complete instantiations to all the r.v.s in W which is consistent with e will be called an *explanation* or *interpretation* of e. Our problem is to find an explanation w^* such that

$$P(w^* \mid e) = \max_w P(w \mid e). \tag{19}$$

Intuitively, we can think of the non-evidence r.v.s in W as possible hypothesis for e.

To solve this using integer linear programming involves mapping the r.v. instantiations into some multi-dimensional space which we will denote by $\Re^n$. A subspace of $\Re^n$ will represent "valid" instantiations where valid includes things like being consistent to the given evidence e, each r.v. has at most one instantiation, etc. In particular, we are interested in transforming it into a *polyhedral convex set*.[10] Such a set can be described by a collection of linear inequalities. As it turns out, these inequalities will intuitively correspond to the restrictions/constraints required in making valid instantiations of the r.v.s. Finally, we would like to define a *linear energy function* such that by minimizing it over the convex set, the resulting answer will be the most probable solution.

[8] See [8] for additional details.
[9] That is, e represents a set of instantiations made on a subset of W.
[10] "Polyhedral" refers to the fact that the boundaries of the subspace are composed of hyperplanes.

Notation. Throughout the remainder of this paper, upper case italicized letters such as $A, B, \ldots$ will represent r.v.s and lower case italicized letters such as $a, b, \ldots$ will represent the possible assignments to the associated upper case letter r.v., in this case, $A, B, \ldots$. Subscripted upper case letters which are not italicized are variables in a constraint system which explicitly represent the instantiation of the associated r.v. with the item in the subscript. For example, $\mathsf{A_a}$ denotes the instantiation of r.v. A with value a.

Definition 7.1. Given an instantiation-set w for a Bayesian network $\mathcal{B} = (V, P)$, we define the *span* of w, $\mathsf{span}(w)$, to be the collection of r.v.s in the first coordinate of the instantiations. Furthermore, an instantiation-set w is said to be *complete* iff $\mathsf{span}(w) = V$.

Notation. For each r.v. A, we define $\mathsf{cond}(A)$ as follows: $B \in \mathsf{cond}(A)$ iff there exists a conditional probability in P of the form $P(A = a \mid \ldots, B = b, \ldots)$.

Notation. Given an instantiation-set w for $\mathcal{B}$ such that $\mathsf{cond}(A) \subseteq \mathsf{span}(w)$, $w(A)$ denotes the instantiation $A = a$ where $(A, a) \in w$.

$$w(\mathsf{cond}(A)) = \big\{w(B) \mid B \in \mathsf{cond}(A)\big\}.$$

Let $\mathcal{E}_\mathcal{B}$ be a collection of encoders for the r.v.s in V such that there is exactly one encoder for each r.v.

Notation. Let T be a collection of conditional probabilities from P.

$$\textsc{rv}(T) = \big\{A \mid P(C_0 = c_0 \mid C_1 = c_1, \ldots, C_n = c_n) \in T \text{ and } C_i \equiv A \text{ for some } i\big\}.$$

Definition 7.2. Let T be a conditional table in $\mathcal{B}$ and assume

$$\textsc{rv}(T) = \{C_0, C_1, \ldots, C_n\}.$$

Let $\rho(T)$ be a partition on T. We say the $\rho(T)$ is *contiguous* with respect to $\mathcal{E}_\mathcal{B}$ if and only if for each cell σ in $\rho(T)$, there exists α_i, β_i for $i = 0, \ldots, n$ such that

$$\sigma = \big\{P(C_0 = c_0 \mid C_1 = c_1, \ldots, C_n = c_n) \in T \mid \\ \alpha_i \leqslant E_{C_i}(c_i) \leqslant \beta_i \text{ for } i = 0, \ldots, n\big\}.$$

We call each σ in $\rho(T)$ a *sub-table* of T.

(*Note:* This is a simpler definition of contiguity than Definition 4.1. Here, we only consider a single encoder-set for ease of discussion. At the end of this section, we will briefly consider multiple encoders.)

Definition 7.3. Let $\rho(T)$ be a contiguous partition with respect to $\mathcal{E}_\mathcal{B}$. For each cell σ in $\rho(T)$, we associate a LPF $S_{\mathcal{E}_\mathcal{B},\sigma}$ from $\Re^{n+1}$ to $\Re$ such that

$$S_{\mathcal{E}_\mathcal{B},\sigma}\big(E_{C_0}(c_0), E_{C_1}(c_1), \ldots, E_{C_n}(c_n)\big) = P(C_0 = c_0 \mid C_1 = c_1, \ldots, C_n = c_n), \quad (20)$$

where $E_{C_0}, E_{C_1}, \ldots, E_{C_n}$ are the encoders found in $\mathcal{E}_\mathcal{B}$ and

$$c_0 \in R(C_0), \ldots, c_n \in R(C_n).$$

We call $S_{\mathcal{E}_\mathcal{B},\sigma}$ a LPF for σ. Let $\mathcal{S}_{\mathcal{E}_\mathcal{B},\rho(T)}$ be a collection of LPFs associated with partition $\rho(T)$. We call $\mathcal{S}_{\mathcal{E}_\mathcal{B},\rho(T)}$ a LPF-*set* for T.

Let $\mathcal{S}_{\mathcal{E}_\mathcal{B}}$ be a space of LPF-sets, also called a *linear potential space*, such that each table T in $\mathcal{B}$ is associated with exactly one LPF-set.

Notation. Given $\mathcal{S}_{\mathcal{E}_\mathcal{B},\rho(T)}$, for any $c_0 \in R(C_0), \ldots, c_n \in R(C_n)$,

$$S_{\mathcal{E}_\mathcal{B},\rho(T)}\big(E_{C_0}(c_0), \ldots, E_{C_n}(c_n)\big)$$

will unambiguously refer to the appropriate LPFLPF defined in

$$\mathcal{S}_{\mathcal{E}_\mathcal{B},\rho(T)}.$$

Given $\mathcal{S}_{\mathcal{E}_\mathcal{B}}$, since all our conditional probability tables are unique, for any $c_0 \in R(C_0), \ldots, c_n \in R(C_n)$,

$$S_{\mathcal{E}_\mathcal{B}}\big(E_{C_0}(c_0), \ldots, E_{C_n}(c_n)\big)$$

will unambiguously refer to the appropriate LPF defined in $\mathcal{S}_{\mathcal{E}_\mathcal{B}}$.

We now redefine our notion of a Bayesian network.

Definition 7.4. Given a Bayesian network $\mathcal{B} = (V, P)$, let $\mathcal{E}_\mathcal{B}$ be a collection of encoders and $\mathcal{S}_{\mathcal{E}_\mathcal{B}}$ be a collection of spline-sets. We define a *splined Bayesian network* to be a 3-tuple $\overline{\mathcal{B}} = (V, \mathcal{E}_\mathcal{B}, \mathcal{S}_{\mathcal{E}_\mathcal{B}})$.

Definition 7.5. Given a complete instantiation set w for $\mathcal{B}$, we define the *spline probability,* P_s for $\overline{\mathcal{B}}$ as

$$P_s(w) = \prod_{A \in \mathsf{span}(w)} S_{\mathcal{E}_\mathcal{B}}\big(w(A), w(\mathsf{cond}(A))\big). \quad (21)$$

Theorem 7.1. Any Bayesian network can be modeled as a splined Bayesian network.

We now proceed to show how we can transform splined Bayesian networks into linear constraint satisfaction problems.

Clearly, a splined Bayesian network $\overline{\mathcal{B}} = (V, \mathcal{E}_{\mathcal{B}}, \mathcal{S}_{\mathcal{E}_{\mathcal{B}}})$ induces a partition on the original conditional probabilities P. Let $[P]_{\overline{\mathcal{B}}}$ denote this induced partitioning. Furthermore, a LPF from $\mathcal{S}_{\mathcal{E}_{\mathcal{B}}}$ is uniquely associated with each cell in the partition. If d is a cell in $[P]_{\overline{\mathcal{B}}}$, then S_d will uniquely denote the appropriate LPF.

We say that a r.v. instantiation $\{A = a\}$ *appears* in a cell in $[P]_{\overline{\mathcal{B}}}$ if there exists a conditional probability in the cell of the form $P(C_0 = c_0 \mid C_1 = c_1, \ldots, C_n = c_n)$ where for some $i = 1, \ldots, n$, $C_i \equiv A$ and $c_i \equiv a$.

Definition 7.6. Given a r.v. A and a splined Bayesian network $\overline{\mathcal{B}}$, we define the partition on $R(A)$ induced by $\overline{\mathcal{B}}$ as follows: $a_1, a_2 \in R(A)$ both belong in the same partition if and only if for all cells, d in $[P]_{\overline{\mathcal{B}}}$, $\{A = a_1\}$ appears in d if and only if $\{A = a_2\}$ appears in d. We call this partitioning the *capsulation* of A by $\overline{\mathcal{B}}$ and denote it by $[A]_{\overline{\mathcal{B}}}$.

Theorem 7.2. Given any cell $d = \{a_1, \ldots, a_k\}$ in $[A]_{\overline{\mathcal{B}}}$ where $E_A(a_i) < E_A(a_{i+1})$, there does not exist $b \in R(A)$ such that $E_A(a_1) < E_A(b) < E_A(a_k)$ and $b \neq a_i$ for $i = 1, \ldots, k$.

Without going into detail, we must generalize our notion of constraint systems. Previously for belief revision, we restricted our variables to values of 0 and 1. We generalize this by allowing variables to be restricted to sets of real values. In doing so, we also generalize our notion of a 0–1 solution to the notion of a *permissible* solution as a solution which satisfies all the value restrictions as well as constraints.

We begin our construction as follows: Let $\overline{\mathcal{B}} = (V, \mathcal{E}_{\mathcal{B}}, \mathcal{S}_{\mathcal{E}_{\mathcal{B}}})$ be a splined Bayesian network and $\mathcal{S}_{\mathcal{E}_{\mathcal{B}}}$ be a linear potential space. For each r.v. A in V, construct a real variable x_A whose values are restricted to $\{E_A(a) \mid E_A \in \mathcal{E}_{\mathcal{B}}$ and $a \in R(A)\}$. Next, arbitrarily label all the cells in $[A]_{\overline{\mathcal{B}}}$ by

$$\{d_{A,1}, d_{A,2}, \ldots, d_{A,n}\}.$$

For each cell in $\{d_{A,1}, d_{A,2}, \ldots, d_{A,n}\}$, construct a new 0–1 restricted variable $x_{d_{a,i}}$. Construct the following constraints:

$$\sum_{i=1}^{n} x_{d_{A,i}} = 1. \tag{22}$$

For each $d_{A,i}$,

$$x_A - \min_{a \in d_{A,i}} E_A(a) \geqslant -M(1 - x_{d_{A,i}}), \tag{23}$$

$$x_A - \max_{a \in d_{A,i}} E_A(a) \leqslant M(1 - x_{d_{A,i}}), \tag{24}$$

where M is some arbitrarily large positive constant. Intuitively, $x_{d_{A,i}}$ is used to "detect" whether x_A falls within a particular interval defined by the partitioning and x_A represents the instantiated value if any.

From our definitions above, we can unambiguously associate a LPF S in $\mathcal{S}_{\mathcal{E}_\mathcal{B}}$ with some table T. Furthermore, using our encoder mappings, we describe the domain of S as a cross product of intervals on real variables associated with the r.v.s in T. For example, let $\rho(T)$ be the partition on T associated with S. According to Definition 7.3, for each r.v. B in RV(T), there exists a set of instantiations of B, $\{B = b_{i_1}, \ldots, B = b_{i_k}\}$ in the cell associated with S in $\rho(T)$ such that there does not exist an instantiation $\{B = c\}$ where $c \neq b_{i_j}$ for all i_j and $E_B(c)$ is in the interval from $\min_{i_j} E_B(b_{i_j})$ to $\max_{i_j} E_B(b_{i_j})$. Thus, we only need the min and the max values to describe the domain for S. Also, remember that we want to incorporate S into our probabilistic computations only when the r.v.s are instantiated within its restricted domain. Let us denote the interval for a r.v. A for LPF S by $[\min(A, S), \max(A, S)]$. We now proceed with our construction. For each LPF S involving the r.v.s $\{C_1, \ldots, C_n\}$, we construct the new variables $x_{C_i,S}$ which are virtual copies of x_{C_i} constructed above. For each $[\min(C_i, S), \max(C_i, S)]$, if there does not yet exist a 0–1 variable which detects whether x_{C_i} is in $[\min(C_i, S), \max(C_i, S)]$, create a new 0–1 variable $d_{C_i,S}$ with the following constraints: For each $d_{A,i}$,

$$x_{C_i} - \min(C_i, S) \geqslant -M(1 - d_{C_i,S}), \tag{25}$$

$$x_{C_i} - \max(C_i, S) \leqslant M(1 - d_{C_i,S}). \tag{26}$$

Now, continue and construct a new 0–1 real variable d_S adding the following constraints:

$$d_S \geqslant n - \sum_{j=1}^{n} d_{C_j,S} + 1, \tag{27}$$

$$d_S \leqslant \frac{1}{n} \sum_{j=1}^{n} d_{C_j,S}. \tag{28}$$

For each i,

$$x_{C_i,S} \geqslant x_{C_i} - M(1 - d_S), \tag{29}$$

$$x_{C_i,S} \leqslant x_{C_i} + M(1 - d_S), \tag{30}$$

$$x_{C_i,S} \leqslant M d_S, \tag{31}$$

d_S is used to indicate whether S will be used. If so, copy the values into the virtual copies and make the computations necessary based on the virtual copies.

We complete our transformation by defining an appropriate objective function. For each LPF S in $\mathcal{S}_{\mathcal{E}_\mathcal{B}}$, introduce the following terms into the objective function: Assume that

$$S_{l,\mathcal{E}_\mathcal{B}}\big(E_A(a), E_{C_1}(c_1), \ldots, E_{C_n}(c_n)\big) \equiv e^{k_0 E_A(a) + k_1 E_{C_1}(c_1) + \cdots + k_n E_{C_n}(c_n) + k}.$$

The terms to be added are

$$-kd_S - \sum_{i=1}^{n} k_i x_{C_i,S}. \tag{32}$$

We now must show that the solution space defined by our induced constraint system is equivalent to the space of all complete instantiation-sets for the Bayesian network. We begin by providing a transformation from permissible assignments in our constraint systems to instantiation-sets. Let s be a permissible solution. We can construct a complete instantiation-set $w[s]$ as follows: For each r.v. A in V, A is instantiated if and only if $s(x_A) > 0$. Since our encoders are one-to-one and onto, then the inverse (or, called *decoder*) exists and we denote them by E_A^{-1}. If $s(x_A) > 0$, then $w[s](A) = E_A^{-1}(x_A)$.

Conversely, given a complete instantiation-set w, we can construct a permissible assignment $s[w]$ as follows: For each r.v. A in V, $s[w](x_A) = E_A(w(A))$. Furthermore, we properly activate the appropriate interval detectors. Finally, according to the instantiation-set w, we can easily determine which LPFs are active. $s[w](d_S) = 1$ if and only if S is an active LPF according to w. And, if S is active, then copy $s[w](x_{C_i,S}) = s[w](x_{C_i})$ for all i involved with S. Otherwise, $s[w](x_{C_i,S}) = 0$.

Theorem 7.3. w is a complete instantiation-set for $\mathcal{B}$ if and only $s[w]$ is a permissible solution for the induced constraint system.

Having shown the equivalence, we can prove the following theorem on the probabilities being calculated.

Theorem 7.4.

$$P_s(w) = e^{-\Theta(s[w])}.$$

Therefore, the optimal permissible solution for our induced constraint system will be the best complete instantiation set.

Finally, we must also incorporate the notion of evidence. Evidence, we recall, is the requirement that a r.v. be instantiated with a certain value. For the case where a r.v. A must be instantiated to a, we simply include the constraint $x_A = E_A(a)$.

One final note for this section is on the use of multiple encoders for a single r.v. By associating a separate encoder-set to each conditional table, this will obviously increase the flexibility of our formulation to compress the tables. Furthermore, multiple encoders will only increase our complexity linearly, we just have to guarantee consistency between encoders. If one encoder says that A is instantiated to a, any other encoders must also instantiate A to a. For example, say we have two encoders E_A^1 and E_A^2 and let $a \in R(A)$. Let x_A^1 and x_A^2 be the encoder variables associated to the two encoders, respectively. Basically, we must make sure that $x_A^1 = E_A^1(a)$ iff $x_A^2 = E_A^2(a)$. We can accomplish this by the following constraints:

$$x_A^1 - E_A^1(a) \geqslant -M(1 - y_{A=a}),$$
$$x_A^1 - E_A^1(a) \leqslant M(1 - y_{A=a}),$$

$$x_A^2 - E_A^2(a) \geqslant -M(1 - y_{A=a}),$$
$$x_A^2 - E_A^2(a) \leqslant M(1 - y_{A=a}),$$

where $y_{A=a}$ is a 0–1 detector variable. Obviously, this can be generalized to any number of encoders.

8. Conclusions

Reducing the storage and processing costs of probabilistic information is necessary for reasoning systems especially when we move to real world problems. This can be achieved by identifying patterns in the probabilistic tables. Splining functions provide a flexible approach to exploiting these patterns. In particular, LPFs can reduce the storage consumption from a typically multiplicative growth, $O(s_1 s_2 \cdots s_n)$, to an additive one, $O(s_1 + s_2 + \cdots + s_n)$.

In addition to the fact that single LPFs can be computed relatively easily via closed-form solutions, we show that the LPFs now provide us with an ordering on the probabilistic information not available before. This ordering can be used to help in searching the tables.

A natural extension to single LPFs are multiple LPFs with multiple encoders. Multiple LPFs provide better approximations while still minimizing storage and processing costs. The difficulty in using multiple LPFs is in trying to determine which LPF we are supposed to be computing with at any given moment. In particular, given a collection of LPFs over a probability table, to retrieve a particular entry in the table via LPFs requires us to know which LPF we must look at. By carefully partitioning the table into *contiguous hypercubes*, we can solve this identification problem by using quick and easy bounds tests.

When the knowledge engineer/domain builder can readily identify patterns within a table, we can immediately apply multiple LPFs. In case such information isn't handy, we can identify patterns automatically using techniques such as bisecting splines and merging splines.

Experimental results for multiple LPFs seem very promising. Going to a 2-LPF approximation alone resulted in an average 500–600% improvement in overall fit to the conditional probability tables.

Both single and multiple LPFs can be used directly in integer linear programming models for reasoning. Integer linear programming has been shown to be an efficient method for reasoning and seems to scale to larger problems quite well [5, 9–12].

Additionally, since LPFs are used in an effort to identify patterns, this impacts on the problem of missing/incomplete information. Without complete information, reasoning models such as Bayesian networks cannot be used. By using the patterns we identified through our approximations, we can then extrapolate the missing information.

Finally, the LPF approach can also be applied to any domain besides probabilistic ones which require large amounts of data. Our approach, as we have presented here, is quite general and independent of the probabilistic framework we targeted.

Some of the questions we continue to pursue include proving whether or not the problem of finding the best set of multiple LPFs is NP-hard. Another thread involves merging our approach with a more advanced independence-based assignments methodology called δ-IB assignments [16]. Instead of requiring that the entries be identical, they can be similar within some δ. This approach can help provide bounds on the expected errors from our approximations and how it impacts the overall reasoning systems.

Appendix. Proofs

Theorem 3.1. The minimal solution to (10) will be for each r.v. C_k in T, and for each $c_k \in R(C_k)$,

$$E_{C_k}(c_k) = \frac{1}{\prod_{\substack{i=0 \\ i\neq k}}^{n} m_i} \left\{ \sum_{\substack{(d_0,\ldots,d_n)\in T \\ d_k = c_k}} g(d_0,\ldots,d_n) \right\} - \frac{n\overline{\xi}(T)}{(n+1)\prod_{i=0}^{n} m_i}, \tag{33}$$

where $m_i = |R(C_i)|$ for $i = 0,\ldots,n$ and

$$\overline{\xi}(T) = \sum_{(d_0,\ldots,d_n)\in T} g(d_0,\ldots,d_n).$$

Proof. We begin by making the following observation: First, the encoders are already variables themselves. We find that we can simply absorb the constants k_j's in front of them in (10). Furthermore, k is a translation factor which can also be absorbed by the encoders. This leaves us with

$$f = \sum_{(c_0,\ldots,c_n)\in T} \left[\sum_{j=0}^{n} E_{C_j}(c_j) - g(c_0,\ldots,c_n) \right]^2.$$

Taking the partial first derivatives, we get

$$\frac{\partial}{\partial E_{C_k}(c_k)} f = 2 \sum_{\substack{(d_0,\ldots,d_n)\in T \\ d_k = c_k}} \left[\sum_{j=0}^{n} E_{C_j}(d_j) - g(c_0,\ldots,c_n) \right].$$

Now, substitute in the encoder assignments above to the partial derivatives.

$$\sum_{\substack{(d_0,\ldots,d_n)\in T \\ d_k = c_k}} \left[\sum_{j=0}^{n} E_{C_j}(d_j) - g(c_0,\ldots,c_n) \right]$$

$$= E_{C_k}(c_k) \prod_{\substack{i=0 \\ i\neq k}}^{n} m_i + \sum_{\substack{j=0 \\ j\neq k}}^{n} \sum_{d_j \in R(C_j)} E_{C_j}(d_j) \prod_{\substack{i=0 \\ i\neq k \\ i\neq j}}^{n} m_i$$

$$- \sum_{\substack{(d_0,\ldots,d_n)\in T \\ d_k = c_k}} g(d_0,\ldots,d_n)$$

$$= \sum_{\substack{(d_0,\ldots,d_n)\in T \\ d_k = c_k}} g(d_0,\ldots,d_n) - \frac{n\overline{\xi}(T)}{(n+1)m_k}$$

$$+ \sum_{\substack{j=0 \\ j\neq k}}^{n} \sum_{d_j \in R(C_j)} \frac{1}{m_k} \sum_{\substack{(e_0,\ldots,e_n)\in T \\ e_j = d_j}} g(e_0,\ldots,e_n)$$

$$- \sum_{\substack{j=0 \\ j\neq k}}^{n} \sum_{d_j \in R(C_j)} \frac{n\overline{\xi}(T)}{(n+1)m_j m_k} - \sum_{\substack{(d_0,\ldots,d_n)\in T \\ d_k = c_k}} g(d_0,\ldots,d_n)$$

$$= -\frac{n\overline{\xi}(T)}{(n+1)m_k} - \frac{1}{m_k} \sum_{\substack{j=0 \\ j\neq k}}^{n} \frac{1}{m_j} \sum_{d_j \in R(C_j)} \frac{n\overline{\xi}(T)}{n+1} + \frac{1}{m_k} \sum_{\substack{j=0 \\ j\neq k}}^{n} \overline{\xi}(T)$$

$$= -\frac{n\overline{\xi}(T)}{(n+1)m_k} - \frac{1}{m_k} \sum_{\substack{j=0 \\ j\neq k}}^{n} \frac{n\overline{\xi}(T)}{n+1} + \frac{1}{m_k} \sum_{\substack{j=0 \\ j\neq k}}^{n} \overline{\xi}(T)$$

$$= -\frac{n\overline{\xi}(T)}{(n+1)m_k} - \frac{n^2\overline{\xi}(T)}{m_k(n+1)} + \frac{n\overline{\xi}(T)}{m_k} = 0. \qquad \square$$

Theorem 4.2. Given any hypercube partition $\rho(t) = \{\sigma_1, \ldots, \sigma_s\}$ and the following encodings

$$E^t_{C_k}(c_k) = \frac{1}{\prod_{\substack{i=0 \\ i\neq k}}^{n} m_i^t} \left\{ \sum_{\substack{d_0 \in R^t(C_0) \\ \cdots \\ d_n \in R^t(C_n) \\ d_k = c_k}} \ln P(C_0 = d_0 \mid \ldots, C_l = d_l, \ldots) \right\}$$

$$- \frac{n\xi^t(T)}{(n+1)\prod_{i=0}^{n} m_i^t}, \tag{34}$$

where $R^t(C_i) = \{c_i \in R(C_i) \mid \chi^t_{C_i}(c_i) = 1\}$, $m_i^t = |R^t(C_i)|$, and

$$\xi^t(T) = \sum_{(c_0,\ldots,c_n)\in\sigma_t} \ln P(C_0 = d_0 \mid \ldots, C_l = d_l, \ldots),$$

this is an extreme point for (14).

Proof. We begin by rewriting (14) in terms of projection functions as well as with Lagrange multipliers:

$$
\begin{aligned}
f = &\sum_{t=1}^{m} \sum_{(c_0,\ldots,c_n)\in T} \left[\prod_{i=0}^{n} \chi_{C_i}^{t}(c_i)\right] \\
&\times \left[\sum_{j=0}^{n} k_j^t E_{C_j}^{t}(c_j) + k^t - \ln P(C_0 = c_0 \mid \ldots, C_l = c_l, \ldots)\right]^2 \\
&+ \sum_{(c_0,\ldots,c_n)\in T} \lambda_{(c_0,\ldots,c_n)} \left[\sum_{t=1}^{m}\prod_{i=0}^{n} \chi_{C_i}^{t}(c_i) - 1\right] \\
&+ \sum_{t=1}^{m}\sum_{i=0}^{n} \sum_{c_i\in R(C_i)} \lambda_{C_i}^{t}(c_i)\left[\chi_{C_i}^{t}(c_i)^2 - \chi_{C_i}^{t}(c_i)\right]. \qquad (35)
\end{aligned}
$$

We make the following observations: First, the encoders are already variables themselves. We find that we can simply absorb the constants k_j^t's in front of them in (35). Furthermore, k^t is a translation factor which can also be absorbed by the encoders. This leaves us with

$$
\begin{aligned}
f = &\sum_{t=1}^{m} \sum_{(c_0,\ldots,c_n)\in T} \left[\prod_{i=0}^{n} \chi_{C_i}^{t}(c_i)\right] \\
&\times \left[\sum_{j=0}^{n} E_{C_j}^{t}(c_j) - \ln P(C_0 = c_0 \mid \ldots, C_l = c_l, \ldots)\right]^2 \\
&+ \sum_{(c_0,\ldots,c_n)\in T} \lambda_{(c_0,\ldots,c_n)} \left[\sum_{t=1}^{m}\prod_{i=0}^{n} \chi_{C_i}^{t}(c_i) - 1\right] \\
&+ \sum_{t=1}^{m}\sum_{i=0}^{n} \sum_{c_i\in R(C_i)} \lambda_{C_i}^{t}(c_i)\left[\chi_{C_i}^{t}(c_i)^2 - \chi_{C_i}^{t}(c_i)\right]. \qquad (36)
\end{aligned}
$$

Now, taking the partial derivatives, we get

$$
\begin{aligned}
\frac{\partial}{\partial \chi_{C_p}^{v}(c_p)} f = &\sum_{\substack{(d_0,\ldots,d_n)\in T \\ d_p = c_p}} \left[\prod_{\substack{i=0 \\ i\neq p}}^{n} \chi_{C_i}^{v}(d_i)\right] \\
&\times \left[\sum_{j=0}^{n} E_{C_j}^{v}(d_j) - \ln P(C_0 = d_0 \mid \ldots, C_l = d_l, \ldots)\right]^2
\end{aligned}
$$

$$+\sum_{\substack{(d_0,\ldots,d_n)\in T\\ d_p=c_p}}\lambda_{(d_0,\ldots,d_n)}\left[\prod_{\substack{i=0\\ i\neq p}}^{n}\chi^v_{C_i}(d_i)\right]$$

$$+2\lambda^v_{C_p}(c_p)\chi^v_{C_p}(c_p)-\lambda^v_{C_p}(C_p), \tag{37}$$

$$\frac{\partial}{\partial\lambda_{(c_0,\ldots,c_n)}}f=\sum_{t=1}^{m}\prod_{i=0}^{n}\chi^t_{C_i}(c_i)-1, \tag{38}$$

$$\frac{\partial}{\partial\lambda^t_{C_i}(c_i)}f=\chi^t_{C_i}(c_i)^2-\chi^t_{C_i}(c_i), \tag{39}$$

$$\frac{\partial}{\partial E^v_{C_p}(c_p)}f=\sum_{\substack{(d_0,\ldots,d_n)\in T\\ d_p=c_p}}2\left[\prod_{i=0}^{n}\chi^v_{C_i}(c_i)\right]$$

$$\times\left[\sum_{j=0}^{n}E^v_{C_j}(d_j)-\ln P(C_0=d_0\mid\ldots,C_l=d_l,\ldots)\right]. \tag{40}$$

An extreme point will be any solution that satisfies the partial derivatives being set to 0.

Since $\rho(T)$ is a hypercube partition of T, that is, we can associate characteristic functions to each cell in $\rho(T)$, (38) and (39) are automatically satisfied.

Next, rewrite (37) as follows:

$$\lambda^v_{C_p}(c_p)=\frac{1}{1-2\chi^v_{C_p}(c_p)}\sum_{\substack{(d_0,\ldots,d_n)\in T\\ d_p=c_p}}\left[\prod_{\substack{i=0\\ i\neq p}}^{n}\chi^v_{C_i}(d_i)\right]$$

$$\times\left[\sum_{j=0}^{n}E_{C_j}(d_j)-\ln P(C_0=d_0|\ldots,C_l=d_l,\ldots)\right]^2$$

$$+\sum_{\substack{(d_0,\ldots,d_n)\in T\\ d_p=c_p}}\lambda_{(d_0,\ldots,d_n)}\left[\prod_{\substack{i=0\\ i\neq p}}^{n}\chi^v_{C_i}(d_i)\right]. \tag{41}$$

We can arbitrarily assign values for $\lambda_{(d_0,\ldots,d_n)}$ and satisfy (37) by (41).

All we have left to do is to show that the choice of encoder mappings above will satisfy (40).

For a particular cell σ_v in $\rho(T)$, (40) reduces to

$$\sum_{\substack{(d_0,\ldots,d_n)\in\sigma_v\\ d_p=c_p}}\left[\sum_{j=0}^{n}E^v_{C_j}(d_j)-\ln P(C_0=d_0\mid\ldots,C_l=d_l,\ldots)\right]. \tag{42}$$

Observe that there are no interdependencies via (42) between encoders from different encoder-sets. Hence, we can group together all the equations from (42) by encoder-sets.

Now, let's plug in the encoder mappings above into (42).

$$\sum_{\substack{(d_0,\ldots,d_n)\in\sigma_v\\ d_p=c_p}} \left[\sum_{j=0}^{n} E^v_{C_j}(c_j) - \ln P(C_0=d_0 \mid \ldots, C_l=d_l,\ldots)\right]$$

$$= E^v_{C_p}(c_p)\prod_{\substack{i=0\\ i\neq p}}^{n} m^v_i + \sum_{\substack{j=0\\ j\neq p}}^{n}\sum_{d_j\in R^v(C_j)} E^v_{C_j}(c_j)\prod_{\substack{l=0\\ l\neq j\\ l\neq p}}^{n} m^v_l$$

$$- \sum_{\substack{(d_0,\ldots,d_n)\in\sigma_v\\ d_p=c_p}} \ln P(C_0=d_0 \mid \ldots, C_l=d_l,\ldots)$$

$$= \sum_{\substack{(d_0,\ldots,d_n)\in\sigma_v\\ d_p=c_p}} \ln P(C_0=d_0 \mid \ldots, C_l=d_l,\ldots) - \frac{n\xi^v(T)}{(n+1)m^v_p}$$

$$+ \sum_{\substack{j=0\\ j\neq p}}^{n}\sum_{d_j\in R^v(C_j)} \frac{1}{m^v_p}\left[\sum_{\substack{(e_0,\ldots,e_n)\in\sigma_v\\ e_j=d_j}} \ln P(C_0=e_0 \mid \ldots, C_l=e_l,\ldots)\right.$$

$$\left. - \frac{n\xi^v(T)}{(n+1)m^v_j}\right] - \sum_{\substack{(d_0,\ldots,d_n)\in\sigma_v\\ d_p=c_p}} \ln P(C_0=d_0 \mid \ldots, C_l=d_l,\ldots)$$

$$= -\frac{n\xi^v(T)}{(n+1)m^v_p} + \frac{1}{m^v_p}\sum_{\substack{j=0\\ j\neq p}}^{n}\left[\xi^v(T) - \frac{n\xi^v(T)}{n+1}\right]$$

$$= -\frac{n\xi^v(T)}{(n+1)m^v_p} + \frac{n\xi^v(T)}{m^v_p} - \frac{n^2\xi^v(T)}{(n+1)m^v_p}$$

$$= -\frac{n\xi^v(T)}{(n+1)m^v_p} + \frac{n(n+1)\xi^v(T)}{(n+1)m^v_p} - \frac{n^2\xi^v(T)}{(n+1)m^v_p}$$

$$= -\frac{n\xi^v(T)}{(n+1)m^v_p} + \frac{n^2\xi^v(T)}{(n+1)m^v_p} + \frac{n\xi^v(T)}{(n+1)m^v_p} - \frac{n^2\xi^v(T)}{(n+1)m^v_p}$$

$$= 0.$$

Thus, we have an extreme point. □

5. ILP Formulation for multiple LPFs

In this section, we consider an integer linear programming approach for determining the best multiple LPFs. In particular, we consider the case where we are searching for the best two-LPF partitioning.

Our approach will be to map the problem into a mixed integer linear programming problem. First, we must define the variables we will use in our formulation. For each C_i involved in the current table, we associate a 0–1 valued variable Q_{C_i} representing whether $R(C_i)$ is to be split into two disjoint sets. We make the observation that for a two partition contiguous solution, exactly one of the $R(C_i)$'s will be divided into two subsets.

Since we will have two LPFs, say S_1 and S_2, let $S_{C_i}(c_i)$ be an integer variable representing which LPF $C_i = c_i$ is associated to. 0 implies S_1, 1 implies S_2, and 2 implies that it's associate to both. Let $E^1_{C_i}(c_i)$ and $E^2_{C_i}(c_i)$ represent the encoder variables for S_1 and S_2, respectively.

Let $d(P(x), S_j(x))$ where $x = (c_0, \ldots, c_n) \in T$ be a real variable representing the distance or error in LPF S_j on entry x. Since S_j is not applied to those entries for it's counterpart LPF, this variable is 0. Otherwise, it's value is $|-\ln P(x) + \sum_{c_i \in x} E_{C_i}(c_i)|$. Let $D(P(x), S_j(x))$ be a 0–1 variable representing that we must consider the above distance measure if x is supposed to be approximated by S_j.

Now, with the variables defined, we now consider the constraints. If $Q_{C_i} = 0$, then $R(C_i)$ is not split which further implies that $S_{C_i}(c_i) = 2$ for all $c_i \in R(C_i)$:

$$S_{C_i}(c_i) \geqslant 2 - 2Q_{C_i}. \tag{43}$$

Otherwise, it must be either 0 or 1:

$$S_{C_i}(c_i) \leqslant 2 - Q_{C_i}. \tag{44}$$

For the distance variables, if $Q_{C_i} = 1$, then $S_{C_i}(c_i)$ will determine which LPF is to be used:

$$D(P(x), S_1(x)) \leqslant 2(1 - Q_{C_i}) + (1 - S_{C_i}(c_i), \tag{45}$$

$$D(P(x), S_2(x)) \leqslant S_{C_i}(c_i), \tag{46}$$

$$D(P(x), S_1(x)) + D(P(x), S_2(x)) = 1. \tag{47}$$

If $D(P(x), S_j(x)) = 0$, then $d(P(x), S_j(x)) = 0$.

$$d(P(x), S_j(x)) \leqslant M D(P(x), S_j(x)), \tag{48}$$

where M is some arbitrarily large constant. Otherwise, $d(P(x), S_j(x)) = |-\ln P(x) + \sum_{c_i \in x} E_{C_i}(c_i)|$.

$$d(P(x), S_j(x)) \geqslant -\ln P(x) + \sum_{c_i \in x} E_{C_i}(c_i) - M\big(1 - D(P(x), S_j(x))\big), \tag{49}$$

$$d(P(x), S_j(x)) \geqslant -\sum_{c_i \in x} E_{C_i}(c_i) + \ln P(x) - M\big(1 - D(P(x), S_j(x))\big). \tag{50}$$

Since only one of $R(C_i)$ will be split,

$$\sum_{i=0}^{n} Q_{C_i} = 1. \tag{51}$$

Finally, the objective function we wish to minimize is

$$\sum_{x \in T} d(P(x), S_j(x)). \tag{52}$$

If there exists a 2-LPF perfect fit for T, solving the above mixed integer programming problem will find it. We do note though that in the case when such a perfect fit does not exist, our optimal solution is based on a slightly different metric from the original problem. However, any solution generated by this approach should be sufficient for most purposes.

Finally, we can generalize this formulation to more than 2 LPFs by encoding the characteristic functions into the problem.

References

[1] G.F. Cooper, Probabilistic inference using belief networks is NP-hard, Technical Report KSL-87-27, Medical Computer Science Group, Stanford University (1987).

[2] P. Dagum and M. Luby, Approximating probabilistic inference in Bayesian belief networks is NP-hard, *Artificial Intelligence* 60(1) (1993) 141–153.

[3] S. Geman and D. Geman, Stochastic relaxation, gibbs distribution, and the bayesian restoration of images, *IEEE Transactions on Pattern Analysis and Machine Intelligence* 6 (1984) 721–741.

[4] D. Heckerman, *Probabilistic Similarity Networks* (MIT Press, 1991).

[5] J. Kirman, A. Nicholson, M. Lejter, E. Santos, Jr. and T. Dean, Using goals to find plans with high expected utility, in: *Proceedings of the Second European Workshop on Planning* (1993).

[6] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference* (Morgan-Kaufmann, San Mateo, CA, 1988).

[7] Y. Peng and J.A. Reggia, Plausibility of diagnostic hypothese: The nature of simplicity, in: *Proceedings of the AAAI Conference* (1986) pp. 140–147.

[8] E. Santos, Jr., On spline approximations for bayesian computations, Technical Report CS-92-47, Department of Computer Science, Brown University (1992).

[9] E. Santos, Jr., Efficient jumpstarting of hill-climbing search for the most probable explanation, in: *Proceedings of International Congress on Computer Systems and Applied Mathematics Workshop on Constraint Processing* (1993) pp. 183–194.

[10] E. Santos, Jr., A fast hill-climbing approach without an energy function for finding mpe, in: *Proceedings of the 5th IEEE International Conference on Tools with Artificial Intelligence* (1993).

[11] E. Santos, Jr., A linear constraint satisfaction approach to cost-based abduction, *Artificial Intelligence* 65(1) (1994) 1–28.

[12] E. Santos, Jr. and S.E. Shimony, Belief updating by enumerating high-probability independence-based assignments, in: *Proceedings of the Conference on Uncertainty in Artificial Intelligence* (1994) pp. 506–513.

[13] R.D. Shachter, Evaluating influence diagrams, *Operations Research* 36 (1986) 871–882.
[14] S.E. Shimony, The role of relevance in explanation, I: Irrelevance as statistical independence, *International Journal of Approximate Reasoning* (June 1993).
[15] S.E. Shimony, Finding MAPs for belief networks is NP-hard, *Artificial Intelligence* 68 (1994) 399–410.
[16] S.E. Shimony, The role of relevance in explanation, II: Disjunctive assignments and approximate independence, to appear in *International Journal of Approximate Reasoning*.
[17] S. Srinivas, A generalization of the noisy-OR model, in: *Proceedings of the Conference on Uncertainty in Artificial Intelligence* (1993) pp. 208–215.
[18] B.K. Sy, Reasoning mpe to multiply connected belief networks using message passing, in: *Proceedings of the Tenth National Conference on Artificial Intelligence* (1992).