A New Look at the Generalized Distributive Law

Payam Pakzad, *Student Member, IEEE,* and Venkat Anantharam, *Fellow, IEEE*

***Abstract*—In this paper, we develop a measure-theoretic version of the junction tree algorithm to compute desired marginals of a product function. We reformulate the problem in a measure-theoretic framework, where the desired marginals are viewed as corresponding conditional expectations of a product of random variables. We generalize the notions of independence and junction trees to collections of σ-fields on a space with a signed measure. We provide an algorithm to find such a junction tree when one exists. We also give a general procedure to augment the σ-fields to create independencies, which we call "lifting." This procedure is the counterpart of the moralization and triangulation procedure in the conventional generalized distributive law (GDL) framework, in order to guarantee the existence of a junction tree. Our procedure includes the conventional GDL procedure as a special case. However, it can take advantage of structures at the *atomic* level of the sample space to produce junction tree-based algorithms for computing the desired marginals that are less complex than those GDL can discover, as we argue through examples. Our formalism gives a new way by which one can hope to find low-complexity algorithms for marginalization problems.**

***Index Terms*—Belief propagation, conditional independence, generalized distributive law (GDL), graphical models, iterative decoding, junction tree, signed measures.**

I. Introduction

LOCAL message-passing algorithms on graphs have seen a resurgence of interest in the communications and coding communities because of the success of the recently invented turbo codes [1] and the older low-density parity-check (LDPC) codes [2] which use such decoding algorithms. They have also long been of interest to the artificial intelligence community [3]. A general framework for describing such algorithms was described by Shafer and Shenoy [4]. Aji and McEliece [5] gave an equivalent (and for our purposes, slightly more convenient) framework known as the generalized distributive law (GDL) to describe such algorithms. The Viterbi algorithm, the Bahl–Cocke–Jelinek–Raviv (BCJR) algorithm [6], belief propagation algorithms [7], and fast Fourier transform (FFT) over a finite field are among the prominent implementations of the junction tree algorithm or GDL.[1]

Local message-passing algorithms such as GDL aim to find the desired set of marginals of a product function, whose component functions will be called "local kernels" in the sequel. Given such a problem, GDL makes use of the *distributivity* of the "product"' operation over "summation" in an underlying *semiring* to reduce the complexity of the required calculations. In many cases, this translates to substantial savings over brute-force computation. As we shall see in this paper, sometimes there is more structure available in the local kernels than the conventional way of thinking about GDL can discover. This is because GDL relies solely on the notion of variables. Any structure at a finer level than that of variables will be ignored by GDL. We illustrate these limitations in the following simple example.

Example 1: Let X and Y be arbitrary real functions on $\{1,\ldots,n\}$. Let $\mu(i,j)$ be a fixed real weight function for $i,j \in \{1,\ldots,n\}$, given by an $n \times n$ matrix M with $\mu(i,j) = M_{i,j}$. We would like to calculate the weighted average of $X \cdot Y$

$$E = \sum_{i=1}^{n}\sum_{j=1}^{n} X(i)Y(j)\mu(i,j).$$

The general GDL-type algorithm (assuming no structure on the weight function μ) will suggest the following:

$$E = \sum_{i=1}^{n} X(i) \sum_{j=1}^{n} Y(j)\mu(i,j)$$

requiring $n(n+1)$ multiplications and $(n-1)(n+1)$ additions. But this is not always the simplest way to calculate E.

Consider a "luckiest" case when the matrix M has rank 1, i.e., the weight function $\mu(i,j)$ factors as $f_1(i) \cdot f_2(j)$. In this case

$$E = \left(\sum_{i=1}^{n} X(i)f_1(i)\right)\left(\sum_{j=1}^{n} Y(j)f_2(j)\right)$$

requiring only $2n+1$ multiplications and $2n-2$ additions.

Suppose next that $\mu(i,j)$ does not factor as above, but the matrix M has a low rank of 2, so that $\mu(i,j) = f_1(i)f_2(j) + g_1(i)g_2(j)$. Then we can compute E as follows:

$$E = \left(\sum_{i=1}^{n} X(i)f_1(i)\right)\left(\sum_{j=1}^{n} Y(j)f_2(j)\right) + \left(\sum_{i=1}^{n} X(i)g_1(i)\right)\left(\sum_{j=1}^{n} Y(j)g_2(j)\right).$$

This requires $4n+2$ multiplications and $4n-4$ additions.

Manuscript received November 26, 2001; revised September 22, 2002. This work was supported by ONR/MURI under Grant N00014-1-0637, the National Science Foundation under Grants ECS-9873086 and ECS-0123512, and by EPRI/DOD under Grant EPRI-W08333-04.

The authors are with the Department of Electrical Engineering and Computer Science, University of California at Berkeley, Berkeley, CA 94720 USA (e-mail: payamp@eecs.berkeley.edu; ananth@eecs.berkeley.edu).

Communicated by R. Koetter, Associate Editor for Coding Theory.
Digital Object Identifier 10.1109/TIT.2004.828058

[1]Throughout this paper we will use both names—*GDL* and the *junction tree algorithm*—interchangeably, but will use the GDL notation from [5] to show connections with this work.

Next suppose that the fixed-weight matrix M is sparse, for example, with $\mu(i,j) = m_i 1(i+j=n+1)$, where $1(\cdot)$ is the indicator function. Then

$$E = \sum_{i=1}^{n} m_i X(i) Y(n+1-i)$$

requiring only $2n$ multiplications and $n-1$ additions.

It would be nice to have an automatic procedure to discover the existence of such complexity-reducing opportunities. The goal of this paper is to develop such a procedure. □

Note that in each case in Example 1, some (manual) introduction of hidden variables and/or redefinition of functions would allow the conventional GDL to also discover the best method of calculation. However, we do not consider this preprocessing phase as part of the GDL treatment. Our aim is to develop an *automatic* procedure that aims to discover such potentially useful structures in the data. We introduce a measure-theoretic framework which goes beyond the focus on "variables" to represent the states of the data. With the conventional GDL approach, once one chooses the set of variables to represent the state space, the consequent attempts to create independencies in order to find a junction tree (i.e., the moralization and triangulation procedure) are confined to working with that chosen set of variables. The primary advantage of our reformulation is to get rid of this restriction. The alternative we provide to the moralization and triangulation procedure, which we call "lifting," *automatically* discovers a way to exploit structure that is not aligned to the variables, which the usual approach is unable to discover. For instance, in Example 1, we can automatically discover the advantage of the low rank of matrix M.

Our measure-theoretic framework replaces GDL's concept of "local domains" with σ-fields in an appropriate sample space. We also replace GDL's "local kernels" with random variables measurable with respect to the corresponding σ-fields. The problem of finding the marginal with respect to a local domain is then naturally replaced by that of taking the conditional expectation given the corresponding σ-field. Our formalism includes the conventional GDL as a special case, in the sense that any junction tree that moralization and triangulation can produce in the conventional GDL can also be discovered using our framework. Although our results are generalizable to an arbitrary *semifield*,[2] we focus on the sum-product algebra in order to avoid abstract distractions.

Here is an outline of this paper. In Section II, we review the GDL algorithm. In Section III, we develop the necessary concepts to work with the independence of σ-fields on a space supporting a signed measure. In Section IV, we reformulate the marginalization problem in our framework, define a notion of junction tree on a collection of σ-fields, and provide a message-passing algorithm analogous to the GDL algorithm to solve this marginalization problem with linear complexity. We also provide an algorithm to find a junction tree on a given collection of σ-fields when one exists. Just as is the case with the ordinary GDL formulation, junction trees do not always exist. In Section V,we discuss a method to construct a junction tree on the augmented σ-fields, which can be used to solve the original marginalization problem with low complexity. This procedure replaces the moralization and triangulation procedure of the conventional GDL, and offers a strictly larger set of alternatives than the GDL does. A discussion of the computational complexity of our methods is presented in Section VI. Several examples and applications are given in Section VII. Of particular interest is the observation that the minimal complexity trellis-based decoding algorithm for linear block codes can be very naturally described as an instance of lifting in our framework. In Section VIII, we further discuss and summarize our results.

[2]A semifield is an algebraic structure with addition and multiplication, both of which are commutative and associative and have identity element. Further, multiplication is distributive over addition, and every nonzero element has a multiplicative inverse. Such useful algebras as the sum-product and the max-sum are examples of semifields (see [8]).

II. GDL Algorithm

Definition 1: A (commutative) *semiring* R is a set with operations $+$ and $\times$ such that both $+$ and $\times$ are commutative and associative and have identity elements in R (0 and 1, respectively), and $\times$ is distributive over $+$. □

Let $\{x_1, \ldots, x_n\}$ be variables taking values in sets $\{A_1, \ldots, A_n\}$, respectively. Let $\{S_1, \ldots, S_N\}$ be a collection of subsets of $\{1, \ldots, n\}$, and for $i \in \{1, \ldots, N\}$, let $\alpha_i : A_{S_i} \to R$ be a function of x_{S_i}, taking value in some *semiring* R. The "marginalize a product function" (MPF) problem is to find, for one or more of the indexes $i = 1, \ldots, N$, the S_i-marginalization of the product of the α_i's, i.e.,

$$\beta_i(x_{S_i}) := \sum_{x_{S_i^c} \in A_{S_i^c}} \left(\prod_{j=1}^{N} \alpha_j(x_{S_j}) \right).$$

In the language of GDL, α_i's are called the *local kernels*, and the variable lists x_{S_i} are called the *local domains*.

The GDL algorithm gives a message-passing solution to the MPF problem when the sets $\{S_1, \ldots, S_N\}$ can be organized into a *junction tree*. A junction tree is a tree with nodes corresponding to $\{S_1, \ldots, S_N\}$, and with the property that the subgraph on the nodes that contain any variable x_i is connected. An equivalent condition is that if A, B, and C are subsets of $\{1, \ldots, N\}$ such that A and B are separated by C on the tree, then $S_A \cap S_B \subseteq S_C$ where $S_A : \bigcup_{i \in A} S_i$. As we will see in Section IV, our definition of a junction tree will resemble this latter definition.

Suppose $\boldsymbol{G}$ is a junction tree on nodes $\{1, \ldots, N\}$ with local kernels $\{\alpha_1, \ldots, \alpha_N\}$. Let $\{E_1, \ldots, E_T\}$ be a message-passing *schedule*, viz. the "message" along the (directed) edge (i,j) of the graph is updated at time t iff $(i,j) \in E_t$. The following asynchronous message-passing algorithm (GDL) will solve the MPF problem.

Algorithm 1: At each time t and for all pairs (i,j) of neighboring nodes in the graph, let the "message" from i to j be a function $\mu_{i,j}^t : A_{S_i \cap S_j} \longrightarrow R$. Initialize all messages to 1. At each time $t \in \{1, \ldots, T\}$, if the edge $(i,j) \in E_t$ then update the message from node i to j as follows:

$$\mu_{i,j}^t(x_{S_i \cap S_j}) = \sum_{x_{S_i \setminus S_j} \in A_{S_i \setminus S_j}} \alpha_i(x_{S_i}) \prod_{k \in N_i \setminus \{j\}} \mu_{k,i}^{t-1}(x_{S_k \cap S_i}) \tag{1}$$

where N_i is the set of neighbors of i in $\boldsymbol{G}$.

This algorithm will converge in finite time, at which time we have

$$\beta_i(x_{S_i}) = \alpha_i(x_{S_i}) \prod_{k \in N_i} \mu_{k,i}(x_{S_k \cap S_i}). \tag{2}$$

Proof: See [5]. □

III. Preliminaries

Let $(\Omega, \mathcal{M})$ be a discrete measurable space, i.e., Ω is a finite or countable set and $\mathcal{M}$ is a σ-field on Ω. Let $\mu : \mathcal{M} \to (-\infty, \infty)$ be a *signed measure* on $(\Omega, \mathcal{M})$, i.e., $\mu(\emptyset) = 0$ and for any sequence $\{a_i\}_{i=1}^{\infty}$ of disjoint sets in $\mathcal{M}$

$$\mu\left(\bigcup_1^{\infty} a_i\right) = \sum_1^{\infty} \mu(a_i)$$

where the infinite sum is implicitly assumed to exist. Then $(\Omega, \mathcal{M}, \mu)$ is called a *measure space*. As a matter of notation, we usually write $\mu(a_1, a_2, \ldots, a_n)$ for $\mu(a_1 \cap a_2 \cap \cdots \cap a_n)$. Also, given events a and b with $\mu(b) \neq 0$, we write $\mu(a \mid b)$ for the ratio $\frac{\mu(a,b)}{\mu(b)}$.

Let $\mathcal{F}, \mathcal{G}, \mathcal{H}$ and $\{\mathcal{F}_1, \ldots, \mathcal{F}_M\}$ be sub-σ-fields of $\mathcal{M}$.

Atoms of a $\boldsymbol{\sigma}$-Field: We define the set of *atoms* of $\mathcal{F}$ to be the collection of the minimal nonempty measurable sets in $\mathcal{F}$ with respect to (w.r.t.) inclusion

$$\mathcal{A}(\mathcal{F}) := \left\{ f \in \mathcal{F} : f \neq \emptyset \text{ and } \forall\, g \in \mathcal{F},\ f \cap g \in \{\emptyset, f\} \right\}.$$

Augmentation of σ-Fields: We denote by $\mathcal{F} \vee \mathcal{G}$ the span of $\mathcal{F}$ and $\mathcal{G}$, i.e., the smallest σ-field containing both $\mathcal{F}$ and $\mathcal{G}$. For a set A of indexes, we write $\mathcal{F}_A$ for $\bigvee_{i \in A} \mathcal{F}_i$, with $\mathcal{F}_\emptyset := \{\emptyset, \Omega\}$, the trivial σ-field on Ω. Note that the atoms of $\mathcal{F} \vee \mathcal{G}$ are all in the form $f \cap g$ for some $f \in \mathcal{A}(\mathcal{F})$ and $g \in \mathcal{A}(\mathcal{G})$.

Conditional Independence: We say $\mathcal{F}$ is conditionally independent of $\mathcal{G}$ given $\mathcal{H}$ and write $\mathcal{F} \perp\!\!\!\perp \mathcal{G} \,\big|\, \mathcal{H}$ w.r.t. μ when for any *atom* h of $\mathcal{H}$

- if $\mu(h) = 0$ then $\forall f \in \mathcal{F}, g \in \mathcal{G},\ \mu(f, g, h) = 0$;
- if $\mu(h) \neq 0$ then $\forall f \in \mathcal{F}, g \in \mathcal{G}$

$$\mu(f, g, h)\mu(h) = \mu(f, h)\mu(g, h).$$

.

When the underlying measure is obvious from the context, we omit the explicit mention of it.

Similarly, we say a collection $\{\mathcal{F}_1, \ldots, \mathcal{F}_M\}$ of sub-σ-fields are mutually conditionally independent given $\mathcal{H}$, and write

$$\mathcal{F}_1 \perp\!\!\!\perp \mathcal{F}_2 \perp\!\!\!\perp \cdots \perp\!\!\!\perp \mathcal{F}_M | \mathcal{H}$$

if $\mathcal{F}_i \perp\!\!\!\perp \bigvee_{j \neq i} \mathcal{F}_j \,\big|\, \mathcal{H}$ for all $i \in \{1, \ldots, M\}$.

Independence: We say $\mathcal{F}$ is independent of $\mathcal{G}$ or $\mathcal{F} \perp\!\!\!\perp \mathcal{G}$ w.r.t. μ when $\mathcal{F} \perp\!\!\!\perp \mathcal{G} \,\big|\, \{\emptyset, \Omega\}$.

Note that these definitions are consistent with the usual definitions of independence when μ is a probability measure.

Expectation and Conditional Expectation: Let $\mathcal{F}$ and $\mathcal{G}$ be a σ-field with sets of atoms $\mathcal{A}(\mathcal{F})$ and $\mathcal{A}(\mathcal{G})$, respectively. A *partially defined* random variable X in $\mathcal{F}$ is a partially defined function on Ω, where for each r in the range of X, $X^{-1}(r)$ is measurable in $\mathcal{F}$. We write $X \in \mathcal{F}$, and denote by $\mathcal{A}_X(\mathcal{F})$ the subset of $\mathcal{A}(\mathcal{F})$ where X is defined. We also denote by $\mathcal{A}'(\mathcal{F})$ the set of atoms of $\mathcal{F}$ with nonzero measure: $\mathcal{A}'(\mathcal{F}) := \{f \in \mathcal{A}(\mathcal{F}) : \mu(f) \neq 0\}$.

Assuming $\mu(\Omega) \neq 0$ and that the sum exists, we define the *expectation* of X as

$$\begin{aligned} \boldsymbol{E}[X] &:= \frac{1}{\mu(\Omega)} \sum_{f \in \mathcal{A}_X(\mathcal{F})} X(f)\mu(f) \\ &= \sum_{f \in \mathcal{A}_X(\mathcal{F})} X(f)\mu(f|\Omega). \end{aligned} \tag{3}$$

When the sum exists, we define the *conditional expectation* of X given $\mathcal{G}$, as a partially defined random variable Y in $\mathcal{G}$, defined on $\mathcal{A}_Y(\mathcal{G}) = \mathcal{A}'(\mathcal{G})$, as

$$\begin{aligned} \boldsymbol{E}[X|\mathcal{G}](g) &:= \frac{1}{\mu(g)} \sum_{f \in \mathcal{A}_X(\mathcal{F})} X(f)\mu(g, f), \quad \forall\, g \in \mathcal{A}'(\mathcal{G}) \\ &= \sum_{f \in \mathcal{A}_X(\mathcal{F})} X(f)\mu(f|g), \quad \forall\, g \in \mathcal{A}'(\mathcal{G}). \end{aligned} \tag{4}$$

It is important to note that (3) and (4) should not be taken at face value, as prescribing the way to carry out the summation; in many cases, such as the ones in Example 1, the calculation can be simplified. We will address this in detail in Section VI-A in the context of our general method to create independencies.

The signed conditional independence relation satisfies certain properties (inference rules) that we now state. See [7] and [3] for discussion of inference rules for the case when μ is a probability measure.[3]

Theorem 1: Let $\mathcal{F}, \mathcal{G}, \mathcal{X}, \mathcal{Y}$ be σ-fields. Then the following properties hold:

Symmetry:

$$\mathcal{F} \perp\!\!\!\perp \mathcal{G} \,\big|\, \mathcal{X} \implies \mathcal{G} \perp\!\!\!\perp \mathcal{F} \,\big|\, \mathcal{X}. \tag{5a}$$

Decomposition:

$$\mathcal{F} \perp\!\!\!\perp \mathcal{G} \vee \mathcal{X} \,\big|\, \mathcal{Y} \implies \mathcal{F} \perp\!\!\!\perp \mathcal{G} \,\big|\, \mathcal{Y} \;\&\; \mathcal{F} \perp\!\!\!\perp \mathcal{X} \,\big|\, \mathcal{Y}. \tag{5b}$$

Contraction:

$$\mathcal{F} \perp\!\!\!\perp \mathcal{G} \,\big|\, \mathcal{X} \;\&\; \mathcal{F} \perp\!\!\!\perp \mathcal{Y} \,\big|\, \mathcal{G} \vee \mathcal{X} \implies \mathcal{F} \perp\!\!\!\perp \mathcal{G} \vee \mathcal{Y} \,\big|\, \mathcal{X}. \tag{5c}$$

Other properties:

$$\mathcal{F} \perp\!\!\!\perp \mathcal{G} \,\big|\, \mathcal{X} \;\&\; \mathcal{F} \vee \mathcal{G} \perp\!\!\!\perp \mathcal{Y} \,\big|\, \mathcal{X} \implies \mathcal{F} \perp\!\!\!\perp \mathcal{G} \vee \mathcal{Y} \,\big|\, \mathcal{X} \tag{5d}$$

$$\mathcal{F} \perp\!\!\!\perp \mathcal{G} \,\big|\, \mathcal{X} \;\&\; \mathcal{F} \vee \mathcal{X} \perp\!\!\!\perp \mathcal{Y} \,\big|\, \mathcal{G} \implies \mathcal{F} \perp\!\!\!\perp \mathcal{G} \vee \mathcal{Y} \,\big|\, \mathcal{X} \tag{5e}$$

$$\begin{aligned} &\mathcal{F} \perp\!\!\!\perp \mathcal{G} \vee \mathcal{X} \,\big|\, \mathcal{Y} \;\&\; \mathcal{F} \vee \mathcal{G} \perp\!\!\!\perp \mathcal{Y} \,\big|\, \mathcal{X} \\ &\qquad \implies \mathcal{F} \perp\!\!\!\perp \mathcal{G} \vee \mathcal{Y} \,\big|\, \mathcal{X} \;\&\; \mathcal{F} \vee \mathcal{Y} \perp\!\!\!\perp \mathcal{G} \,\big|\, \mathcal{X}. \end{aligned} \tag{5f}$$

Proof: See Appendix A. □

IV. Probabilistic MPF and Junction Trees

We now formulate a probabilistic version of the MPF problem and introduce the corresponding concept of *junction trees*. We also describe a probabilistic version of the GDL algorithm to solve this MPF problem.

[3]Note that the signed conditional independence relation satisfies *symmetry, decomposition,* and *contraction,* but in general *weak union* does not hold. So this relation is not a *semi-graphoid*.

Throughout this paper, let $(\Omega, \mathcal{M}, \mu)$ be a measure space, $\{\mathcal{F}_1, \ldots, \mathcal{F}_M\}$ be sub-σ-fields of $\mathcal{M}$, and let $\{X_1, \ldots, X_M\}$ be a collection of partially defined random variables with $X_i \in \mathcal{F}_i$. We will speak of the measure space $(\Omega, \{\mathcal{F}_1, \ldots, \mathcal{F}_M\}, \mu)$ since the choice of $\mathcal{M}$ will not be relevant.

Probabilistic MPF Problem: For one or more $i \in \{1, \ldots, M\}$, find $Y_i := \boldsymbol{E}[\prod_j X_j | \mathcal{F}_i]$, the conditional expectation of the product, given $\mathcal{F}_i$. □

Given a conventional MPF problem, one can choose a subset of local functions whose product is viewed as a measure function in our framework, and the other local functions can be viewed as random variables, each measurable w.r.t. the σ-field defined by the variables comprising its local domain. Then for a configuration x_{S_i} of the variables corresponding to a local domain S_i, we have

$$\boldsymbol{E}\left[\prod_{j=1}^{M} X_j \middle| \mathcal{F}_i\right](x_{S_i}) = \frac{1}{\mu(x_{S_i})} \sum_{x_{S_i^c} \in A_{S_i^c}} \left(\prod_{j=1}^{N} \alpha_j(x_{S_j})\right) \propto \beta_i(x_{S_i}) \quad (6)$$

where, out of the N local functions, the first M were treated as random variables, and the last $N - M$ have been relegated to the measure, so that

$$\mu(x_S) = \prod_{k=M+1}^{N} \alpha_k(x_{S_k})$$

where $S := \{1, \ldots, n\}$ is the index set of the variables. Equation (6) shows that solving the probabilistic MPF problem with this setup will in effect amount to a solution to the conventional MPF problem.

In most applications, for a family of MPF problems the local kernels can be categorized as either *fixed* or *arbitrary*. For example, in an LDPC decoding problem, the code itself is fixed, so the local kernels at the check nodes are fixed; we only receive new observations and try to find the most likely codeword given each observation set. As another example, when finding the Hadamard transform

$$\sum_{x_1, \ldots, x_n} \prod_{i=1}^{n} (-1)^{x_i y_i} f(x_1, \ldots, x_n)$$

of an arbitrary function f, the functions $(-1)^{x_i y_i}$ are fixed. Typically, we want to assign (some of) the fixed kernels as the measure function, and the arbitrary kernels as the marginalizable random variables; this way, once a junction tree has been found for one problem, it can be used to marginalize the product of any arbitrary collection of random variables measurable in the same σ-fields. See Section VII for more examples.

We define junction trees as follows.

Definition 2: Let $\boldsymbol{G}$ be a tree with nodes $\{1, \ldots, M\}$. We say subsets A and B of $\{1, \ldots, M\}$ are*separated* by a node i if $\forall x \in A$, $y \in B$, the path from x to y contains i. Then we call $\boldsymbol{G}$ a *junction tree* if $\forall A, B \subset \{1, \ldots, M\}$ and $i \in \{1, \ldots, M\}$ so that (s.t.) i separates A and B on the tree, we have $\mathcal{F}_A \perp\!\!\!\perp \mathcal{F}_B \,|\, \mathcal{F}_i$. □

A. Probabilistic Junction Tree Algorithm

Suppose $\boldsymbol{G}$ is a junction tree with nodes labeled by σ-fields $\{\mathcal{F}_1, \ldots, \mathcal{F}_M\}$ as defined above, and let $\{X_1, \ldots, X_M\}$ be random variables with $X_i \in \mathcal{F}_i$. Then the following message-passing algorithm will solve the probabilistic MPF problem.

Algorithm 2: For each edge (i, j) on the graph, define a "message" $Y_{i,j}$ from node i to j as a partially defined random variable measurable w.r.t. $\mathcal{F}_j$, initialized to 1.

For each $i = 1, \ldots, M$, let N_i denote the set of neighbors of i on $\boldsymbol{G}$, and for each edge (i, j) in the tree, define $N_{i,j} = N_i \backslash \{j\}$.

For each edge (i, j) in the tree, update the message $Y_{i,j}$ (asynchronously) as

$$Y_{i,j} = \boldsymbol{E}\left[X_i \prod_{k \in N_{i,j}} Y_{k,i} \middle| \mathcal{F}_j\right]. \quad (7)$$

This algorithm will converge in finite time, at which time we have

$$Y_i = X_i \prod_{k \in N_i} Y_{k,i} \quad (8)$$

where as before

$$Y_i := \boldsymbol{E}\left[\prod_{j=1}^{M} X_j \middle| \mathcal{F}_i\right]$$

is the objective random variable.

Proof: See Appendix B. □

Note that message $Y_{i,j}$ can be viewed as a function $Y_{i,j}(f_j)$ on $\mathcal{A}'(\mathcal{F}_j)$, where f_j ranges over $\mathcal{A}'(\mathcal{F}_j)$. Then the update rule (7) can be rewritten as

$$Y_{i,j}(f_j) = \sum_{f_i \in \mathcal{A}'(\mathcal{F}_i)} \mu(f_i \mid f_j) X_i(f_i) \prod_{k \in N_{i,j}} Y_{k,i}(f_i). \quad (9)$$

Similarly, (8) can be rewritten as

$$Y_i(f_i) = X_i(f_i) \prod_{k \in N_i} Y_{k,i}(f_i). \quad (10)$$

One should note that rarely is the way suggested by (9) an efficient way to carry out the summation (we will discuss this further in Section VI-A). However, there may be some value to seeing (7) and (8) written as (9) and (10), since in some sense the language of σ-fields has been translated into one of "variables." However, it should be noted that the "variables" $f_j \in \mathcal{A}'(\mathcal{F}_j)$ for $j = 1, \ldots, M$ do not in any sense play the role of what are called variables in the original GDL algorithm.

We will discuss the computational complexity of Algorithm 2 in Section VI.

B. Existence of Junction Trees

In the conventional GDL formulation, given the local kernels $\{S_1, \ldots, S_N\}$ there is a rather simple way to determine whether a junction tree exists.

As described in [5, Sec. 4], we can define a local domain graph $G_{\rm LD}$ for the local domains. This is a weighted complete graph with N vertices, corresponding to the N local domains, with the weight of the edge (i,j) given by $w_{i,j} = |S_i \cap S_j|$. Then a junction tree exists on $\{S_1, \ldots, S_N\}$ iff $w_{\max} = \sum_{i=1}^{N} |S_i| - n$, where $w_{\max}$ is the weight of the maximum-weight spanning tree of $G_{\rm LD}$ (see [5]). In case equality holds, any maximum weight spanning tree of $G_{\rm LD}$ will be a junction tree. Therefore, the problem of existence of a junction tree can be solved using a greedy algorithm (such as Prim's algorithm or Kruskal's algorithms, see, e.g., [9]) to find a maximum-weight spanning tree of $G_{\rm LD}$.

In our measure-theoretic framework, on the other hand, this problem is not as simple. The reason is that in the GDL framework the conditional independence property is much easier to verify: local domains S_i and S_j are conditionally independent given S_k iff $S_i \cap S_j \subseteq S_k$. In our framework, on the other hand, the σ-fields in general may not be rectangular, σ-fields defined as those generated by certain underlying variables, making it harder to determine if conditional independencies exist.

In this section, we present an algorithm to determine whether a junction tree exists on the given σ-fields, and to find one if it does exist. We will prove results that will allow us to break down the problem of finding a junction tree into smaller problems and recursively build a junction tree from smaller trees.

Definition 3: A *valid partition* of $\{1, \ldots, M\} \backslash \{i\}$ with respect to a node i is a partition $\{p_1, \ldots, p_l\}$ of $\{1, \ldots, M\} \backslash \{i\}$ (i.e., $\bigcup_{j=1}^{l} p_j = \{1, \ldots, M\} \backslash \{i\}$ and $p_j \cap p_k = \emptyset$ for $j \neq k$) such that $\mathcal{F}_{p_j}$s are mutually conditionally independent, given $\mathcal{F}_i$. □

Definition 4: Let $P = \{p_1, \ldots, p_l\}$ be any partition of $\{1, \ldots, M\} \backslash \{i\}$. A tree with nodes $\{1, \ldots, M\}$ is called *compatible* with partition P at node i if its subtrees hanging from i correspond to the elements of P. □

Lemma 2: $\forall i \in \{1, \ldots, M\}$, there is a finest valid partition w.r.t. i, which we shall denote by P_i, such that every other valid partition w.r.t. i is a coarsening of P_i. Further, if p is an element of P_i and p is the disjoint union of nonempty sets e_1 and e_2, then $\mathcal{F}_{e_1} \not\perp\!\!\!\perp \mathcal{F}_{e_2} \,\big|\, \mathcal{F}_i$.

Proof: Suppose $A = \{p_1, \ldots, p_l\}$ and $B = \{q_1, \ldots, q_m\}$ are valid partitions w.r.t. node i. Now construct another partition, $C = \{p \cap q : p \in A \;\&\; q \in B\}$. We claim that C is also a valid partition w.r.t. i (finer than both A and B): To see this, we need to show that for each $d = p \cap q \in C$, $\mathcal{F}_d \perp\!\!\!\perp \mathcal{F}_{d^c} \,\big|\, \mathcal{F}_i$. Using simple manipulations like

$$\mathcal{F}_p = \mathcal{F}_{p \cap (q \cup q^c)} = \mathcal{F}_{p \cap q} \vee \mathcal{F}_{p \cap q^c}$$

we get

$$\mathcal{F}_{p \cap q} \vee \mathcal{F}_{p \cap q^c} \perp\!\!\!\perp \mathcal{F}_{p^c} \,\big|\, \mathcal{F}_i$$
$$\mathcal{F}_{p \cap q} \vee \mathcal{F}_{p^c \cap q} \perp\!\!\!\perp \mathcal{F}_{p \cap q^c} \vee \mathcal{F}_{p^c \cap q^c} \,\big|\, \mathcal{F}_i \Rightarrow \mathcal{F}_{p \cap q} \perp\!\!\!\perp \mathcal{F}_{p \cap q^c} \,\big|\, \mathcal{F}_i$$

where the last implication follows from (5b). And, finally, the last two relations and (5d) imply that $\mathcal{F}_{p \cap q} \perp\!\!\!\perp \mathcal{F}_{p^c \cup (p \cap q^c)} \,\big|\, \mathcal{F}_i$, and hence, $\mathcal{F}_{p \cap q} \perp\!\!\!\perp \mathcal{F}_{(p \cap q)^c} \,\big|\, \mathcal{F}_i$. So a finest valid partition w.r.t. i exists, whose atoms are the intersections of atoms of all the valid partitions w.r.t. i.

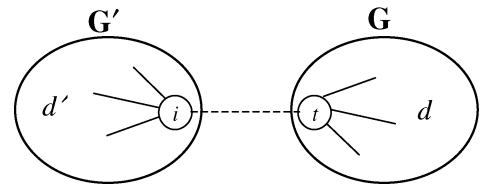

Fig. 1. Augmenting junction trees; Lemma 4.

Now suppose p is an element of P_i and p is the disjoint union of nonempty sets e_1 and e_2, and $\mathcal{F}_{e_1} \perp\!\!\!\perp \mathcal{F}_{e_2} \,\big|\, \mathcal{F}_i$. We also have $\mathcal{F}_{e_1} \vee \mathcal{F}_{e_2} \perp\!\!\!\perp \mathcal{F}_{p^c} \,\big|\, \mathcal{F}_i$. Then, from the last two relations and by (5d) we get $\mathcal{F}_{e_1} \perp\!\!\!\perp \mathcal{F}_{p^c} \vee \mathcal{F}_{e_2} \,\big|\, \mathcal{F}_i$, and hence $\mathcal{F}_{e_1} \perp\!\!\!\perp \mathcal{F}_{e_1^c} \,\big|\, \mathcal{F}_i$. Then e_1 and e_2 would be elements in a finer valid partition which is a contradiction. □

Lemma 3: Given $\{\mathcal{F}_1, \ldots, \mathcal{F}_M\}$, a tree with nodes $\{1, \ldots, M\}$ is a junction tree iff at each node i it is compatible with some valid partition of $\{1, \ldots, M\} \backslash \{i\}$ w.r.t. i.

Proof: Immediate from the definitions. □

Lemma 4: Let d be a subset of $\{1, \ldots, M\}$ and let d' be its complement in $\{1, \ldots, M\}$. Suppose there exist $t \in d$ and $i \in d'$ such that $\mathcal{F}_d \perp\!\!\!\perp \mathcal{F}_{d'} \,\big|\, \mathcal{F}_t$ and $\mathcal{F}_d \perp\!\!\!\perp \mathcal{F}_{d'} \,\big|\, \mathcal{F}_i$. Let $\boldsymbol{G}$ be any junction tree on d and $\boldsymbol{G}'$ any junction tree on d'. Then, the tree obtained by connecting $\boldsymbol{G}$ and $\boldsymbol{G}'$ by adding an edge between t and i is a junction tree on $\{1, \ldots, M\}$ (see Fig. 1).

Proof: Let x be any node that separates A and B on the resultant tree. We will show that $\mathcal{F}_A \perp\!\!\!\perp \mathcal{F}_B \,\big|\, \mathcal{F}_x$ and hence we have a junction tree.

Let $A_1 = A \cap d, A_2 = A \cap d', B_1 = B \cap d$, and $B_2 = B \cap d'$ and without loss of generality (WLOG) suppose $x \in d$. Then at least one of A_2 and B_2 must be empty, or else x would not separate A and B. Suppose $A_2 = \emptyset$.

First suppose $x = t$. Then we have

$$\mathcal{F}_{A_1} \perp\!\!\!\perp \mathcal{F}_{B_1} \,\big|\, \mathcal{F}_t \qquad \text{by j. t. property on } \boldsymbol{G}$$
$$\mathcal{F}_{A_1} \vee \mathcal{F}_{B_1} \perp\!\!\!\perp \mathcal{F}_{B_2} \,\big|\, \mathcal{F}_t \quad \text{since } A_1 \cup B_1 \subset d \text{ and } B_2 \subset d'.$$

So by (5d), we have $\mathcal{F}_{A_1} \perp\!\!\!\perp \mathcal{F}_{B_1} \vee \mathcal{F}_{B_2} \big| \mathcal{F}_t$, i.e., $\mathcal{F}_A \perp\!\!\!\perp \mathcal{F}_B \,\big|\, \mathcal{F}_t$ and we are done.

Next suppose $x \in d \backslash \{t\}$. Then we must also have that x separates A_1 from $B_1 \cup \{t\}$ (assuming WLOG that B_2 is nonempty.) Then

$$\mathcal{F}_{A_1} \perp\!\!\!\perp \mathcal{F}_{B_1} \vee \mathcal{F}_t \,\big|\, \mathcal{F}_x \tag{11}$$
$$\mathcal{F}_{A_1} \vee \mathcal{F}_x \vee \mathcal{F}_{B_1} \perp\!\!\!\perp \mathcal{F}_{B_2} \,\big|\, \mathcal{F}_t \tag{12}$$

where (12) holds since $A_1 \cup B_1 \cup \{x\} \subset d$ and $B_2 \subset d'$. We will show that $\mathcal{F}_{A_1} \perp\!\!\!\perp \mathcal{F}_{B_1} \vee \mathcal{F}_{B_2} \vee \mathcal{F}_t \,\big|\, \mathcal{F}_x$ and hence $\mathcal{F}_A \perp\!\!\!\perp \mathcal{F}_B \,\big|\, \mathcal{F}_x$.

Let $\chi, \tau, \alpha, \beta_1$, and β_2 be arbitrary atoms of $\mathcal{F}_x, \mathcal{F}_t, \mathcal{F}_A, \mathcal{F}_{B_1}$, and $\mathcal{F}_{B_2}$, respectively.

- *Case* $\mu(\chi) = \mu(\tau) = 0$. Then from (12), we have that $\mu(\alpha, \beta_1, \beta_2, \chi, \tau) = 0$, and so we are done.
- Case $\mu(\chi) = 0$ and $\mu(\tau) \neq 0$. Then from (12) we have

  $$\mu(\alpha, \beta_1, \beta_2, \chi, \tau) = \mu(\alpha, \beta_1, \chi, \tau)\mu(\beta_2, \tau)/\mu(\tau).$$

  But from (11), $\mu(\alpha, \beta_1, \chi, \tau) = 0$ since $\mu(\chi) = 0$. Thus, $\mu(\alpha, \beta_1, \beta_2, \chi, \tau) = 0$ and we are done.

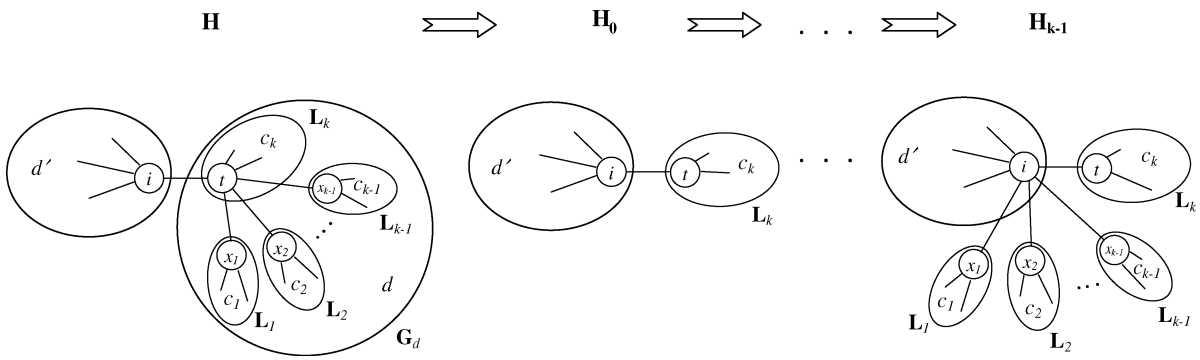

Fig. 2. Transformation of junction trees used in Theorem 5.

- Case $\mu(\chi) \neq 0$ and $\mu(\tau) = 0$. Then from (12) we have that

$$\mu(\alpha, \beta_1, \beta_2, \chi, \tau) = \mu(\beta_1, \beta_2, \chi, \tau) = 0$$

and so we have the equality $\mu(\alpha, \beta_1, \beta_2, \chi, \tau)\mu(\chi) = \mu(\alpha, \chi)\mu(\beta_1, \beta_2, \chi, \tau) = 0$ and we are done.

- Case $\mu(\chi) \neq 0$ and $\mu(\tau) \neq 0$. Then from (12)

$$\mu(\alpha, \beta_1, \beta_2, \chi, \tau) = \mu(\alpha, \beta_1, \chi, \tau)\mu(\beta_2, \tau)/\mu(\tau)$$

and from (11)

$$\mu(\alpha, \beta_1, \chi, \tau) = \mu(\alpha, \chi)\mu(\beta_1, \chi, \tau)/\mu(\chi).$$

Substituting the latter into the former, we obtain

$$\mu(\alpha, \beta_1, \beta_2, \chi, \tau) = \mu(\alpha, \chi)\mu(\beta_1, \chi, \tau)\mu(\beta_2, \tau)/(\mu(\chi)\mu(\tau)).$$

But by (12), $\mu(\beta_1, \chi, \tau)\mu(\beta_2, \tau)/\mu(\tau) = \mu(\beta_1, \beta_2, \chi, \tau)$, so

$$\mu(\alpha, \beta_1, \beta_2, \chi, \tau) = \mu(\alpha, \chi)\mu(\beta_1, \beta_2, \chi, \tau)/\mu(\chi)$$

and we are done. □

We now state our main theorem on the existence of junction trees.

Therem 5: Given a set of σ-fields $\{\mathcal{F}_1, \ldots, \mathcal{F}_M\}$, if there exists a junction tree on $\{1, \ldots, M\}$, then for every $i \in \{1, \ldots, M\}$ there exists a junction tree compatible with P_i, the finest valid partition w.r.t. i.

Notice that Theorem 5 and Lemma 4 give an algorithm to find a junction tree, when one exists, as we shall describe in Section IV-D.

Proof: The claim is trivial for $M \leq 3$. We will prove the theorem for $M > 3$ by induction.

Let $P_i = \{c_1, \ldots, c_l\}$ with $\bigcup_{j=1}^{l} c_j = \{1, \ldots, M\}\backslash\{i\}$ and $c_j \cap c_k \emptyset$ for $j \neq k$. Let $\boldsymbol{G}$ be a junction tree. Let $Q = \{d_1, \ldots, d_n\}$ be the partition of $\{1, \ldots, M\}\backslash\{i\}$ compatible with $\boldsymbol{G}$. Let $d = d_j$ be an arbitrary element of Q, and let $d' = \bigcup_{k \neq j} d_k \cup \{i\}$. Let $t = N_i \cup d$ be the node in d that is neighbor to i in tree $\boldsymbol{G}$. By Lemmas 2 and 3, d is the union of some of c_k's. WLOG, assume that $d = \bigcup_{k=1}^{K} c_k$ where $K \leq l$, and also assume that $t \in c_K$.

Then, from the junction tree property, we have

$$\mathcal{F}_i \perp\!\!\!\perp \mathcal{F}_d \,\big|\, \mathcal{F}_t. \tag{13}$$

Since $\boldsymbol{G}$ is a junction tree, the subtree on d is also a junction tree. Now $|d| < M$, and so by induction hypothesis there exists a junction tree on d compatible with P'_t, the finest valid partition w.r.t. t of $d\backslash\{t\}$.

Now we claim that $R = \{c_k\backslash\{t\} : 1 \leq k \leq K\}$ is a valid partition of $d\backslash\{t\}$ w.r.t. t. To see this, let $c = c_k$ for some arbitrary $k = 1, \ldots, K$, and let $c' = d\backslash\{c\}$ so $\mathcal{F}_d = \mathcal{F}_c \vee \mathcal{F}_{c'}$. But one of c and c' contains t. Then by the properties of valid partition w.r.t. i, we have

$$\mathcal{F}_c \vee \mathcal{F}_t \perp\!\!\!\perp \mathcal{F}_{c'} \,\big|\, \mathcal{F}_i \qquad \text{or} \qquad \mathcal{F}_c \perp\!\!\!\perp \mathcal{F}_{c'} \vee \mathcal{F}_t \,\big|\, \mathcal{F}_i$$

also, $\mathcal{F}_i \perp\!\!\!\perp \mathcal{F}_c \vee \mathcal{F}_{c'} \,\big|\, \mathcal{F}_t$ since t separates i from d on $\boldsymbol{G}$.

Then by (5f) followed by (5b), the last relations imply that $\mathcal{F}_c \perp\!\!\!\perp \mathcal{F}_{c'} \,\big|\, \mathcal{F}_t$ and we are done.

Next, we show that for all $k \in \{1, \ldots, K-1\}$ (so that $t \notin c_k$), c_k is an element of P'_t. If not, then there exists a $c = c_k \in R$, with $t \notin c$, s.t. c is the disjoint union of some subsets e_1 and e_2 and $\mathcal{F}_{e_1} \perp\!\!\!\perp \mathcal{F}_{e_2} \,\big|\, \mathcal{F}_t$. Also, $\mathcal{F}_{e_1} \vee \mathcal{F}_{e_2} \perp\!\!\!\perp \mathcal{F}_i \,\big|\, \mathcal{F}_t$ so by (5d) we get $\mathcal{F}_{e_1} \perp\!\!\!\perp \mathcal{F}_{e_2} \vee \mathcal{F}_t \,\big|\, \mathcal{F}_i$. We also have $\mathcal{F}_{e_1} \vee \mathcal{F}_{e_2} \perp\!\!\!\perp \mathcal{F}_t \,\big|\, \mathcal{F}_i$ since $e_1 \cup e_2 = c$ and t belongs to another set in the finest valid partition w.r.t. i. From the last two relations and by (5f) followed by (5b) we get $\mathcal{F}_{e_1} \perp\!\!\!\perp \mathcal{F}_{e_2} \,\big|\, \mathcal{F}_i$. But by Lemma 2, $c \in P_i$ cannot be so decomposed, so $\{e_1, e_2\} = \{c, \emptyset\}$ and we have proved the claim.

So we have shown, by induction, that there exists a junction tree $\boldsymbol{G}_d$ on d, where node t has at least $K-1$ neighbors with subtrees corresponding to $c_k,\ 1 \leq k \leq K-1$. Now we modify the original junction tree $\boldsymbol{G}$ in $K+1$ steps to get trees $\boldsymbol{H}, \boldsymbol{H}_0, \ldots, \boldsymbol{H}_{K-1}$ as follows.

First, we form $\boldsymbol{H}$ by replacing the subtree in $\boldsymbol{G}$ on d, with $\boldsymbol{G}_d$ above, connecting i to t with an edge. By Lemma 4, $\boldsymbol{H}$ is a junction tree on $\{1, \ldots, M\}$.

Let $\boldsymbol{H}_0$ be the subtree of $\boldsymbol{H}$ after removing the subtrees around t on $c_k,\ 1 \leq k \leq K-1$. Then $\boldsymbol{H}_0$ is also a junction tree. For each $j = 1, \ldots, K-1$, let $\boldsymbol{L}_j$ be the subtree of $\boldsymbol{H}$ on c_j, and let x_j be the node on c_j that was connected to t in $\boldsymbol{H}$. Then at each step $j = 1, \ldots, K-1$ we form $\boldsymbol{H}_j$ by joining $\boldsymbol{H}_{j-1}$ and $\boldsymbol{L}_j$ by adding the edge between i and x_j (see Fig. 2).

We now show inductively that each $\boldsymbol{H}_j$ is a junction tree. By induction hypothesis $\boldsymbol{H}_{j-1}$ is a junction tree. At the same time,

$\boldsymbol{L}_j$, being a subtree of a junction tree, is also a junction tree. Further

$$\mathcal{F}_{c_j} \perp\!\!\!\perp \bigvee_{r=1}^{j-1} \mathcal{F}_{c_r} \vee \mathcal{F}_{c_K} \vee \mathcal{F}_{d'} \,\big|\, \mathcal{F}_i$$

since c_j is a set in a valid partition w.r.t. i.

Also, $\mathcal{F}_{c_j} \perp\!\!\!\perp \bigvee_{r=1}^{j-1} \mathcal{F}_{c_r} \vee \mathcal{F}_{c_K} \vee \mathcal{F}_{d'} \,\big|\, \mathcal{F}_{x_j}$, since on the junction tree $\boldsymbol{H}$, node x_j separates c_j from $c_K \cup d' \bigcup_{r=1}^{j-1} c_r$. Then, by Lemma 4, each $\boldsymbol{H}_j$ is a junction tree. (Note that $\boldsymbol{H}_{K-1}$ is a junction tree on $\{1, \ldots, M\}$.)

Next, we perform the same transformation on $\boldsymbol{H}_{K-1}$, starting with other neighbors of i. The resulting tree will be a junction tree, and will be compatible with P_i. □

C. Algorithm to Find a Junction Tree

We will now give an algorithm to find a junction tree when one exists.

Given a set of σ-fields $\{\mathcal{F}_1, \ldots, \mathcal{F}_M\}$.

Algorithm 3: Pick any node $i \in \{1, \ldots, M\}$ as the root.

- If $M = 2$ then the single edge $(1, 2)$ is a junction tree. Stop.
- Find the finest valid partition of $\{1, \ldots, M\} \backslash \{i\}$ w.r.t. i, $P_i = \{c_1, \ldots, c_l\}$ (see notes below).
- For j = 1 to l
- Find a node $t \in c_j$ s.t. $\mathcal{F}_i \perp\!\!\!\perp \mathcal{F}_{c_j} \,\big|\, \mathcal{F}_t$. If no such node exists, then stop; no junction tree exists.
- Find a junction tree on c_j with node t as root. Attach this tree, by adding edge (i, t).
- End For

Proof of Correctness of Algorithm 3: At each iteration, t is chosen so $\mathcal{F}_i \perp\!\!\!\perp \mathcal{F}_{c_j} \,\big|\, \mathcal{F}_t$. But we also had $\mathcal{F}_{c_j^c} \perp\!\!\!\perp \mathcal{F}_t \vee \mathcal{F}_{c_j} \,\big|\, \mathcal{F}_i$. By (5e), the last two relations imply $\mathcal{F}_{c_j} \perp\!\!\!\perp \mathcal{F}_i \vee \mathcal{F}_{c_j^c} \,\big|\, \mathcal{F}_t$. But we also have $\mathcal{F}_{c_j} \perp\!\!\!\perp \mathcal{F}_i \vee \mathcal{F}_{c_j^c} \,\big|\, \mathcal{F}_i$. So by Lemma 4, we have a junction tree at each step. Also, from Theorem 5, if the algorithm fails, then there is no junction tree. □

Remark: In the general case of the signed conditional independence, we know of no better way to find the finest valid partition than an exhaustive search in an exponential subset of all the partitions. In the case of unsigned measures, however, we can show that when a junction tree exists, the finest valid partition coincides with the finest pairwise partition, which can be found in polynomial time, see [10]. Therefore, starting with a conventional MPF problem, the question of existence of a *probabilistic* junction tree can be answered in polynomial time, as long as a nonnegative measure is chosen.

V. Construction of Junction Trees – Lifting

In the preceding section, we gave an algorithm to find a junction tree, when one exists. In this section, we deal with the case when Algorithm 3 declares that no junction tree exists for the given set of σ-fields. In particular, we would like to expand the σ-fields in some *minimal* sense, so as to ensure that we can construct a junction tree.

First, we review some methods used in the conventional GDL framework to create junction trees (for details see [3]). Given the GDL variables $\{x_1, \ldots, x_n\}$ and local domains $\{S_1, \ldots, S_M\}$ as defined in Section II, we define the *moral graph* for the problem to be a undirected graph with vertices $1, \ldots, n$, where an edge (i, j) exists iff $\{i, j\} \subseteq S_k$ for some $k = 1, \ldots, M$. With this construction, each S_k is the subset of at least one of the (maximal) cliques of the moral graph and, hence, a junction tree on the cliques of the moral graph can serve as a junction tree for the local domains $\{S_1, \ldots, S_M\}$.

It is further known that a junction tree of the cliques of a graph $\boldsymbol{G}$ exists iff $\boldsymbol{G}$ is *triangulated* or *chordal*, i.e., every cycle of length 4 or more on $\boldsymbol{G}$ has a chord (see, e.g., [3, Sec. 4.3]). Then the general procedure to construct a junction tree is the moralization and triangulation process: form the moral graph and triangulate it; then a junction tree will exist on the cliques of the triangulated graph. There are efficient algorithms to find a triangulation of a given graph, although the general problem of finding the optimal triangulation for moral graph of a GDL problem is NP-hard (see [3]), where optimality is measured in terms of the complexity of the message-passing algorithm on the junction tree created from the cliques of the triangulated graph.

Note that the triangulation process in effect expands the cliques of the moral graph—by adding variables—in a way to create conditional independencies which are required on a junction tree. We will see that our lifting procedure achieves the same goal, with the added possibility of expansion of σ-fields without adding a whole variable direction. In a sense, this amounts to automatically discovering new hidden variables in the space and using them to minimally expand the local domains.

A. Lifting

Definition 5: Let $(\Omega, \{\mathcal{F}_1, \ldots, \mathcal{F}_M\}, \mu)$ be a given measure space. We call a measure space $(\Omega', \{\mathcal{F}'_1, \ldots, \mathcal{F}'_M\}, \mu')$ a *lifting* of $(\Omega, \{\mathcal{F}_1, \ldots, \mathcal{F}_M\}, \mu)$ if there is a map $f : \Omega' \to \Omega$ such that

- μ' is consistent with μ under the map f, i.e.,
$$\forall A \in \mathcal{F}_{\{1,\ldots,M\}}, \qquad \mu(A) = \mu'(f^{-1}(A));$$
- for all $i = 1, \ldots, M$, f is $(\mathcal{F}'_i, \mathcal{F}_i)$-*measurable*, i.e.,
$$\forall\, A \in \mathcal{F}_i, \qquad f^{-1}(A) \in \mathcal{F}'_i$$
where for $A \in \Omega$
$$f^{-1}(A) := \{\omega' \in \Omega' : f(\omega') \in A\}.$$

In words, up to some renaming of the elements, each σ-field $\mathcal{F}_i$ is a sub-σ-field of $\mathcal{F}'_i$, and $\mathcal{F}'_i$ is obtained from $\mathcal{F}_i$ by *splitting* some of the atoms.

We now describe the connection of the above concept with our problem.

Let $(\Omega', \{\mathcal{F}'_1, \ldots, \mathcal{F}'_M\}, \mu')$ be a lifting of $(\Omega, \{\mathcal{F}_1, \ldots, \mathcal{F}_M\}, \mu)$ with the lifting map $f : \Omega' \to \Omega$ as described in Definition 5. Let $\boldsymbol{G}'$ be a junction tree on $\{1, \ldots, M\}$ corresponding to σ-fields $\{\mathcal{F}'_1, \ldots, \mathcal{F}'_M\}$. We will construct a junction tree $\boldsymbol{G}''$ from $\boldsymbol{G}'$ such that running Algorithm 2 on $\boldsymbol{G}''$ will produce the desired conditional expectations at the appropriate nodes.

For each $i = 1, \ldots, M$, let $\mathcal{G}_i$ be the σ-field on Ω' with atoms

$$\mathcal{A}(\mathcal{G}_i) = \{f^{-1}(a) \,:\, a \in \mathcal{A}(\mathcal{F}_i)\}$$

and let $Y_i \in \mathcal{G}_i$ be the random variable with $Y_i(f^{-1}(a)) = X_i(a)$ for all $a \in \mathcal{A}(\mathcal{F}_i)$; so that up to a renaming of the atoms

and elements, $(\Omega, \mathcal{F}_i, \mu)$ and $(\Omega', \mathcal{G}_i, \mu')$ are identical measure spaces and X_i and Y_i are identical random variables. Let $\boldsymbol{G}''$ be a tree with nodes $\{1, \ldots, M, M+1, \ldots, 2M\}$—with corresponding σ-fields $\{\mathcal{F}'_1, \ldots, \mathcal{F}'_M, \mathcal{G}'_1, \ldots, \mathcal{G}'_M\}$ and random variables $\{1, \ldots, 1, Y_1, \ldots, Y_M\}$—which is generated by starting with $\boldsymbol{G}'$ and adding edges $(j, M+j)$ for each $j = 1, \ldots, M$. In words, $\boldsymbol{G}''$ is a graph obtained from $\boldsymbol{G}'$ by adding and attaching each node with σ-fields $\mathcal{G}_i$ for $i = 1, \ldots, M$ (which are in turn equivalent to the original $\mathcal{F}_i$'s) to the node with σ-field $\mathcal{F}'_i$. Then, by Lemma 4, $\boldsymbol{G}''$ is a junction tree and hence running Algorithm 2 on $\boldsymbol{G}''$ will produce $\boldsymbol{E}\big[\prod_{i=1}^{M} Y_i \big| \mathcal{G}_j\big]$ at the node labeled $(M+j)$ for each $j = 1, \ldots, M$. But these are equivalent to $\boldsymbol{E}\big[\prod_{i=1}^{M} X_i \big| \mathcal{F}_j\big]$ for $j = 1, \ldots, M$ and we have thus solved the probabilistic MPF problem.

So we only need to establish how to lift a certain collection of σ-fields to create the required independencies and form a junction tree.

Suppose we have three σ-fields, $\mathcal{F}_1, \mathcal{F}_2$, and $\mathcal{F}_3$, and we would like to find a lifting $(\Omega', \{\mathcal{F}'_1, \mathcal{F}'_2, \mathcal{F}'_3\}, \mu')$ of $(\Omega, \{\mathcal{F}_1, \mathcal{F}_2, \mathcal{F}_3\}, \mu)$ so as to have the conditional independence relation $\mathcal{F}'_1 \perp\!\!\!\perp \mathcal{F}'_3 \,\big|\, \mathcal{F}'_2$. Let $a_i \in \mathcal{A}(\mathcal{F}_1)$, $c_k \in \mathcal{A}(\mathcal{F}_2)$, and $b_j \in \mathcal{A}(\mathcal{F}_3)$ be arbitrary atoms. For each c_k, let A_k be the matrix with (i,j) entry equal to $\mu(a_i, b_j, c_k)$. Let $A_k = A_k^1 + A_k^2$ be an additive decomposition of A_k. Then this decomposition corresponds to a lifting of the measure space obtained by splitting the atom c_k of $\mathcal{F}_2$ into two, say c_k^1 and c_k^2, where $\mu'(a_i, b_j, c_k^1)$ and $\mu'(a_i, b_j, c_k^2)$ are defined as the (i,j) entries of A_k^1 and A_k^2, respectively. We will use this decomposition technique to obtain a lifting that makes $\mathcal{F}'_1 \perp\!\!\!\perp \mathcal{F}'_3 \,\big|\, \mathcal{F}'_2$.

Remember first that in order to have this independence, if $\mu(c_k) = 0$, we must have $\mu(a_i, b_j, c_k) = 0$ for all i, j, i.e., the matrix A_k must be zero. Therefore, if A_k is a nonzero matrix with zero sum of entries, we first decompose it as the sum of two matrices with nonzero sum of entries. This corresponds to splitting atom c_k in a way that the new atoms have nonzero measure.

Next for each such matrix A_k with nonzero sum of entries, the independence condition corresponding to c_k is exactly the condition that A_k is rank one. Then, if A_k is *not* rank one, we can use an "optimal" decomposition of A_k as the sum of say q rank-one matrices (so that none of the matrices are zero sum).[4] This corresponds to splitting the atom c_k into q atoms, $\{c_1^k, \ldots, c_q^k\}$ where each of c_l^k's renders $\mathcal{F}_1$ and $\mathcal{F}_3$ independent; c_l^k's are the new atoms of the lifted σ-fields $\mathcal{F}'_2$.

B. Algorithm to Construct a Junction Tree

Combining the above ideas with the Algorithm 3 we obtain the following.

Algorithm 4: Pick any node $i \in \{1, \ldots, M\}$ as the root.

- If $M = 2$, then the single edge $(1,2)$ is a junction tree. Stop.
- Find any[5] valid partition of $\{1, \ldots, M\} \backslash \{i\}$ w.r.t. i, $P_i = \{c_1, \ldots, c_l\}$.
- For j $= 1$ to l
- Find a node $t \in c_j$ s.t. $\mathcal{F}_i \perp\!\!\!\perp \mathcal{F}_{c_j} \big| \mathcal{F}_t$. If no such node exists, then pick any $t \in c_j$. Lift $\mathcal{F}_t$ by splitting some or all of its atoms as discussed above, so to have $\mathcal{F}_i \perp\!\!\!\perp \mathcal{F}_{c_j} \big| \mathcal{F}_t$.
- Find a junction tree on c_j with node t as root. Attach this tree, by adding edge (i,t).
- End For

The resulting measure space $(\Omega', \{\mathcal{F}'_1, \ldots, \mathcal{F}'_M\}, \mu')$ is a lifting of the original measure space $(\Omega, \{\mathcal{F}_1, \ldots, \mathcal{F}_M\}, \mu)$, and the tree generated by this algorithm is a junction tree corresponding to this lifted collection of σ-fields.

Once a junction tree is available, Algorithm 2 can be applied to solve the probabilistic MPF problem of Section IV. Note that any algorithm obtained in this manner is simply a reformulation of the original marginalization problem, and can be viewed as a GDL-type algorithm after introduction of certain new variables and potentially unintuitive manipulation of the objective functions. The advantage of our measure-theoretic framework is that it allows for *automation* of this process, without the need for discovering "hidden variables."

The complexity of Algorithm 4 will be discussed in Section VI.

C. Measure Theory Versus Variables

In Section IV-B, after presenting the measure-theoretic version of Algorithm 2, we emphasized the connection with the original GDL by rewriting Algorithm 2 in terms of "variables" representing atoms of each σ-field. The purpose of doing this was to show that, while the language of σ-fields might seem exotic, all our algorithms can be discussed in more conventional terms. We emphasize once again that the "variables" appearing in (9) and (10) bear no direct relation to the "variables" of the conventional GDL.

It would probably be useful if one could similarly describe Algorithm 4 in terms of some variables, in order to emphasize once again that our algorithms can be thought of in conventional terms. We now describe how to rephrase Algorithm 4 in the language of "variables." One should note that the main step of Algorithm 4, namely, the splitting of atoms of a σ-field in order to create conditional independencies, in a sense introduces new "variables." We illustrate how to think of Algorithm 4 in terms of variables by first considering a basic triangulation step in the conventional GDL along the lines of the basic step of our algorithm, and then describing how the basic step of our algorithm can be though of in terms of "variables."

Consider state space

$$\Omega := \big\{(x_1, \ldots, x_n) : x_i \in [q_i] := \{1, \ldots, q_i\}\big\}$$

for integers $q_i \geq 2$, with a uniform measure, i.e., $\mu(\boldsymbol{x}) = 1$ for all $\boldsymbol{x} \in \Omega$. Let r, s, and t be subsets of $\{1, \ldots, n\}$, and define $s' := (r \cap t) \backslash s$. Define $\mathcal{F}_1, \mathcal{F}_2$ and $\mathcal{F}_3$ to be the σ-fields

[4]This can always be done with $q = \mathrm{rank}(A_k)$. Obviously $\mathrm{rank}(A_k)$ is also a lower bound for q. An optimal decomposition, however, not only aims to minimize q, but also involves minimizing the number of nonzero entries of the decomposing matrices, as discussed in Section VI.

[5]Although any valid partition will work, in general finer partitions should result in better and less complex algorithms (see Section VI).

with atoms corresponding to (the level sets of) $\boldsymbol{x}_r, \boldsymbol{x}_s$ and $\boldsymbol{x}_t$ respectively, so that for example

$$\mathcal{A}(\mathcal{F}_1) = \left\{ \{\boldsymbol{x} \in \Omega : \boldsymbol{x}_r = \boldsymbol{x}_r^1\} \, , \, \forall \boldsymbol{x}_r^1 \right\}$$

and so on. We will index the atoms of $\mathcal{F}_1$ by $\boldsymbol{x}_r^1 \in [q_r]$ where $q_r := \prod_{i \in r} q_i$, and similarly for $\mathcal{F}_2$ and $\mathcal{F}_3$. Note that for convenience we have overloaded symbol $\boldsymbol{x}_r$, as either an integer in $[q_r]$ or, isomorphically, as an $|r|$-tuple in $\prod_{i \in r}[q_i]$. The distinction will be apparent from the context.

For each atom $\boldsymbol{x}_s^2$ of $\mathcal{F}_2$, let $A_{\boldsymbol{x}_s^2}$ be the $(q_r \times q_t)$ matrix of joint measure, with $(\boldsymbol{x}_r^1, \boldsymbol{x}_t^3)$ entry equal to $\mu(\boldsymbol{x}_r^1, \boldsymbol{x}_t^3, \boldsymbol{x}_s^2)$, where we use $\mu(\boldsymbol{x}_r^1, \boldsymbol{x}_t^3, \boldsymbol{x}_s^2)$ as the shorthand for the measure of the intersection of the $\boldsymbol{x}_r^1$th atom of $\mathcal{F}_1$, the $\boldsymbol{x}_t^3$th atom of $\mathcal{F}_3$, and the $\boldsymbol{x}_s^2$th atom of $\mathcal{F}_2$. But μ is the uniform measure, so $\mu(\boldsymbol{x}_r^1, \boldsymbol{x}_t^3, \boldsymbol{x}_s^2)$ is 1 iff $\boldsymbol{x}_r^1, \boldsymbol{x}_s^2$ and $\boldsymbol{x}_t^3$ are consistent, i.e.,

$$\begin{aligned}\mu(\boldsymbol{x}_r^1, \boldsymbol{x}_t^3, \boldsymbol{x}_s^2) &= 1(\boldsymbol{x}_{r\cap s}^1 = \boldsymbol{x}_{r\cap s}^2) 1(\boldsymbol{x}_{s\cap t}^2 = \boldsymbol{x}_{s\cap t}^3) 1(\boldsymbol{x}_{r\cap t}^1 = \boldsymbol{x}_{r\cap t}^3) \\ &= 1(\boldsymbol{x}_{r\cap s}^1 = \boldsymbol{x}_{r\cap s}^2) 1(\boldsymbol{x}_{s\cap t}^2 = \boldsymbol{x}_{s\cap t}^3) 1(\boldsymbol{x}_{s'}^1 = \boldsymbol{x}_{s'}^3).\end{aligned} \tag{14}$$

But we showed earlier that $\mathcal{F}_1 \perp\!\!\!\perp \mathcal{F}_3 \,\big|\, \mathcal{F}_2$ iff $A_{\boldsymbol{x}_s^2}$ is rank one for all $\boldsymbol{x}_s^2$, i.e., (14) factorizes as $f_1(\boldsymbol{x}_r^1) \cdot f_3(\boldsymbol{x}_t^3)$. From (14), it is obvious that this happens iff $s' = (r \cap t) \backslash s$ is empty, i.e., $r \cap t \subseteq s$. Remember from Section II that this is precisely the condition for r — s — t to be a (GDL) junction tree.

Now suppose that s' is not empty. Then, after possibly reordering its rows and/or columns, $A_{\boldsymbol{x}_s^2}$ will be a diagonal block matrix, where each block is a $(q_{r\backslash s'} \times q_{t\backslash s'})$ matrix, and where the diagonal blocks (D) are blocks of all 1's, and off-diagonal blocks (O) are 0 matrices

$$A_{\boldsymbol{x}_s^2} = \begin{bmatrix} D & O & \cdots & O \\ O & D & & O \\ \vdots & & \ddots & \vdots \\ O & O & \cdots & D \end{bmatrix}. \tag{15}$$

Then the rank-one decomposition of $A_{\boldsymbol{x}_s^2}$ corresponds to the decomposition where the diagonal blocks are separated

$$A_{\boldsymbol{x}_s^2} \begin{bmatrix} D & O & \cdots & O \\ O & O & & O \\ \vdots & & \ddots & \vdots \\ O & O & \cdots & O \end{bmatrix} + \begin{bmatrix} O & O & \cdots & O \\ O & D & & O \\ \vdots & & \ddots & \vdots \\ O & O & \cdots & O \end{bmatrix} + \cdots + \begin{bmatrix} O & O & \cdots & O \\ O & O & & O \\ \vdots & & \ddots & \vdots \\ O & O & \cdots & D \end{bmatrix}. \tag{16}$$

This corresponds to the following decomposition of (14):

$$A_{\boldsymbol{x}_s^2}(\boldsymbol{x}_r^1, \boldsymbol{x}_t^3) = \sum_{x_{s'}} 1(\boldsymbol{x}_{r\cap s}^1 = \boldsymbol{x}_{r\cap s}^2) 1(\boldsymbol{x}_{s\cap t}^2 = \boldsymbol{x}_{x\cap t}^3) 1(\boldsymbol{x}_{s'}^1 = \boldsymbol{x}_{s'}^3 = \boldsymbol{x}_{s'}).$$

The corresponding lifting is obtained by splitting atom $\boldsymbol{x}_s^2$ of $\mathcal{F}_2$ by intersecting it with the level sets of $\boldsymbol{x}_{s'}$, so the new atoms can be represented by $(\boldsymbol{x}_s^2, \boldsymbol{x}_{s'})$. This means that the atoms of $\mathcal{F}_2'$ are the level sets of $\boldsymbol{x}_{s\cup s'} = \boldsymbol{x}_{s\cup(r\cap t)}$. This is precisely what would be done in the GDL framework: in order for the local domains $\boldsymbol{x}_r, \boldsymbol{x}_s$ and $\boldsymbol{x}_t$ to form a junction chain r — s — t, we expand the domain s to contain $r \cap t$. We have, therefore, shown that when the state space and the σ-fields are represented in terms of orthogonal directions of variables, our condition for independence reduces to that of GDL, and our lifting algorithm will produce the same expansions as in the GDL framework.

Next consider the case when the sample space Ω and the underlying measure μ are arbitrary. For each $i = 1, 2, 3$ let $q_i := |\mathcal{A}(\mathcal{F}_i)|$, and define variables x, y, z taking values in $[q_1], [q_2]$, and $[q_3]$, respectively, corresponding to different atoms of $\mathcal{F}_1, \mathcal{F}_2$, and $\mathcal{F}_3$; this means that

$$\begin{aligned}\mathcal{A}(\mathcal{F}_1) &= \big\{\{x = 1\}, \ldots, \{x = q_1\}\big\} \\ \mathcal{A}(\mathcal{F}_2) &= \big\{\{y = 1\}, \ldots, \{y = q_2\}\big\} \quad \text{and} \\ \mathcal{A}(\mathcal{F}_3) &= \big\{\{z = 1\}, \ldots, \{z = q_3\}\big\}.\end{aligned}$$

Once again, we use the shorthand $\mu(x^1, y^2, z^3)$ for the measure of the intersection of the x^1th atom of $\mathcal{F}_1$, and the y^2th atom of $\mathcal{F}_2$, and the z^3th atom of $\mathcal{F}_3$.

As before, for each $y \in [q_2]$ denote by A_y the matrix of the joint measures $\mu(x, z, y)$. Let r_y denote the rank of A_y. Correspondingly, we decompose each A_y as the sum of r_y rank-one matrices, which is equivalent to splitting the yth atom of $\mathcal{F}_2$ into r_y new atoms. To properly index these new atoms, we need to invent a new "hidden variable" w_y taking value in the set $[r_y]$. Then the new atoms of the lifted σ-field, $\mathcal{F}_2'$ can be indexed by pairs of variables $(y, w_y) \in [q_2] \times [r_y]$. In particular, $\mathcal{F}_2'$ will have $\sum_{y=1}^{q_2} r_y$ atoms, compared to the q_2 atoms of $\mathcal{F}_2$. Note, however, that in general, the "directions" corresponding to w_y's are not aligned to those of variables x, y or z, and the pair (y, w_y) does not take value in a product space. Considering some special cases, however, may add some intuition into the process of lifting.

If A_y is full rank for some y, then w_y is aligned to either x or z: Remember that A_y is a $(q_1 \times q_3)$ matrix. Suppose $r_y = q_1 \leq q_3$, so that A_y has full row rank. We then decompose A_y row-wise, into q_1 single-row matrices. But rows of A_y correspond to the atoms of $\mathcal{F}_1$, which are indexed precisely by the variable x. Therefore, for this particular value of y, the variable w_y is identical to the variable x, and the new atoms are indexed by pairs (y, x) where $x \in [q_1]$. This indeed *is* a product space. If A_y is full rank for all y's, then creating the desired conditional independence requires a full augmentation of $\mathcal{F}_2$ and $\mathcal{F}_1$, i.e., $\mathcal{F}_2' = \mathcal{F}_1 \vee \mathcal{F}_2$, and atoms of $\mathcal{F}_2'$ correspond to the elements of the product space $[q_1] \times [q_2]$, indexed by pair (x, y).

More generally, if A_y can be rearranged as a block matrix with rank-one blocks such that each row and column of blocks contains at most one nonzero block—such as in (15)—then the "hidden variables" w_y's will correspond to "subdirections" of variables x and z, as determined by the position of the nonzero blocks.

The other trivial case is when A_y is rank one for some y. In this case, no splitting of the yth atom of $\mathcal{F}_2$ is needed, and the corresponding atom of $\mathcal{F}_2$ will be duplicated as an atom of $\mathcal{F}_2'$.

We will close this section by showing that any junction tree obtained using the moralization and triangulation procedure can also be found using Algorithm 4.

Suppose we start with the conventional GDL problem discussed in Section II, with $N = M$ local domains $\{S_1, \ldots, S_M\}$. Let $\boldsymbol{G}$ be a junction tree of cliques of the triangulated moral graph for the given MPF problem. For each node i of $\boldsymbol{G}$, let S_i' be the domain (subset of $\{1, \ldots, n\}$) for the corresponding

clique. Then, as discussed before, each S_j of the original GDL local domains is contained in the domain S'_i for a node of $\boldsymbol{G}$. We can assume then that $\boldsymbol{G}$ is a tree on $\{1,\ldots,M\}$ by identifying each node of $\boldsymbol{G}$ with the index i of a local domain it contains (if an index $i \in \{1,\ldots,M\}$ is left unaccounted for, we will add a node i as a neighbor of a node in $\boldsymbol{G}$ which also contained local domain S_i, and we set $S'_i = S_i$.) Therefore, we will have $S_i \subseteq S'_i$ for each node i of $\boldsymbol{G}$. Then, as before, we let the state space Ω to be represented by the GDL variables $\{x_1,\ldots,x_n\}$, equipped with the uniform measure, and identify $\{\mathcal{F}_1,\ldots,\mathcal{F}_M\}$ as the σ-fields whose atoms are the level sets of $\{x_{S_1},\ldots,x_{S_M}\}$, respectively.

We will then run Algorithm 4, using the convention that each time a lifting is required to create a conditional independence relation $\mathcal{F}_i \perp\!\!\!\perp \mathcal{F}_c \,|\, \mathcal{F}_t$ (where $c \subset \{1,\ldots,M\}$), we lift $\mathcal{F}_t$ to $\mathcal{F}'_t$, where the atoms of $\mathcal{F}'_t$ are the level sets of variables $x_{S'_t}$. Note then that since the atoms of $\mathcal{F}_t$ correspond to x_{S_t} and $S_t \subseteq S'_t$, we have $\mathcal{F}_t \subseteq \mathcal{F}'_t$, confirming that atoms of $\mathcal{F}_t$ are indeed split to obtain those of $\mathcal{F}'_t$. We will now show that with the above choice for the liftings, Algorithm 4 can create $\boldsymbol{G}$ as the junction tree.

We start Algorithm 4 with a node i as the root, where i is a leaf node of $\boldsymbol{G}$. Let t be the neighbor of i in $\boldsymbol{G}$ and let $P := \{c_1,\ldots,c_l\}$ be the partition of $\{1,\ldots,M\}\backslash\{t\}$ that is compatible with $\boldsymbol{G}$. We can assume WLOG that $c_l = \{i\}$. Also, let $j_k \in c_k$ be the neighbor of t in $\boldsymbol{G}$ lying in c_k, for each $k = 1,\ldots,l$. Then the first lifting will be a splitting of atoms of $\mathcal{F}_t$ to ensure $\mathcal{F}_i \perp\!\!\!\perp \bigvee_{k=1}^{l-1} \mathcal{F}_{c_k} \,|\, \mathcal{F}_t$. We do this, according to the above convention, by splitting atoms of $\mathcal{F}_t$ to form $\mathcal{F}'_t$, whose atoms correspond to the level sets of $x_{S'_t}$. But since $\boldsymbol{G}$ is a (GDL) junction tree on domains $\{S'_1,\ldots,S'_M\}$, we must have

$$S'_{c_j} \bigcap (\cup_{k\neq j} S'_{c_k}) \subseteq S'_t, \qquad \text{for all } j = 1,\ldots,l.$$

Then, from earlier discussions in this section

$$\mathcal{F}_{c_1} \perp\!\!\!\perp \mathcal{F}_{c_2} \perp\!\!\!\perp \cdots \perp\!\!\!\perp \mathcal{F}_{c_l} | \mathcal{F}'_t$$

holds, and in particular P is now a valid partition of $\{1,\ldots,M\}\backslash\{t\}$ w.r.t. t (where $\mathcal{F}_t$ has been replaced by its lifting $\mathcal{F}'_t$). This valid partition can be used in the next call to the algorithm to ensure that the neighboring subtrees of t in the junction tree will be identical to those in $\boldsymbol{G}$. It is then easy to see that following similar choices in each step of Algorithm 4, we will end up with $\boldsymbol{G}$ as the junction tree, while $(\Omega, \{\mathcal{F}_1,\ldots,\mathcal{F}_M\}, \mu)$ is lifted to $(\Omega', \{\mathcal{F}'_1,\ldots,\mathcal{F}'_M\}, \mu')$ where $\Omega' = \Omega$, μ' is the uniform measure on Ω', and $\mathcal{F}'_i$ is the σ-field whose atoms are the level sets of variables $x_{S'_i}$. Therefore, we have shown that using Algorithm 4 , one can always retrieve a junction tree equivalent to any (GDL) junction tree obtained using the moralization and triangulation procedure.

VI. Complexity Issues

In this section, we discuss the computational complexity of the methods proposed in this paper. We start by estimating the complexity of computing the conditional expectation of a random variable given a σ-field, with respect to a signed measure.

A. Computing Conditional Expectations

Consider the problem of efficiently computing the conditional expectation $\boldsymbol{E}\big[X\big|\mathcal{G}\big]$. Let $\mathcal{A}'(\mathcal{G}) = \{\beta_1,\ldots,\beta_m\}$ be the atoms of $\mathcal{G}$ with nonzero measure, and let $\mathcal{F}$ denote the sigma field generated by X, with atoms $\mathcal{A}(\mathcal{F}) = \{\alpha_1,\ldots,\alpha_l\}$. Let W denote the $l \times m$ matrix of conditional measures, with (i,j) entry $\mu(\alpha_j|\beta_i)$. Also, define $\boldsymbol{x}$ as an $m \times 1$ column vector with entries $\boldsymbol{x}_j = X(\alpha_j)$. Then, calculating $\boldsymbol{E}\big[X\big|\mathcal{G}\big]$ is precisely equivalent to computing $\boldsymbol{y} := W \cdot \boldsymbol{x}$.

At first look it appears that calculating $W \cdot \boldsymbol{x}$ requires $l \cdot m$ multiplications and $l\cdot(m-1)$ additions. However, using the fact that matrix W is known *a priori*, in many cases we can perform the calculations with a lower number of operations, as suggested in Example 1. To illustrate this in the context of Algorithm 2, note that $\boldsymbol{E}\big[X\big|\mathcal{G}\big]$ is precisely the message sent from $\mathcal{F}$ to $\mathcal{G}$, in a junction "tree" in the form $\mathcal{F}$— $\mathcal{G}$. Now let $\mathcal{H} := \{\emptyset, \Omega\}$ be the trivial σ-field on the same state space Ω. We will lift the space so as to create the conditional independence $\mathcal{F}' \perp\!\!\!\perp \mathcal{G}' \,|\, \mathcal{H}'$. We will then have a junction chain $\mathcal{F}'$— $\mathcal{H}'$— $\mathcal{G}'$. Then it is easy to see that $\boldsymbol{E}\big[X\big|\mathcal{G}\big]$ will be the same as the message from $\mathcal{H}'$ to $\mathcal{G}'$ on this junction tree (after message from $\mathcal{F}'$ has been received at $\mathcal{H}'$).

Now to create the desired conditional independence, we need the rank-one decomposition of W (note that W—the matrix of conditional measures—is rank one iff A_Ω—the matrix of joint measures $\mu(\alpha_j,\beta_i,\Omega) = \mu(\alpha_j,\beta_i)$ is rank one).

Suppose now that W has rank $r \leq \min(l,m)$. Denote the ith row of W by W_i, and define $z(i)$ to be the number of nonzero elements of W_i. Let $\{W_{i_1},\ldots,W_{i_r}\}$ be a linearly independent subset of rows of W, chosen so as to minimize the sum $z := \sum_{k=1}^{r} z(i_k)$. Then for each $i = 1,\ldots,l$, $W_i = \sum_{k=1}^{r} C_{i,k} W_{i_k}$ for some $l\times r$ matrix C. Incidentally, this matrix has the property that after possibly reordering its rows and/or columns, it has the $r \times r$ identity matrix as a submatrix. Denote by z' the number of nonzero elements of the $(l-r)\times r$ submatrix of C when this $r \times r$ identity submatrix has been removed.

We then have the following rank-one decomposition:

$$W = \sum_{k=1}^{r} C^k \cdot W_{i_k}$$

where C^k is the kth column of C. Corresponding to this decomposition, $W \cdot \boldsymbol{x}$ can be computed as $\boldsymbol{y} = \sum_{k=1}^{r} C^k \cdot W_{i_k} \cdot \boldsymbol{x}$. Calculating $W_{i_k} \cdot \boldsymbol{x}$ requires $z(i_k)$ multiplications and $(z(i_k)-1)$ additions. Note that, as discussed before, while the vector $\boldsymbol{x}$ is arbitrary, the matrix W is known *a priori* and hence we can exclude multiplications and additions by 0's. To compute $\boldsymbol{y} = \sum_{k=1}^{r} C^k \cdot (W_{i_k} \cdot \boldsymbol{x})$, given scalars $(W_{i_k} \cdot \boldsymbol{x})$, we need an additional z' multiplications and $(z' - l + r)$ additions, where z' was defined above for matrix C.

Therefore, to compute $W \cdot \boldsymbol{x}$ we need a total of $(z' + z)$ multiplications and $(z' - l + z)$ additions. Define $\chi := 2(z'+z) - l$ as the *edge complexity* associated with the hypothetical directed edge from $\mathcal{F}$ to $\mathcal{G}$, which is the number of additions and multiplications required for computing $\boldsymbol{E}\big[X\big|\mathcal{G}\big]$. Note from definitions that $r \leq z \leq r\,m$ and $l - r \leq z' \leq (l-r)m$, and, therefore, $l \leq z' + z \leq l \cdot m$. Therefore, using this technique,

as compared to the straightforward multiplications and summations suggested by (3), depending on the structure of the matrix W, there is potential for a saving by a factor of m in the total operations required to carry out the computations.

Also note that one can always use a row-wise rank-one decomposition (i.e., $W = \sum_{k=1}^{l} e^k W_k$, where e^k is an $(l \times 1)$ column vector with a 1 at the kth position and 0's everywhere else). Then an upper bound on χ is $(2\mathrm{nz} - l)$, where variable nz is defined to be the total number of nonzero elements in matrix W.

Next we will discuss the complexity of Algorithm 2.

B. Complexity of the Junction Tree Algorithm 2

First notice that, as discussed in Section V-C, any junction tree obtained using the moralization and triangulation (possibly after introduction of hidden variables), corresponds to a junction tree on a lifting of the original σ-fields—where liftings are done in the whole-variable directions—which can also be discovered by Algorithm 4. Algorithm 2 on this junction tree will then be equivalent to GDL on the junction tree of cliques of the triangulated graph. In this sense then, using our framework we can always find a marginalization algorithm which is at least as good as GDL.

With that point noted, in this subsection we will discuss the complexity of Algorithm 2 in terms of the sizes of the σ-fields on the junction tree, rather than those of the original σ-fields before possible lifting. It is clear that the lifted σ-fields can be substantially larger than the original ones. This is also the case in the original GDL framework, where triangulation can produce enormous cliques.

Let $\boldsymbol{G}$ be a junction tree with σ-fields $\{\mathcal{F}_1, \ldots, \mathcal{F}_M\}$, as defined above, and let $\{X_1, \ldots, X_M\}$ be arbitrary random variables with $X_i \in \mathcal{F}_i$. Denote by q_i the number of atoms of the σ-field $\mathcal{F}_i$, so $q_i = |\mathcal{A}(\mathcal{F}_i)|$.

It can be seen that, in general, the sample space Ω can have as many as $\prod_{i=1}^{M} q_i$ elements and thus full representation of σ-fields and the measure function requires exponentially large storage resources. Fortunately, however, a full representation is not required. Along each edge (i,j) on the tree, Algorithm 2 only requires local computation of $\boldsymbol{E}\big[X\big|\mathcal{F}_i\big]$ for a random variable $X \in \mathcal{F}_j$. This only requires a $q_i \times q_j$ table of the joint measures of the atoms of $\mathcal{F}_i$ and $\mathcal{F}_j$. For an arbitrary edge (i,j), let $\mathcal{A}(\mathcal{F}_i) = \{a_1, \ldots, a_{q_i}\}$ and $\mathcal{A}(\mathcal{F}_j) = \{b_1, \ldots, b_{q_j}\}$ be the sets of atoms of $\mathcal{F}_i$ and $\mathcal{F}_j$. Define $W(i,j)$ to be the $q_i \times q_j$ matrix with (r,s) entry equal to $\mu(a_s \mid b_r)$; note that from Lemma 6 (possibly after trivial simplification of the problem by eliminating the events with measure zero), no atom of $\mathcal{F}_j$ can have measure 0, so $\mu(a_s \mid b_r)$ is defined for all atoms of $\mathcal{F}_j$. Then once a junction tree has been found, we need only keep $2(M-1)$ such matrices (corresponding to the $(M-1)$ edges of the tree) to fully represent the algorithm, for a total of $2\sum_{(i,j)>\text{an edge}} q_i q_j$ storage units.

As defined in the previous section, for each directed edge (i,j) in $\boldsymbol{G}$ let $\chi(i,j)$ be the corresponding edge complexity, i.e., the arithmetic complexity of computing $\boldsymbol{E}\big[X_i\big|\mathcal{F}_j\big]$. From Algorithm 2, calculation of the conditional expectation of the product given a single σ-field $\mathcal{F}_i$ with the most efficient schedule requires updating of the messages from the leaves toward the node i. Each edge is activated in one direction, and at each nonleaf node l the messages need to be multiplied to update the message from l to its neighbor in the direction of i. This requires, for each edge (k,l), an additional q_l multiplications. Thus, the grand total arithmetic operations needed to calculate $\boldsymbol{E}\big[\prod_j X_j\big|\mathcal{F}_i\big]$ is $\sum_{(k,l)} \big(\chi(k,l) + q_l\big)$, where the summation is taken over all the directed edges (k,l) approaching i.

The complexity of the full algorithm, in which $\boldsymbol{E}\big[\prod_j X_j\big|\mathcal{F}_i\big]$ is calculated for all $i = 1, \ldots, M$, can also be found using similar ideas. For each node k, let $d(k)$ denote the number of the neighbors of k on the tree. Then, for each directed edge (k,l), first the $d(k) - 1$ messages from other neighbors of k must be multiplied by X_k and then the conditional expectation given $\mathcal{F}_l$ must be taken. For each nonleaf node k, calculating the product of X_k and all the products of $d(k) - 1$ out of $d(k)$ messages incoming to k requires $(3d(k) - 4)q_k$ multiplications, using a method similar to the one described in [5, Sec. 5]. So the total number of operations required for the full algorithm is

$$\sum_{(k,l)\text{ a dir. edge}} \chi(k,l) + \sum_{k\text{ non-leaf}} (3d(k) - 4)q_k.$$

In particular, if $\boldsymbol{G}$ is a chain $\mathcal{F}_1 - \mathcal{F}_2 - \cdots - \mathcal{F}_M$, the complexity of Algorithm 2 becomes

$$\sum_{i=1}^{M-1} \big(\chi(i,i+1) + \chi(i+1,i)\big) + 2\sum_{i=2}^{M-1} q_i.$$

Using the upper bound $\chi(i,i+1) \le (2\mathrm{nz}(i,i+1) - q_{i+1})$ derived in the previous section, the above complexity of Algorithm 2 on a chain is upper-bounded by $4\sum_{i=1}^{M-1} \mathrm{nz}(i,i+1)$. Here, we have used the fact that $\mathrm{nz}(i,i+1) = \mathrm{nz}(i+1,i)$, i.e., the number of nonzero elements of matrices $W(i,i+1)$ and $W(i+1,i)$ are equal.

C. Complexity of Lifting

In this subsection, we discuss the complexity of Algorithm 4. Recall that the lifting process at each step of Algorithm 4 can increase not only the number of atoms of the σ-field which is being processed, but also the overall number of states in the state space. As mentioned before, in general it is not possible to *a priori* get a reasonable estimate of these figures, since the rank-one decompositions can be arbitrarily irregular and unpredictable. Rather, the complexity will depend on the choices made in each lifting. Consider again the setup used above, with σ-fields $\mathcal{F}_1, \mathcal{F}_2$, and $\mathcal{F}_3$, where lifting is done to create $\mathcal{F}_1' \perp\!\!\!\perp \mathcal{F}_3' \,\big|\, \mathcal{F}_2'$. Also for each $c_k \in \mathcal{A}(\mathcal{F}_2)$, let A_k be the matrix of joint measures $\mu(a_i, b_j, c_k)$ where $\mathcal{A}(\mathcal{F}_1) = \{a_i\}$ and $\mathcal{A}(\mathcal{F}_3) = \{b_j\}$. Then clearly, as noted before, the number of atoms of the lifted σ-field, $\mathcal{F}_2'$ is minimized by using a decomposition of A_k as the sum of r_k rank-one matrices, where $r_k = \mathrm{rank}(A_k)$. However, this is not the only factor determining the complexity of the algorithm. Each time a nonzero entry of A_k is decomposed as sum of nonzero numbers, in effect new states are added to the state space. It is, therefore, preferable to choose decompositions that minimize splitting of nonzero entries. For example, the splitting in (16) does not create any new states, since the nonzero blocks are nonoverlapping. As another example, suppose $A_k = \left(\begin{smallmatrix} 2 & 3 \\ 3 & 4 \end{smallmatrix}\right)$. Then decomposition $A_k = \left(\begin{smallmatrix} 2 & 3 \\ 0 & 0 \end{smallmatrix}\right) + \left(\begin{smallmatrix} 0 & 0 \\ 3 & 4 \end{smallmatrix}\right)$ is preferable to

$A_k = \left(\begin{smallmatrix}1&2\\1&2\end{smallmatrix}\right) + \left(\begin{smallmatrix}1&1\\2&2\end{smallmatrix}\right)$, since the latter results in creating four new states in the state space.

An interesting observation is that in the GDL framework, once a representation of the space is decided by the choice of variables, the operations to create independencies (e.g., moralization and triangulation) will never expand the state space; all expansions of local domains are achieved by addition of whole variables, equivalent to splitting of nonoverlapping rank-one blocks in our framework.

The complexity of each lifting is related (polynomially) to the number of states in the lifted state space; in the above setup, this number is $|\mathcal{A}(\mathcal{F}_1' \vee \mathcal{F}_2' \vee \mathcal{F}_3')|$. Algorithm 4 consists of liftings at different stages, so the overall complexity of Algorithm 4 is on the order of that of the most complex lifting in the process. Now consider the case when the algorithm is run on the σ-fields $\mathcal{F}_1, \ldots, \mathcal{F}_M$, each with q atoms, and when the σ-fields are processed in the order of their index to form a junction chain. It is clear that the original state space can have as many as q^M states, indicating that the general algorithm is exponentially complex in M. Once the lifting has been done to create the independence condition $\mathcal{F}_1' \perp\!\!\!\perp \mathcal{F}_{\{3,\ldots,M\}}' \mid \mathcal{F}_2'$, the number of atoms of $\mathcal{F}_2'$ can grow to q^2. The next lifting, however, will only involve σ-fields $\mathcal{F}_2'$ through $\mathcal{F}_M'$. Thus, the number of relevant states cannot exceed $|\mathcal{A}(\mathcal{F}_{\{2,\ldots,M\}}')|$. But

$$|\mathcal{A}(\mathcal{F}_{\{2,\ldots,M\}}')| \leq |\mathcal{A}(\mathcal{F}_2')| \cdot |\mathcal{A}(\mathcal{F}_{\{3,\ldots,M\}}')| \leq q^2 \cdot q^{(M-2)} = q^M.$$

In other words, the size of the relevant state space will not exceed q^M, and hence, the complexity of none of the consequent liftings will exceed that of the first lifting. An upper bound on the complexity of Algorithm 4 is, therefore, $k\,M\,q^{\frac{3}{2}M}$ for a constant k, where $k\ q^{\frac{3}{2}M}$ is an upper bound on the complexity of rank-one decomposition of a matrix with q^M elements.

However, even though the general form of Algorithm 4 is complex, there are justifications for why it can be a useful practical tool. First, note that as mentioned in Section VI-B, a full description of the marginalization algorithm requires storing only the $W(i,j)$ matrices of the conditional measures of the atoms of the neighboring σ-fields, and a full representation of the exponentially large state space is unnecessary. Second, remember that Algorithm 4 can be executed offline once and for all. Once a junction tree has been created, the corresponding marginalization algorithm can be used for all possible inputs, namely, the random variables $X_1, \ldots, X_M$. Therefore, the cost of creating an efficient algorithm is amortized over many future uses.

As mentioned before, our framework is general enough to be able to take advantage of partial independencies in the objective functions; together with ideas of lifting, we have an automatic algorithm for exploiting these structures.

VII. Examples

In this section, we consider several examples where GDL-based algorithms are applicable. We have created a MATLAB library containing the necessary functions to set up a general marginalization problem and create a junction tree. The examples in this section are processed using that code. To explain the algorithms resulting from our code, we have described each example in detail. Note, however, that to generate the marginalization algorithms, no calculations were done manually and the entire process was automatic. The details are presented here simply to clarify the end result. In each example, we further compare the complexity of our message-passing algorithm with the naive implementation of GDL.

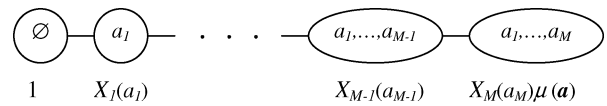

Fig. 3. GDL junction tree for Example 2.

We note that there do exist other methods and techniques that deal with each specific class of problems. Such methods are tailored for a single class of problems and are not applicable to the general case. For instance, [11] gives a procedure that is somewhat more general than the conventional GDL for generation low-complexity algorithms, when the local functions are of specific form called "causally independent." Our measure-theoretic framework, on the other hand, is general in handling all marginalization problems. It is also worth noting that in one of the examples considered—that of decoding linear block codes—our procedure is able to discover the minimal complexity trellis-based decoding algorithm.

Our Example 1 at the beginning of this paper can be generalized as follows.

Example 2: Let A be a finite set of size n, and for each $i = 1, \ldots, M$, let X_i be a real function on A. Let μ be a real (weight) function on A^M. We would like to compute the weighted average of the product of X_i's, namely

$$E := \sum_{a_1 \in A} \cdots \sum_{a_M \in A} \prod_{i=1}^{M} X_i(a_i)\mu(a_1, \ldots, a_M).$$

Suppose that the weight function is in the following form:

$$\mu(a_1, \ldots, a_M) = \prod_{i=1}^{M} f_i(a_i) + \prod_{i=1}^{M} g_i(a_i).$$

As long as the weight function μ is not in product form, the naive version of the GDL algorithm will prescribe

$$E = \sum_{a_1 \in A} X_1(a_1) \cdots \sum_{a_M \in A} X_M(a_M)\mu(a_1, \ldots, a_M)$$

requiring $O(n^M)$ additions and multiplications, corresponding to the junction tree in Fig. 3.

Now, let Ω be the product space A^M with signed measure μ, and for $i = 1, \ldots, M$, let $\mathcal{F}_i$ be the σ-field containing the planes $a_i = c$, so $X_i \in \mathcal{F}_i$. Let $\mathcal{F}_0 = \{\emptyset, \Omega\}$ be the trivial σ-field. Then the problem is to find the conditional expectation of the product of the X_i's given $\mathcal{F}_0$.

The best junction tree is obtained by lifting the space so that given $\mathcal{F}_0'$ all other σ–fields are mutually conditionally independent. To do this, we split the atom Ω of $\mathcal{F}_0$ into two atoms Ω_1 and Ω_2. In effect, for each element $(a_1, \ldots, a_M)$ of Ω, the new space Ω' has two elements $(a_1, \ldots, a_M) \cap \Omega_1$ and $(a_1, \ldots, a_M) \cap \Omega_2$. The new weight function is defined on Ω' as

$$\mu'((a_1, \ldots, a_M^1) \cap \Omega_1) = \prod_{i=1}^{M} f_i(a_i)$$

and

$$\mu'((a_1,\ldots,a_M)\cap\Omega_1)=\prod_{i=1}^{M} g_i(a_i).$$

Then there is a star-shaped junction tree on $\{0,\ldots,M\}$ with node 0 at the center. The message-passing algorithm on this tree is

$$\begin{aligned}E &= \boldsymbol{E}\big[\prod X_i\big|\mathcal{F}_0\big](\Omega)\\ &= \boldsymbol{E}\big[\prod X_i\big|\mathcal{F}_0'\big](\Omega_1)+\boldsymbol{E}\big[\prod X_i\big|\mathcal{F}_0'\big](\Omega_2)\\ &= \prod_{i=1}^{M}\boldsymbol{E}\big[X_i\big|\mathcal{F}_0'\big](\Omega_1)+\prod_{i=1}^{M}\boldsymbol{E}\big[X_i\big|\mathcal{F}_0'\big](\Omega_2)\\ &= \prod_{i=1}^{M}\sum_{a_i\in A}X_i(a_i)f_i(a_i)+\prod_{i=1}^{M}\sum_{a_i\in A}X_i(a_i)g_i(a_i).\end{aligned}$$

Note that this requires only $O(Mn)$ additions and multiplications. □

Example 3: Partition Function in a Circuit-Switched Network: In a circuit-switched network, one is interested in finding the invariant distribution of calls in progress along routes of the network. It can be shown (see [12]) that the invariant distribution has the form

$$\pi(a_1,\ldots,a_M)=\frac{X_1(a_1)\cdots X_M(a_M)}{Z}\prod_{j=1}^{L}1\left(\sum_{i\in R_j}a_i<n_j\right).$$

Here, a_i is the number of calls along route i; M is the total number of routes; $X_i(a_i)$ is a known function (invariant distribution of a_i if the links had an infinite number of circuits); L is the number of links in the network; n_j is the capacity of link j; and $R_j\subset\{1,\ldots,M\}$ is the index set of routes that use link j. Finally, Z is a normalizing factor called the partition function, and is defined by

$$Z:=\sum_{a_1,\ldots,a_M}\prod_{i=1}^{M}X_i(a_i)\prod_{j=1}^{L}1\left(\sum_{i\in R_j}a_i<n_j\right).$$

Therefore, in order to calculate the invariant distribution, one only needs to calculate the partition function Z. Having this in mind, we consider the following simplest case.

Let $X_1(a_1),\ldots,X_M(a_M)$ be arbitrary functions with integer variables $a_1,\ldots,a_M$. Let $f(s)=1(0\le s<n)\cdot g(s)$ be an arbitrary function of the integer variable s, which vanishes when $s\notin\{0,\ldots,n-1\}$. We would like to calculate the following weighted marginalization:

$$Z=\sum_{a_1}\cdots\sum_{a_M}f(a_1+\cdots+a_M)\prod_{i=1}^{M}X_i(a_i).$$

Relying on orthogonal directions of independent variables to represent the state space, a GDL algorithm will suggest

$$Z=\sum_{a_1}X_1(a_1)\sum_{a_2}X_2(a_2)\cdots\sum_{a_M}f(a_1+\cdots+a_M)X_M(a_M).$$

Define $s_j=\sum_{i=1}^{j}a_i$. Then, noting that $s_1\le s_2\le\cdots\le s_M$ and that $f(s_M)=0$ for $s_M\ge n$, one can interpret the above sum as the follows:

$$Z=\sum_{a_1=0}^{n-1}X_1(a_1)\sum_{a_2=0}^{n-1-s_1}X_2(a_2)\cdots\sum_{a_M=0}^{n-1-s_{M-1}}f(a_1+\cdots+a_M)X_M(a_M).$$

Note, however, that even this simplified version requires $O(n^M)$ arithmetic operations.

We will now show that our Algorithm 4 creates a simple junction tree, and implementation of Algorithm 2 on that junction tree substantially simplifies the marginalization problem.

Define a sample space $\Omega=\{(a_1,\ldots,a_M):0\le a_i,\ 0\le s_M<n\}$ with uniform measure $\mu(\omega)=1$ for all $\omega\in\Omega$; here, for ease of notation, we denote elements of Ω by ω, and define $a_i(\omega)$ to be the ith coordinate of ω. We also define, as before, $s_j(\omega)=\sum_{i=1}^{j}a_i(\omega)$. When there is no risk of confusion, we drop the explicit dependence of a_i's and s_j's on ω.

For each $i=1,\ldots,M$, let $\mathcal{F}_i$ be the σ-field with atoms

$$\mathcal{A}(\mathcal{F}_i)=\big\{\{\omega\in\Omega:a_i=j\}:j=0,\ldots,n-1\big\}.$$

Define also σ-field $\mathcal{G}$ with atoms

$$\mathcal{A}(\mathcal{G})=\big\{\{\omega\in\Omega:s_M=j\}:j=0,\ldots,n-1\big\}.$$

Next we view local functions as random variables measurable in the corresponding σ-field: $X_i\in\mathcal{F}_i$ and $g\in\mathcal{G}$. The problem is then to find $\boldsymbol{E}\big[g\prod_{i=1}^{M}X_i\big]$.

We now follow Algorithm 4 step by step to create a junction tree. We pick $\mathcal{G}$ as the root node and $\mathcal{F}_M$ as the neighboring node, and lift to create conditional independence $\mathcal{G}\perp\!\!\!\perp\mathcal{F}_{\{1:M-1\}}\,\big|\,\mathcal{F}_M$: For each atom $f_k:=\{\omega\in\Omega:a_M=k\}$ of $\mathcal{F}_M$, let A_k be the $(n\times n^{M-1})$ matrix with (i,j) entry $\mu(g_i,h_j,f_k)$, where $g_i:=\{\omega\in\Omega:s_M=i\}$ and $h_j:=\{\omega\in\Omega:a_{l+1}=j_l\text{ for }l=0,\ldots,M-2\}$ are atoms of $\mathcal{G}$ and $\mathcal{F}_{\{1:M-1\}}$, respectively, and $(j_{M-2}j_{M-3}\cdots j_0)$ is the n-ary expansion of j, so that $j=\sum_{l=0}^{M-2}j_l\,n^l$. Then it can be seen that the (i,j) entry

$$\mu(g_i,h_j,f_k)=1\left(i=k+\sum_{l=0}^{M-2}j_l\right)$$

where $1(\cdot)$ is the indicator function. Clearly, A_k has rank $(n-k)$ since it has precisely $(n-k)$ nonzero (linearly independent) rows; namely, rows k through $n-1$. Hence, we can split atom f_k into $(n-k)$ new atoms, corresponding to row-wise decomposition of A_k. Let $\mathcal{F}_M'$ be the σ-field whose atoms are these split atoms. Then

$$\begin{aligned}\mathcal{A}(\mathcal{F}_M') &= \{f_{l,m}':0\le l\le m\le n-1\}\\ &:=\big\{\{\omega\in\Omega:a_M=l\ \&s_M=m\}:0\le l\le m\le n-1\big\}.\end{aligned}$$

In particular, $\mathcal{F}_M'$ has $n(n+1)/2$ atoms.

Next we try to lift to create conditional independence $\mathcal{F}_M'\perp\!\!\!\perp\mathcal{F}_{\{1:M-2\}}\big|\mathcal{F}_{M-1}$: For each atom

$$f_k:=\{\omega\in\Omega:a_{M-1}=k\}$$

of $\mathcal{F}_{M-1}$, let A_k be the $(\frac{n(n+1)}{2}\times n^{M-2})$ matrix with (i,j) entry $\mu(f_{l_i,m_i}',h_j,f_k)$. Here, $f_{l,m}'$'s are the atoms of $\mathcal{F}_M'$ and

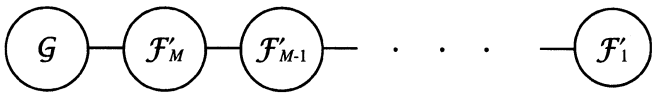

Fig. 4. Junction tree for Example 3.

(l_i, m_i) is a map from $\{0,\ldots,n(n+1)/2-1\}$ to $\{0,\ldots,n-1\}\times\{0,\ldots,n-1\}$ such that $l_i \le m_i$. Also

$$h_j := \{\omega \in \Omega : a_{l+1} = j_l \text{ for } l = 0,\ldots,M-3\}$$

are atoms of $\mathcal{F}_{\{1:M-2\}}$, and $(j_{M-3}j_{M-4}\cdots j_0)$ is the n-ary expansion of j, so that $j = \sum_{l=0}^{M-3} j_l\, n^l$. Again, it can be seen that the (i,j) entry

$$\mu(f'_{l_i,m_i}, h_j, f_k) = 1\left(m_i = l_i + k + \sum_{l=0}^{M-3} j_l\right).$$

It is evident that this function only depends on row index i through $m_i - l_i$; for a given $r \in \{0,\ldots,n-1\}$, all the rows with $m_i - l_i = r$ are identical, and hence can be grouped together for rank-one decomposition. Also notice that

$$\begin{aligned}\{f'_{l,m} &: 0 \le l \le m \le n-1,\, m-l = r\}\\ &= \Big\{\{\omega \in \Omega : a_M = l \ \&s_{M-1} = m-l\} :\\ &\qquad 0 \le l \le m \le n-1,\, m-l = r\Big\}\\ &= \Big\{\{\omega \in \Omega : s_{M-1} = r\}\Big\}.\end{aligned}$$

Further, each A_k has $(n-k)$ such groups of rows, corresponding to $k \le r \le n-1$. Hence, we can split atom f_k into $(n-k)$ new atoms. Let $\mathcal{F}'_{M-1}$ be the σ-field whose atoms are these split atoms. Then

$$\begin{aligned}&\mathcal{A}(\mathcal{F}'_{M-1})\\ &\quad = \Big\{\{\omega\in\Omega : a_{M-1}=l \ \&s_{M-1}=m\} : 0\le l\le m\le n-1\Big\}.\end{aligned}$$

Note that $\mathcal{F}'_{M-1}$ has $n(n+1)/2$ atoms.

This procedure will be repeated and it can be seen that the lifted σ-fields will have atoms

$$\mathcal{A}(\mathcal{F}'_i) = \Big\{\{\omega \in \Omega : a_i = l \& s_i = m\} : 0 \le l \le m \le n-1\Big\}.$$

The corresponding junction tree is the chain pictured in Fig. 4.

Examining the matrices of joint measures along edges of this chain, it can be verified that the corresponding junction tree algorithm is equivalent to the following:

$$\begin{aligned}Z = \sum_{s_M=0}^{n-1} \mu(s_M) \sum_{s_{M-1}=0}^{s_M} X_M(s_M - s_{M-1})\\ \cdots \sum_{s_2=0}^{s_3} X_3(s_3 - s_2) \sum_{s_1=0}^{s_2} X_2(s_2 - s_1) X_1(s_1).\end{aligned} \tag{17}$$

This requires only $O(Mn^2)$ arithmetic operations.

The form given in (17) suggests that, after introduction of variables s_i, GDL can also produce (17). However, variables s_i as defined above are not independent, so GDL can never come up with (17) exactly. To account for this problem in GDL, one can take s_i's to be free and independent variables taking value in $\{0,\ldots,n-1\}$, and then redefine functions $X_i(s_i, s_{i-1})$ to equal $X_i(s_i - s_{i-1})$ when $0 \le s_{i-1} \le s_i$, and be zero otherwise. Then, indeed, GDL will come up with a form similar to (17) (note that upper limits in the sums cannot depend on variable s_i; a postprocessor can, however, realize that the terms that correspond to $s_{i-1} > s_i$ will vanish.)

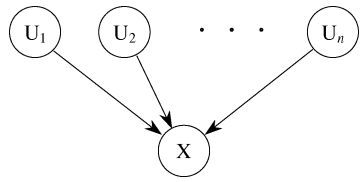

Fig. 5. Bayesian network of Example 4.

Notice, however, the *automatic* nature of our procedure: although we have introduced the auxiliary variables s_i to analytically express the atoms of each σ-field and to express the algorithm of (17) in closed form, the automatic procedure of Algorithm 4 does not rely on these variables. There is no need for pre- or post-processing of data or introduction of variables. Given a representation of the σ-fields $\mathcal{G}$ and $\mathcal{F}_1,\ldots,\mathcal{F}_M$, a computer program will automatically come up with algorithm of (17) without any assistance. This is the essence of our method. □

In the next example we show that Pearl's treatment of the belief propagation algorithm in the case of a node with *disjunctive interaction* or *noisy-or-gate* causation ([7, Sec. 4.3.2]) can be viewed as a special case of our algorithm.

Example 4: Bayesian Network With Disjunctive Interaction: Let the binary inputs $U = (U_1, U_2, \ldots, U_n)$ be the parents of the binary node X in the Bayesian network of Fig. 5, interacting on X through a noisy-or-gate.

This means that there are parameters $q_1,\ldots,q_n \in [1]$ so that

$$\begin{aligned}P(X = 0 \mid U) &= \prod_{i\in T_u} q_i\\ P(X = 1 \mid U) &= 1 - \prod_{i\in T_u} q_i\end{aligned}$$

where $T_u := \{i : U_i = 1\}$.

A normal moralization and triangulation technique applied to this graph will give a single clique with all the variables. However, because of the structure in the problem, a better solution exists.

Let $\mathcal{F}_X, \mathcal{F}_1, \ldots, \mathcal{F}_n$ be the σ-fields generated by the (independent) variables $x, u_1, \ldots, u_n$, respectively. All variables are binary so each of the σ-fields has precisely two atoms. In our framework, let the "random variables" be (the function) $1 \in \mathcal{F}_X$, and

$$\pi_X(u_i) = P(U_i = u_i) \in \mathcal{F}_i, \qquad \text{for } i = 1,\ldots,n$$

with the underlying joint measure on $x, u_1, \ldots, u_n$ defined to be

$$\mu(x, u_1, \ldots, u_n) = P(X = x \mid U = (u_1, \ldots, u_n)).$$

Then F_i's are *not* mutually conditionally independent given $\mathcal{F}_X$, however, the following simple lifting of space will create the independence: Let a variable x' be defined to take value in $0, 1, 2$, where the event $\{x' = 0\}$ corresponds to $\{x = 0\}$,

and $\{x' = 1\} \cup \{x' = 2\}$ correspond to $\{x = 1\}$. Extend the measure as follows:

$$\mu'(x', u_1, \ldots, u_n) = \begin{cases} \prod_{i \in T_u} q_i, & \text{if } x' = 0 \\ -\prod_{i \in T_u} q_i, & \text{if } x' = 1 \\ 1, & \text{if } x' = 2. \end{cases}$$

Then we see that in this *lifted* space, the σ-fields $\mathcal{F}'_i$ (generated by variables u_i, respectively) are mutually conditionally independent given $\mathcal{F}'_{X'}$ (the σ-field generated by variable x'). Then we have a junction tree in the shape of a star, with $\mathcal{F}'_{X'}$ corresponding to the central node. The junction tree algorithm will calculate the following marginalized random variable at the node corresponding to $\mathcal{F}'_{X'}$:

$$\beta(x') = \begin{cases} \prod_{i=1}^n \left(P(U_i = 0) + P(U_i = 1)\, q_i\right), & \text{if } x' = 0 \\ -\prod_{i=1}^n \left(P(U_i = 0) + P(U_i = 1)\, q_i\right), & \text{if } x' = 1 \\ 1, & \text{if } x' = 2. \end{cases}$$

Then the *belief* at X is the function

$$\mathrm{BEL}(x) = \begin{cases} \prod_{i=1}^n \left(P(U_i = 0) + P(U_i = 1)\, q_i\right), & \text{if } x = 0 \\ 1 - \prod_{i=1}^n \left(P(U_i = 0) + P(U_i = 1)\, q_i\right), & \text{if } x = 1 \end{cases}$$

where we have merged the atoms $x' = 1$ and $x' = 2$ of $\mathcal{F}'_{X'}$ to get back $x = 1$. This is essentially the same as [7, eq. (4.57)]. □

Example 5: Hadamard Transform: Let $x_1, \ldots, x_n$ be binary variables and let $f(x_1, \ldots, x_n)$ be a real function of x_i's. The Hadamard transform of f is defined as

$$g(y_1, \ldots, y_n) := \sum_{x_1, \ldots, x_n} \prod_{i=1}^n (-1)^{x_i y_i} f(x_1, \ldots, x_n)$$

where $y_1, \ldots, y_n$ are binary variables.

Since our framework is particularly useful when the underlying functions are structured, we consider the case when f is a symmetric function of $x_1, \ldots, x_n$, i.e., f depends only on the sum of the x_i's. Then it is easy to verify that when f is symmetric, its Hadamard transform g is also a symmetric function.

We now set up the problem in our framework. Let Ω be $\{0, 1\}^{2n}$ with elements $\omega = (x_1, \ldots, x_n, y_1, \ldots, y_n)$. Let $\mathcal{F}$ and $\mathcal{G}$ be the σ-fields in which, respectively, f and g are measurable; in our case, of symmetric f and g

$$\begin{aligned} \mathcal{A}(\mathcal{F}) &= \{\alpha_k\,,\ k = 0, \ldots, n\} \\ &= \left\{ \{\omega : \sum_i x_i = k\}\,, k = 0, \ldots, n \right\} \end{aligned}$$

and

$$\begin{aligned} \mathcal{A}(\mathcal{G}) &= \{\beta_k\,,\ k = 0, \ldots, n\} \\ &= \left\{ \{\omega : \sum_i y_i = k\},\ k = 0, \ldots, n \right\}. \end{aligned}$$

Next we note that all the factors involving terms $(-1)^{x_i y_i}$ can be summarized as a signed measure μ on $\mathcal{F} \vee \mathcal{G}$ as follows:

$$\begin{aligned} \mu(\alpha_j, \beta_k) &= \sum_{\omega \in \alpha_j \cap \beta_k} (-1)^{\sum_i x_i y_i} \\ &= \sum_{\substack{\omega : \sum_i x_i = j, \\ \sum_i y_i = k}} (-1)^{\sum_i x_i y_i} \\ &= \sum_{(x_1, \ldots, x_n) : \sum_i x_i = j} (-1)^{\sum_{i=1}^k x_i}. \end{aligned}$$

Note that μ can be stored in a $(n+1) \times (n+1)$ table.

Now we have a junction tree with only two nodes, corresponding to $\mathcal{F}$ and $\mathcal{G}$, and the marginalization is done as follows:

$$\begin{aligned} g(\beta_k) &= \boldsymbol{E}[f | \mathcal{G}] \\ &= \sum_{j=0}^n f(\alpha_j) \mu(\alpha_j, \beta_k) \end{aligned}$$

where

$$f(\alpha_j) = f\left((x_1, \ldots, x_n) : \sum_i x_i = j \right)$$

and

$$g(\beta_k) = g\left((y_1, \ldots, y_n) : \sum_i y_i = k \right).$$

This requires only n additions and $(n+1)$ multiplications for each of $(n+1)$ possible values of g. □

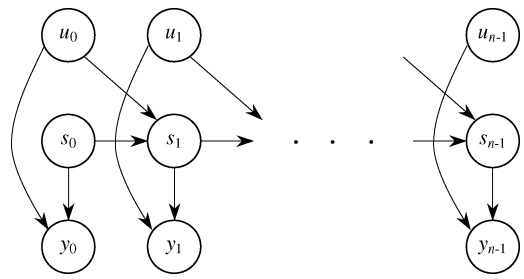

Fig. 6. Bayesian network for a probabilistic state machine.

Example 6: Probabilistic State Machine: Consider the Bayesian network depicted in Fig. 6, where u_i, s_i, and y_i denote inputs, hidden states, and the outputs of a chain of length n. Let m be the memory of the state machine, so that each state s_i can be taken to be $(u_{i-m}, \ldots, u_{i-1})$, where for ease of notation we fix $u_{-1} = u_{-2} = \cdots := 0$ so we will not have to worry about stages $i < m$.

A specific device used to find the maximum-likelihood input symbols given y_i's is a trellis, which is equipped with an efficient marginalization algorithm, namely, the BCJR algorithm on the trellis (see [6]). As mentioned before, however, this is not a general method, but rather a method tailored only for a specific class of marginalization problems. Description of that algorithm in GDL format with variables requires introduction of complicated hidden variables (see Example 9).

The general and automatic GDL solution for this problem is the BCJR algorithm on a junction chain (rather than a trellis). The functions involved are $P(s_0), P(u_i), P(y_i^* | u_i, s_i)$ and $P(s_i | u_{i-1}, s_{i-1})$, so the GDL local domains for this chain are $\{s_0\}$, and $\{u_i\}$ and $\{u_{i-m}, \ldots, u_i\}$ for $i = 1, \ldots, n-1$. Assuming binary inputs and outputs, the BCJR algorithm will require about $3 \cdot \sum_v d(v)\, q_v$ operations, where v ranges over the GDL local domains, $d(v)$ is the number of neighbors of v, and $q_v = 2^{|v|}$ is the size of the set of possible values for the variables in domain v (see [5]). The complexity of GDL then

TABLE I
COMPARISON BETWEEN COMPLEXITY OF GDL AND PROBABILISTIC GDL

(n,m)	(9,5)	(9,6)	(10,5)	(10,6)	(10,7)	(11,5)	(11,6)	(11,7)	(12,5)	(12,6)	(12,7)
nz	70	70	82	84	82	94	98	98	106	112	114
GDL ops	1710	2670	2094	3438	5358	2478	4206	6894	2862	4974	8430
PGDL ops	292	292	343	352	343	394	412	412	445	472	481

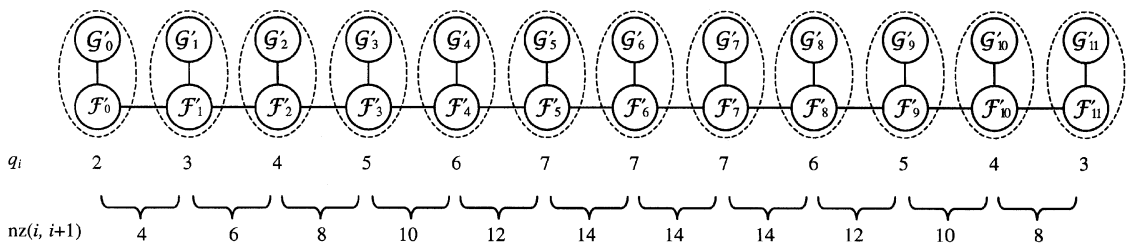

Fig. 7. Junction tree created for chain of length 12 and memory 6.

grows roughly as $3(n-m)2^{m+2}$ (the exact formula used in Table I is $3(2^{m+1}(2n-2m+1)-6)$).

Now consider a case when the output of the state machine depends on the input and state in a simple, but nonproduct form. For the purposes of this example, we have chosen the output y_i to be the outcome of an "OR" gate on the state s_i and input u_i, passed through a binary symmetric channel, i.e.,

$$P(y_i|u_i,s_i)=(1-p)1(y_i=\vee_{j=0}^{m}u_{i-j})+p\cdot 1(y_i\neq\vee_{j=0}^{m}u_{i-j})$$

where "$\vee$" indicates a logical "OR," and $1(\cdot)$ is the indicator function.

We formed the Ω space as $\{0,1\}^n$, with elements $\omega=(u_0,\ldots,u_{n-1})$. Then each function $P(y_i^*|u_i,s_i)$ is measurable in a σ-field $\mathcal{F}_i$, with two atoms $\{\omega:\vee_{j=0}^{m}u_{i-j}=0\}$ and $\{\omega:\vee_{j=0}^{m}u_{i-j}=1\}$. Since we like to calculate the posterior probabilities on each input u_i, we also include σ-fields $\mathcal{G}_i$ each with two atoms $\{\omega:u_i=0\}$ and $\{\omega:u_i=1\}$.

We then run our algorithm to create a junction tree on the $\mathcal{F}_i$'s and the $\mathcal{G}_i$'s, lifting the space whenever needed. The result is a chain consisting of the $\mathcal{F}'_i$'s with each $\mathcal{G}'_i$ hanging from its corresponding $\mathcal{F}'_i$ (see Fig. 7).

We have run the algorithm on chains with various values of n and m. Table I compares the complexity of the message-passing algorithm on the probabilistic junction tree and the normal GDL (BCJR) algorithm. GDL complexity was estimated as discussed above. The complexity of probabilistic algorithm on the lifted chain, as discussed in Section VI, is at most $(4\,\mathrm{nz})$ where "nz" is the total number of nonzero entries in the tables of pairwise joint measures. We add to this the number of additions required to get the desired marginals at the level of atoms of the original σ-fields to obtain the figures listed in the table.

The details of the case $n=12$, $m=6$ have been portrayed in Fig. 7. The number underneath each $\mathcal{F}'_i$ is q_i, the number of atoms of $\mathcal{F}'_i\vee\mathcal{G}'_i$ after lifting has been done. Note that with our setup, originally $\mathcal{F}_0\vee\mathcal{G}_0$ has two atoms, and all other $\mathcal{F}_i\vee\mathcal{G}_i$'s have three atoms. The numbers under the brackets denote $\mathrm{nz}(i,i+1)$, the number of nonzero elements in the matrix of joint measures between the atoms of adjacent nodes. Here we note that the pattern that can be seen in these sequences of numbers is not a coincidence. We will re-examine this general problem at the end of this section. □

Example 7: Exact Decoding of LDPC Codes: In this example, we apply our method to some LDPC codes of small block size over a memoryless channel. The codes used in this example are depicted in Fig. 8 as bipartite graphs, in which the upper nodes correspond to the bits and the lower nodes correspond to the parity checks (see [2]). In each case, we will obtain an exact algorithm to find the *a posteriori* probabilities for each bit. We then compare these algorithms with the exact algorithms obtained under GDL using the triangulation method (see [4] and [5]). As we will see, for a randomly generated LDPC code, the cliques of the triangulated graph are almost as big as the whole variable set, resulting in poor algorithms. On the other hand, using our framework we are able to find exact algorithms that are much less complex. In fact, as we will show later, the algorithms derived using our method in this case are equivalent to the best known exact decoding algorithms for linear block codes, i.e., the BCJR on the minimal trellis of the code (see, e.g., [13], [6]).

Let $\boldsymbol{H}\in\mathrm{GF}(2)^{m\times n}$ be the parity-check matrix for a binary linear block code with m check nodes and block size n. Then the column vector $\boldsymbol{x}=[x_1\cdots x_n]^T$ in GF(2)^n is a codeword iff it satisfies $\boldsymbol{H}\boldsymbol{x}=\boldsymbol{0}$. Given observations $\boldsymbol{y}^*$ of $\boldsymbol{x}$, the *a posteriori* joint probability of $\boldsymbol{x}$ factors as

$$P^*(x_1,\ldots,x_n)=\frac{1}{Z}\prod_{i=1}^{n}P(x_i)P(y_i^*|x_i)\prod_{j=1}^{m}1(H_j\cdot\boldsymbol{x}=0).$$

Here again, Z is a normalizing factor and H_j is the jth row of the matrix $\boldsymbol{H}$. We are interested in finding $P^*(x_i)$'s, the marginals of P^* for $i\in\{1,\ldots,n\}$.

To set up the problem in our framework, we define the sample space Ω to be the set of codewords, i.e.,

$$\Omega:=\{(x_1,\ldots,x_n):\boldsymbol{H}\boldsymbol{x}=\boldsymbol{0}\}.$$

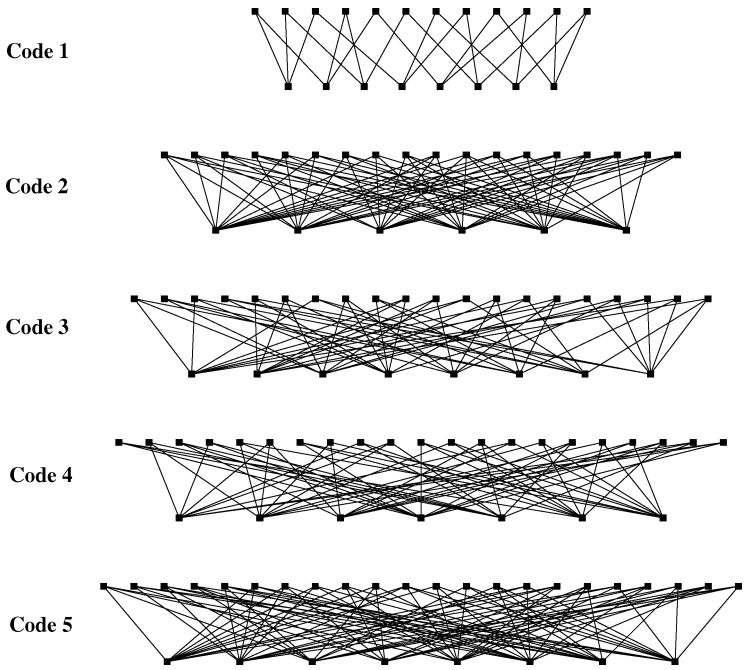

Fig. 8. LDPC codes of Example 7.

We choose the uniform measure on Ω, so that $\mu(\boldsymbol{x}) = 1$ for all $\boldsymbol{x} \in \Omega$. For $i = 1, \ldots, n$, define σ-field $\mathcal{F}_i$ with atoms

$$\mathcal{A}(\mathcal{F}_i) = \big\{\{\boldsymbol{x} \in \Omega : x_i = j\} : j = 0, 1\big\}.$$

Finally, we define random variables $X_i \in \mathcal{F}_i$ to equal $P(x_i = j)P(y_i^*|x_i = j)$. Then the marginals $P^*(x_i)$ correspond to the conditionals expectations $\boldsymbol{E}\big[\prod_j X_j \big| \mathcal{F}_i\big]$.

We ran our MATLAB lifting code for a number of randomly chosen LDPC codes with small block size (Fig. 8), and in each case formed a junction chain with lifted σ-fields $\mathcal{F}'_1, \ldots, \mathcal{F}'_n$. Table II summarizes each chain.

Each code is presented with block length n and parity checks m, as well as code rate and parameter c, the number of 1's per each column of $\boldsymbol{H}$ (checks per bit). For each code, we have listed q_i's, the number of atoms of the lifted σ-fields $\mathcal{F}'_i$. We have also reported functions $\mathrm{nz}(i, i+1)$, the number of nonzero entries of the matrix of joint measures of the atoms of neighboring σ-fields; as discussed in Section VI, total arithmetic complexity of Algorithm 2 on the chain is at most $4\sum_i \mathrm{nz}(i, i+1)$. Finally, to get the marginal $P^*(x_i)$ from the conditional expectation $\boldsymbol{E}\big[\prod_j X_j \big| \mathcal{F}_i\big]$, we need $(q_i - 2)$ additions. We have, therefore, calculated and reported the total arithmetic complexity of our exact algorithm.

For each code, we also triangulated the moral graph and found the junction tree of the cliques suitable for GDL-type algorithm. Specifically, for each LDPC code we form the moral graph; this is a graph with n nodes corresponding to the bits of a codeword, and where an edge (i, j) exists iff bits i and j are involved in a common parity constraint. We then triangulate this graph using the algorithm given in [7, Sec. 3.2.4], by adding edges (chords) in each unchorded cycle of the graph with length at least 4. The cliques of this graph can be put on a junction tree.

In Table II, for each code we report the size of the cliques of the triangulated graph; the clique sizes should be compared with $\log_2$ of the number of atoms of the lifted σ-fields in the left column. We also report the "edge sizes," i.e., the size of the intersection of the cliques that are connected by an edge in the junction tree of the cliques; these in turn should be compared with $\log_2$ of the functions $\mathrm{nz}(i, i+1)$. Finally, we use $4\sum_{v\ clique} d(v)q_v$ as the (approximate) arithmetic complexity of GDL algorithm, where $d(v)$ is the number of neighbors of clique v in the junction tree and $q_v = 2^{|v|}$ is the cardinality of the set of possible values for variables in v (see [5]).

It can be seen that the triangulation method is incapable of recognizing the structure that exists within each parity-check term $1(H_j \cdot \boldsymbol{x} = 0)$. It is, therefore, forced to treat each indicator function as a complete function of the bit variables that appear in that check. As a result of this and the interconnection between the bits, clique sizes are almost as big as the block size n, resulting in very inefficient algorithms.

On the other hand, avoiding representation with variables, our measure-theoretic approach is able to discover minimal liftings that render the σ-fields independent, and hence come up with much more efficient marginalization algorithms. Since the general complexity bounds we have given on the lifting procedure in Section VI-C are exponential, it is not clear for how large a block size it will be practicable to have a complete implementation of the lifting algorithm. Our MATLAB implementations were far from optimized, and were run on a 850-MHz Pentium III Laptop, with 256 MB of memory. It took about half an hour for the largest reported code size, namely, $n = 22$. This suggests that a highly optimized implementation on a state-of-the-art computing cluster could carry out lifting for much larger sized codes.

We will now show that applying Algorithm 4 to the problem of decoding a linear block code will produce a junction tree which is equivalent to the minimal trellis representing the code. Let Ω be a binary linear (n, k) code, with codewords $\boldsymbol{x} \in \Omega$

TABLE II
LDPC RESULTS FROM EXAMPLE 7 (SEE ALSO FIG. 8)

Code 1: $n = 12, m = 8, c = 2$, rate=$5/12$	
Lifting (Algorithm 4)	Moralization/Triangulation
q_i: $2, 4, 4, 8, 4, 4, 8, 8, 8, 8, 4, 2$	Clique sizes: $5, 6, 6, 5, 5, 5$
$\mathrm{nz}(i, i+1)$: $4, 4, 8, 8, 4, 8, 8, 16, 8, 8, 4$	Edge sizes: $4, 5, 4, 3, 4$
Total PGDL complexity: $\mathbf{360}$	Total GDL complexity: $\mathbf{1.8 \times 10^3}$
Code 2: $n = 18, m = 6, c = 4$, rate=$2/3$	
Lifting (Algorithm 4)	Moralization/Triangulation
q_i: $2, 4, 8, 16, 16, 32, 64, 64, 64, 64, 32, 64, 64, 32, 16, 8, 4, 2$	Clique sizes: 18
$\mathrm{nz}(i, i+1)$: $4, 8, 16, 32, 32, 64, 128, 128, 128, 64, 64, 128, 64, 32, 16, 8, 4$	Edge sizes: –
Total PGDL complexity: $\mathbf{5.95 \times 10^3}$	Total GDL complexity: $\mathbf{1.0 \times 10^6}$
Code 3: $n = 20, m = 8, c = 3$, rate=$3/5$	
Lifting (Algorithm 4)	Moralization/Triangulation
q_i: $2, 4, 8, 16, 32, 64, 128, 256, 256, 512, 512, 256, 256, 128, 64, 32, 16, 8, 4, 2$	Clique sizes: $14, 16, 16, 16$
$\mathrm{nz}(i, i+1)$: $4, 8, 16, 32, 64, 128, 256, 512, 512, 1024, 512, 512, 256, 128, 64, 32, 16, 8, 4$	Edge sizes: $13, 15, 14$
Total PGDL complexity: $\mathbf{2.02 \times 10^4}$	Total GDL complexity: $\mathbf{1.4 \times 10^6}$
Code 4: $n = 21, m = 7, c = 3$, rate=$2/3$	
Lifting (Algorithm 4)	Moralization/Triangulation
q_i: $2, 4, 8, 16, 32, 64, 128, 256, 256, 256, 128, 128, 64, 64, 64, 64, 32, 16, 8, 4, 2$	Clique sizes: $16, 17, 17$
$\mathrm{nz}(i, i+1)$: $4, 8, 16, 32, 64, 128, 256, 512, 512, 256, 256, 128, 128, 128, 128, 64, 32, 16, 8, 4$	Edge sizes: $14, 15$
Total PGDL complexity: $\mathbf{2.14 \times 10^4}$	Total GDL complexity: $\mathbf{1.8 \times 10^6}$
Code 5: $n = 22, m = 8, c = 4$, rate=$7/11$	
Lifting (Algorithm 4)	Moralization/Triangulation
q_i: $2, 4, 8, 16, 32, 64, 64, 128, 128, 256, 256, 256, 256, 128, 128, 128, 64, 32, 16, 8, 4, 2$	Clique sizes: $21, 21$
$\mathrm{nz}(i, i+1)$: $4, 8, 16, 32, 64, 128, 128, 256, 256, 256, 512, 512, 256, 256, 256, 128, 64, 32, 16, 8, 4$	Edge sizes: 20
Total PGDL complexity: $\mathbf{2.40 \times 10^4}$	Total GDL complexity: $\mathbf{8.4 \times 10^6}$

defined by $\boldsymbol{H} \cdot \boldsymbol{x} = \mathbf{0}$. As before, we define μ to be a uniform measure on Ω, so that $\mu(\boldsymbol{x}) = 1$ for all $\boldsymbol{x} \in \Omega$, and for each $i = 1, \ldots, n$, define $\mathcal{F}_i$ as the σ-field whose atoms are the level sets of the ith bit of $\boldsymbol{x}$, i.e.,

$$\mathcal{A}(\mathcal{F}_i) = \big\{\{\boldsymbol{x} \in \Omega : x_i = j\} : j = 0, 1\big\}.$$

Let T be a rank-n trellis with binary labels. This is defined as a graph whose vertices are partitioned into $n+1$ *time stages*, $V_0, V_1, \ldots, V_n$, with $|V_0| = |V_n| = 1$, such that any edge in T connects vertices from neighboring time stages. Further, each edge is labeled with a symbol from the set $\{0, 1\}$. We denote by $E_{i-1,i}$ the set of edges between V_{i-1} and V_i, and for each edge e, we denote by $\lambda(e)$ the label associated with edge e. More generally, we define $E_{m,m+d}$ to be the collection of all paths $(e_{m,m+1}, \ldots, e_{m+d-1,m+d})$ between V_m and V_{m+d}, with $e_{j,j+1} \in E_{j,j+1}$. Also, for each path $p \in E_{m,m+d}$, we denote by $\lambda(p)$ the $(d \times 1)$ vector of the labels of the edges of p. We will further assume that if p and p' are two different paths in $E_{0,n}$, then $\lambda(p) \neq \lambda(p')$.

A trellis T is said to represent code Ω if the labeled paths of T from V_0 to V_n are precisely the codewords of Ω, i.e., $\Omega = \{\lambda(p),\ p \in E_{0,n}\}$. Therefore, we identify codewords of Ω by the paths $p \in E_{0,n}$ in T. With this representation, we can view measure function μ as a function of n-tuples of edges, such that $\mu(p) = 1$ iff p is a path in $E_{0,n}$. Also, using this representation

for Ω, the two atoms of $\mathcal{F}_i$ are precisely $\{(e_{0,1},\ldots,e_{n-1,n}) \in E_{0,n} : \lambda(e_{i-1,i}) = j\}$ for $j \in \{0,1\}$.

Now note that each trellis representing Ω can be viewed as a lifting $(\Omega, \{\mathcal{F}'_i\}, \mu)$ of $(\Omega, \{\mathcal{F}_i\}, \mu)$, where for each $i = 1,\ldots,n$, $\mathcal{F}'_i$ is defined as the σ-field whose atoms correspond to *all* the edges in $E_{i-1,i}$. Specifically

$$\mathcal{A}(\mathcal{F}'_i) = \{\{(e_{0,1},\ldots,e_{n-1,n}) \in E_{0,n} : e_{i-1,i} = e^0\}, \quad e^0 \in E_{i-1,i}\}.$$

Then the atoms of $\bigvee_{i=m}^{m+d} \mathcal{F}'_{i+1}$ correspond in similar way to the paths in $E_{m,m+d}$.

We now claim that for every trellis representing code Ω, the chain $\mathcal{F}'_1$—$\mathcal{F}'_2$—$\cdots$—$\mathcal{F}'_n$ is a junction chain. To see this, let e be an edge in $E_{m-1,m}$, corresponding to an atom of $\mathcal{F}'_m$. Also let

$$p^1 := (e^1_{0,1},\ldots,e^1_{m-2,m-1}) \in E_{0,m-1}$$

and

$$p^2 := (e^2_{m,m+1},\ldots,e^2_{n-1,n}) \in E_{m,n}$$

be representing atoms of $\bigvee_{i=1}^{m-1} \mathcal{F}'_i$ and $\bigvee_{i=m+1}^{n} \mathcal{F}'_i$, respectively. Then from definitions above, $\mu(p^1, e, p^2)$ is 1 precisely if (p^1, e, p^2) is a path in $E_{0,n}$, and is zero otherwise. In other words

$$\mu(p^1, e, p^2) = 1\big(\mathrm{fin}(e^1_{m-2,m-1}) = \mathrm{init}(e)\big) \times 1\big(\mathrm{fin}(e) = \mathrm{init}(e^2_{m,m+1})\big)$$

where for an edge $e \in E_{i-1,i}$, we define $\mathrm{init}(e) \in V_{i-1}$ and $\mathrm{fin}(e) \in V_i$ as the initial and final vertices of e, respectively. This means precisely that for a fixed $e \in E_{m-1,m}$, $\mu(p^1, e, p^2)$ factorizes as a product of functions of p^1 and p^2, and hence,

$$\bigvee_{i=1}^{m-1} \mathcal{F}'_i \perp\!\!\!\perp \bigvee_{i=m+1}^{n} \mathcal{F}'_i \big| \mathcal{F}'_m.$$

This proves that $\mathcal{F}'_1$—$\mathcal{F}'_2$—$\cdots$—$\mathcal{F}'_n$ is a junction chain.

Finally, recall that for any linear block code there is a minimal representing trellis that has the smallest number of vertices and edges at each time stage. The minimal trellis has the property that for any two initial (or terminal) paths $p^1, p^2 \in E_{0,m}$ (or $p^1, p^2 \in E_{m-1,n}$), we have $p^1 = p^2$ iff $\lambda(p^1) = \lambda(p^2)$. This means that the atoms of $\bigvee_{i=1}^{m} \mathcal{F}_i$ can be precisely identified by the paths in $E_{0,m}$ (which are also the atoms of $\bigvee_{i=1}^{m} \mathcal{F}'_i$). Similarly, the atoms of $\bigvee_{i=m}^{n} \mathcal{F}_i$ can be precisely identified by the paths in $E_{m-1,n}$ (which are also the atoms of $\bigvee_{i=m}^{n} \mathcal{F}'_i$). Starting Algorithm 4 with the original σ-fields $\{\mathcal{F}_i\}$, the lifting of $\mathcal{F}_i$ will then amount to rank-one decomposition of a matrix of 0's and 1's with nonoverlapping blocks of 1's (as mentioned in Section V-C). The lifted atoms will be none other than the edges in $E_{i-1,i}$, and the lifted σ-field will be precisely $\mathcal{F}'_i$. Therefore, Algorithm 4 in this case will produce a junction chain which is equivalent to the minimal trellis of the code. It is then easy to verify that Algorithm 2 on this junction tree is the same as the BCJR algorithm on the minimal trellis. □

Example 8: CH-ASIA: Our next example is CH-ASIA from [3, pp. 110–111]. The chain graph of Fig. 9 describes the dependencies between the variables.

The problem is to find the marginals of $P(S, A, L, T, B, E, D, C, X)$, which is the product of the following functions: $P(S)$, $P(A)$, $P(L|S)$, $P(T|A)$, $P(B|S)$, $1(E{=}L{\vee}T)$, $P(X|E)$, $f(C, D, B)$, $g(C, B, E)$, and $h(B, E)$.

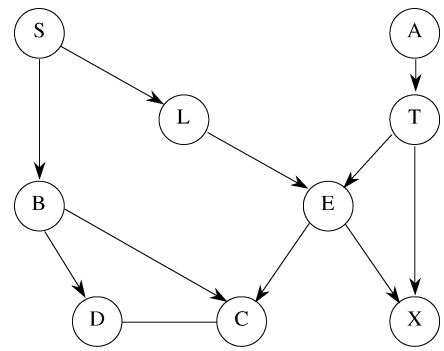

Fig. 9. Graph of CH-ASIA example.

Again, we set up measurable spaces, with σ-fields corresponding to each of the above functions. We then ran the lifting algorithm to find a junction tree in form of a chain, as in the previous example. This time, however, due to lack of structure at the level of the marginalizable functions (i.e., the aforementioned conditional probabilities), the algorithm produced exactly a junction tree that one could obtain by the process of moralization and triangulation at the level of original variables. In other words, all liftings were done by addition of one or more "whole" orthogonal directions (i.e., GDL variables) of the Ω space to the σ-fields. After reconverting σ-fields to "variables," the junction tree we obtained is equivalent to the one shown on Fig. 10. In this case, our algorithm has reduced to GDL. □

Example 9: Probabilistic State Machine Revisited: In the automatic treatment of Example 6 it was seen that the lifted junction chains exhibit strong structures, suggesting that there is a simple closed-form solution for the general problem. This in fact is the case, and as will be seen shortly our marginalization algorithm is equivalent to the trellis BCJR algorithm. Here, we simply report the closed-form representation of the general junction chain, and give the corresponding marginalization algorithm. We have the same underlying sample space Ω as in Example 6. For compact representation, we continue to use variables, viewing each specific value of a variable as an event, i.e., a subset of Ω. Now for each $i = 0,\ldots,n-1$ we define a variable v_i taking value in $\{0,\ldots,q_i - 1\}$, as follows.

- $v_0 := u_0$.
- For $i = 1,\ldots,m-1$:
$$v_i := \begin{cases} v_{i-1}, & \text{if } u_i = 0 \\ i+1, & \text{if } u_i = 1. \end{cases}$$
Note that here $q_i = i + 2$.
- For $i = m,\ldots,n-m+1$:
$$v_i := \begin{cases} \max(0, v_{i-1} - 1), & \text{if } u_i = 0 \\ m, & \text{if } u_i = 1 \end{cases}$$
with $q_i = m + 1$.
- For $i = n-m+2,\ldots,n-1$:
$$v_i := \begin{cases} \max(0, v_{i-1} - 2), & \text{if } u_i = 0 \\ n-i+1 & , \quad \text{if } u_i = 1 \end{cases}$$
with $q_i = n - i + 2$.

Once the lifting has been done, the resulting junction tree is a chain similar to that of Fig. 7 where atoms of the σ-fields are

Fig. 10. Junction tree for CH-ASIA example.

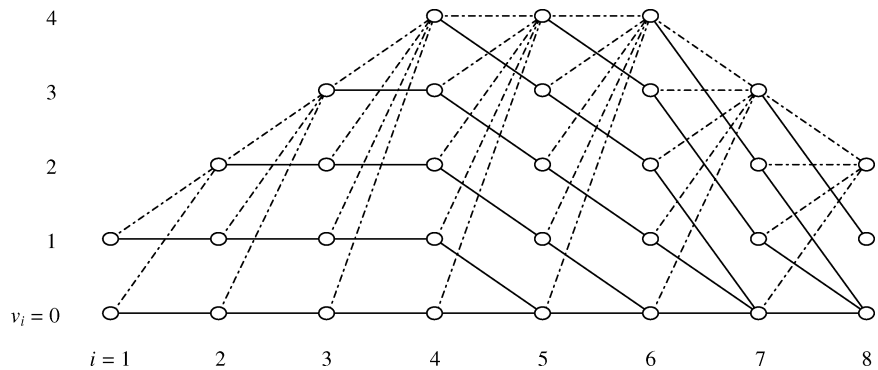

Fig. 11. Trellis of the state machine with $n = 9$ and $m = 4$. Solid and dashed lines correspond to 0 and 1 inputs, respectively.

given by the events $\{v_i = j\}$; more specifically, the jth atom of $\mathcal{F}_i' \vee \mathcal{G}_i'$ is $\{\omega \in \Omega : v_i = j\}$. Then q_i, the number of atoms of $\mathcal{F}_i' \vee \mathcal{G}_i'$, is the number of possible values for v_i as listed above. Also, in this case $\mathrm{nz}(i, i+1)$, the number of nonzero entries in the matrix of joint measures of atoms of $\mathcal{F}_i' \vee \mathcal{G}_i'$ and $\mathcal{F}_{i+1}' \vee \mathcal{G}_{i+1}'$ is simply $2q_i$. The total arithmetic complexity of implementing Algorithm 2 on this chain to solve the original marginalization problem is $(9nm + 6n + 18m - 7m^2 - 32)$, which should be compared to $3(n-m)2^{m+2}$ operations required by naive implementation of GDL.

Finally, we give the explicit message update rules of Algorithm 2 on this chain. Note that the original form of Algorithm 2 will involve multiplying each term below by a fixed weight, which we have simplified throughout the expressions. Also, for compactness, here we are defining $f_i(v_i)$ to equal the product $P(u_i)P(y_i^*|u_i, \ldots, u_{i-m})$, noting that both these functions are measurable given v_i: for a given value of v_i, the corresponding value for u_i is 1 if $v_i = q_i - 1$, and is 0, otherwise; also, the corresponding value for the term $\vee_{j=0}^{m} u_{i-j}$ is 0 if $v_i = 0$, and is 1 otherwise.

- For $i = 1, \ldots, m-1$:

$$Y_{i-1,i}(v_i) = \begin{cases} f_{i-1}(v_i)Y_{i-2,i-1}(v_i), & \text{if } v_i < q_i - 1 \\ \sum_{j=0}^{q_{i-1}-1} f_{i-1}(j)Y_{i-2,i-1}(j), & \text{if } v_i = q_i - 1 \end{cases}$$

and

$$Y_{i,i-1}(v_{i-1}) = f_i(q_i - 1)Y_{i+1,i}(q_i - 1) + f_i(v_{i-1})Y_{i+1,i}(v_{i-1})$$

- For $i = m, \ldots, n-m+1$:

$$Y_{i-1,i}(v_i) = \begin{cases} \sum_{j=0}^{1} f_{i-1}(j)Y_{i-2,i-1}(j), & \text{if } v_i = 0 \\ f_{i-1}(v_i+1)Y_{i-2,i-1}(v_i+1), & \text{if } 0 < v_i < q_i - 1 \\ \sum_{j=0}^{q_{i-1}-1} f_{i-1}(j)Y_{i-2,i-1}(j), & \text{if } v_i = q_i - 1 \end{cases}$$

and

$$Y_{i,i-1}(v_{i-1}) = f_i(q_i - 1)Y_{i+1,i}(q_i - 1) + f_i\big(\max(0, v_{i-1} - 1)\big)Y_{i+1,i}\big(\max(0, v_{i-1} - 1)\big)$$

- For $i = n-m+2, \ldots, n-1$:

$$Y_{i-1,i}(v_i) = \begin{cases} \sum_{j=0}^{2} f_{i-1}(j)Y_{i-2,i-1}(j), & \text{if } v_i = 0 \\ f_{i-1}(v_i+2)Y_{i-2,i-1}(v_i+2), & \text{if } 0 < v_i < q_i - 1 \\ \sum_{j=0}^{q_{i-1}-1} f_{i-1}(j)Y_{i-2,i-1}(j), & \text{if } v_i = q_i - 1 \end{cases}$$

and

$$Y_{i,i-1}(v_{i-1}) = f_i(q_i - 1)Y_{i+1,i}(q_i - 1) + f_i\big(\max(0, v_{i-1} - 2)\big)Y_{i+1,i}\big(\max(0, v_{i-1} - 2)\big).$$

Careful examination of these expressions reveals that the above algorithm is equivalent to the BCJR algorithm on the trellis tailored for this problem. Fig. 11 shows the corresponding trellis for $n = 9$ and $m = 4$, and the connection with variables v_i above.

It is evident that the original general version of GDL relying on variables is neither natural nor adequate in dealing with this problem. Although, as shown above, a description of the optimal marginalization algorithm in terms of some "variables" exists, it requires a careful preprocessing phase to discover such variables. Of course, in practice this inadequacy is addressed by invention of trellises, but that method is not a general method that is applicable to all classes of problems. Our lifting algorithm, on the other hand, is general and automatic in detecting the structures in any marginalization problem, and producing an efficient algorithm, without the need to discover hidden variables or to have any knowledge of trellises. Similarly, the eventual marginalization algorithm does not run on any trellis nor does it require complicated descriptions as shown in this example.

VIII. Conclusion

In this paper, we have developed a measure-theoretic version of the junction tree algorithm. We have generalized the notions of independence and junction trees at the level of σ-fields, and have produced algorithms to find or construct a junction tree on a given set of σ-fields. By taking advantage of structures at the atomic level of sample space Ω, our lifting algorithm is capable of automatically producing solutions less complex than the GDL-type algorithms. Although one can typically introduce new variables and redefine the local functions in a way

that GDL will also come up with the same efficient algorithm as our method, this requires an intelligent processor to examine the data and determine a good way of introducing these variables. This process, then, remains more of an art than science. As we saw through example, our framework is capable of automating this process.

The cost of generating a junction tree with Algorithm 4 is exponential in the size of the problem, and so is the size of any complete representation of the sample space Ω. Once a junction tree has been constructed, however, the algorithm will only depend on the joint measure of the atoms of adjacent pairs of σ-fields on the tree. This means that an algorithm which was build by considering an Ω space, with exponentially many elements, can be stored compactly and efficiently and used for all combinations of input functions. Therefore, the cost of generating an efficient marginalization algorithm is amortized over many uses.

Using our framework, the tradeoff between the construction complexity of junction trees and the overall complexity of the marginalization algorithm can be made with an appropriate choice for the representation of the measurable spaces; at one extreme, one considers the complete sample space, taking advantage of all the possible structures, and at the other, one represents the sample space with independent variables (i.e., orthogonal directions), in which case our framework reduces to GDL, both in concept and in implementation

The validity of this theory for the *signed* measures is of enormous convenience; it allows for introduction of atoms of negative weight in order to create independencies. This greatly simplifies the task of lifting, which now can be done using standard techniques such as singular value decomposition. By contrast, the problem of finding a *positive* rank-one decomposition of a positive matrix (which would arise if one confined the problem to the positive measures functions) is a hard problem (see [14]). Meanwhile, the complexities and abstractions due to use of signed measure theory are transparent to the end user: the eventual marginalization algorithm will only consist of additions and multiplications of the original data.

The measure-theoretic framework is the natural choice that is capable of discovering all the structure inherent in a problem. The extension of the GDL to this framework is quite natural and mathematically elegant. Together with the lifting algorithm, we feel that this approach can have strong practical as well as theoretical significance.

Appendix A
Proof of Theorem 1

Let f, g, x, and y be arbitrary elements of $\mathcal{F}, \mathcal{G}, \mathcal{X}$, and $\mathcal{Y}$, respectively. The proofs below consider all possible cases for the value of the μ on these atoms.

(5a): Symmetry is obvious, since $f \cap g = g \cap f$.

(5b): If $\mu(y) = 0$, then $\mu(f, g, x, y) = 0$ and if $\mu(y) \neq 0$, then

$$\mu(f, g, x, y) = \frac{\mu(f, y)\mu(g, x, y)}{\mu(y)}$$

for all choices of f, g, and x in $\mathcal{F}, \mathcal{G}$, and $\mathcal{X}$, respectively. In particular, ranging x over all atoms of $\mathcal{X}$ and summing the above equation yields the first part of result, since

$$\sum_{x \in \mathcal{A}(\mathcal{X})} \mu(f, g, x, y) = \mu(f, g, y)$$

and

$$\sum_{x \in \mathcal{A}(\mathcal{X})} \mu(g, x, y) = \mu(g, y).$$

Similarly, ranging g over all atoms of $\mathcal{G}$ and summing yields the second part of the result.

(5c):

- $\mu(x) = 0$. Then from $\mathcal{F} \perp\!\!\!\perp \mathcal{G} \big| \mathcal{X}$, we get $\mu(g, x) = 0$ for all g. Then from $\mathcal{F} \perp\!\!\!\perp \mathcal{Y} \big| \mathcal{G} \vee \mathcal{X}$ we get $\mu(f, g, x, y) = 0$ for all f and y, and so we are done.
- $\mu(x) \neq 0$ and $\mu(g, x) = 0$. Then from $\mathcal{F} \perp\!\!\!\perp \mathcal{Y} \big| \mathcal{G} \vee \mathcal{X}$ we get $\mu(f, g, x, y) = 0$ for all f and y, and so $\mu(f, g, x, y)\mu(x) = \mu(f, x)\mu(g, x, y) = 0$ and we are done.
- $\mu(x) \neq 0$ and $\mu(g, x) \neq 0$.Then from $\mathcal{F} \perp\!\!\!\perp \mathcal{Y} \big| \mathcal{G} \vee \mathcal{X}$ we get

$$\mu(f, g, x, y) = \mu(f, g, x)\mu(g, x, y)/\mu(g, x). \quad (18)$$

Also, from $\mathcal{F} \perp\!\!\!\perp \mathcal{G} \big| \mathcal{X}$ we have $\mu(f, g, x)/\mu(g, x) = \mu(f, x)/\mu(x)$. Replacing this into (18) we obtain $\mu(f, g, x, y) = \mu(g, x, y)\mu(f, x)/\mu(x)$ and we are done.

(5d):

- $\mu(x) = 0$. Then from $\mathcal{F} \vee \mathcal{G} \perp\!\!\!\perp \mathcal{Y} \big| \mathcal{X}$, we get $\mu(f, g, x, y) = 0$ for all f, g, and y, and so we are done.
- $\mu(x) \neq 0$ and $\mu(x, y) = 0$. Then from $\mathcal{F} \vee \mathcal{G} \perp\!\!\!\perp \mathcal{Y} \big| \mathcal{X}$ we get $\mu(f, g, x, y) = \mu(f, g, x)\mu(x, y)/\mu(x) = 0$ for all f, g, and y, and in particular $\mu(g, x, y) = 0$ and so we have the desired equality $\mu(f, g, x, y)/\mu(x) = \mu(f, x)\mu(g, x, y) = 0$.
- $\mu(x) \neq 0$ and $\mu(x, y) \neq 0$. We have

$$\mu(f, g, x) = \mu(f, x)\mu(g, x)/\mu(x) \quad \text{since } \mathcal{F} \perp\!\!\!\perp \mathcal{G} \big| \mathcal{X} \quad (19)$$

$$\mu(f, g, x, y) = \mu(f, g, x)\mu(x, y)/\mu(x) \quad \text{since } \mathcal{F} \vee \mathcal{G} \perp\!\!\!\perp \mathcal{Y} \big| \mathcal{X} \quad (20)$$

$$\mu(g, x)/\mu(x) = \mu(g, x, y)/\mu(x, y) \quad \text{since by (5b), } \mathcal{G} \perp\!\!\!\perp \mathcal{Y} \big| \mathcal{X}. \quad (21)$$

Replacing (21) into (19) and then into (20) we obtain

$$\mu(f, g, x, y) = \mu(f, x)\mu(g, x, y)/\mu(x).$$

(5e):

- $\mu(x) = 0$ and $\mu(g) = 0$. Then from $\mathcal{F} \vee \mathcal{X} \perp\!\!\!\perp \mathcal{Y} \big| \mathcal{G}$, we get $\mu(f, g, x, y) = 0$ and we are done.
- $\mu(x) = 0$ and $\mu(g) \neq 0$. From $\mathcal{F} \perp\!\!\!\perp \mathcal{G} \big| \mathcal{X}$ we have $\mu(f, g, x) = 0$. Then from $\mathcal{F} \vee \mathcal{X} \perp\!\!\!\perp \mathcal{Y} \big| \mathcal{G}$, $\mu(f, g, x, y) = \mu(f, g, x)\mu(y, g)/\mu(g) = 0$ and we are done.

- $\mu(x) \neq 0$ and $\mu(g) = 0$. Then from $\mathcal{F} \vee \mathcal{X} \perp\!\!\!\perp \mathcal{Y} \,\big|\, \mathcal{G}$, we get both $\mu(f,g,x,y) = 0$ and $\mu(g,x,y) = 0$, so the desired equality hold
$$\mu(f,g,x,y) = \mu(f,x)\mu(g,x,y)/\mu(x) = 0.$$
- $\mu(x) \neq 0$ and $\mu(g) \neq 0$. Then from $\mathcal{F} \vee \mathcal{X} \perp\!\!\!\perp \mathcal{Y} \,\big|\, \mathcal{G}$, we get $\mu(f,g,x,y) = \mu(f,g,x)\mu(g,y)/\mu(g)$. Also, from $\mathcal{F} \perp\!\!\!\perp \mathcal{G} \,\big|\, \mathcal{X}$ we have
$$\mu(f,g,x) = \mu(f,x)\mu(g,x)/\mu(x).$$
So we obtain the equality
$$\mu(f,g,x,y) = \mu(f,x)\mu(g,x)\mu(g,y)/(\mu(g)\mu(x)).$$
Finally, *decomposition* applied to $\mathcal{F} \vee \mathcal{X} \perp\!\!\!\perp \mathcal{Y} \,\big|\, \mathcal{G}$ yields $\mu(g,x)\mu(g,y)/\mu(g) = \mu(g,x,y)$. So we have proved $\mu(f,g,x,y) = \mu(f,x)\mu(g,x,y)/\mu(x)$ and this completes the proof.

(5f):
- $\mu(x) = 0$. Then from $\mathcal{F} \vee \mathcal{G} \perp\!\!\!\perp \mathcal{Y} \,\big|\, \mathcal{X}$ we have $\mu(f,g,x,y) = 0$ and we are done.
- $\mu(x) \neq 0$ and $\mu(x,y) = 0$. Then from $\mathcal{F} \vee \mathcal{G} \perp\!\!\!\perp \mathcal{Y} \,\big|\, \mathcal{X}$ we have
$$\mu(f,g,x,y) = \mu(f,g,x)\mu(x,y)/\mu(x)$$
and so $\mu(f,g,x,y) = 0$. Also, after applying (5b) to the above, we have
$$\mu(f,x,y) = \mu(f,x)\mu(x,y)/\mu(x) = 0$$
and
$$\mu(g,x,y) = \mu(g,x)\mu(x,y)/\mu(x) = 0.$$
So we have the equality
$$\begin{aligned}\mu(f,g,x,y)\mu(x) &= \mu(f,x,y)\mu(g,x)\\ &= \mu(f,x)\mu(g,x,y)\\ &= 0\end{aligned}$$
and we are done.
- $\mu(x) \neq 0$ and $\mu(x,y) \neq 0$. Then also $\mu(y) \neq 0$ or else from $\mathcal{F} \perp\!\!\!\perp \mathcal{G} \vee \mathcal{X} \,\big|\, \mathcal{Y}$ we would have $\mu(x,y) = 0$. Then from $\mathcal{F} \perp\!\!\!\perp \mathcal{G} \vee \mathcal{X} \,\big|\, \mathcal{Y}$ we get
$$\mu(f,g,x,y) = \mu(f,y)\mu(g,x,y)/\mu(y)$$
and also after (5b) to the above, we get
$$\mu(f,x,y)/\mu(x,y) = \mu(f,y)/\mu(y).$$
Replacing the latter equation into the former we obtain
$$\mu(f,g,x,y) = \mu(g,x,y)\mu(f,x,y)/\mu(x,y). \tag{22}$$
But from $\mathcal{F} \vee \mathcal{G} \perp\!\!\!\perp \mathcal{Y} \,\big|\, \mathcal{X}$ and by (5b) we have both
$$\mu(f,x,y)/\mu(x,y) = \mu(f,x)/\mu(x)$$
and
$$\mu(g,x,y)/\mu(x,y) = \mu(g,x)/\mu(x).$$
Replacing each of these into (22) we obtain
$$\mu(f,g,x,y) = \mu(g,x,y)\mu(f,x)/\mu(x)$$
and
$$\mu(f,g,x,y) = \mu(f,x,y)\mu(g,x)/\mu(x)$$
and we are done. □

Appendix B
Proof of Correctness of Algorithm 2

In this appendix, we give a proof for the correctness of the Probabilistic Junction Tree Algorithm 2. We will use a proof that parallels that given in [5]. We will need the following lemmas.

Lemma 6: Suppose there exists a junction tree with nodes corresponding to σ-fields $\{\mathcal{F}_1, \ldots, \mathcal{F}_M\}$. Then if f is a zero-measure atom of any of the $\mathcal{F}_i$'s, and $g \subset f$ is measurable in $\bigvee_{i=1}^{M} \mathcal{F}_i$, then $\mu(g) = 0$.

Proof: Node i vacuously separates the empty subset of $\{1, \ldots, M\}$ from $\{1, \ldots, M\} \backslash \{i\}$. Thus,
$$\{\emptyset, \Omega\} \perp\!\!\!\perp \bigvee_{\substack{j=1\\ j\neq i}}^{M} \mathcal{F}_j \,\big|\, \mathcal{F}_i.$$
Thus, by the definition of conditional independence, whenever $f \in \mathcal{A}(\mathcal{F}_i)$ has zero measure, all its subsets measurable in $\bigvee_{i=1}^{M} \mathcal{F}_i$ also have measure zero. □

Lemma 7: (cf. [5, Lemma A.1]) Let $\mathcal{F}_1, \mathcal{F}_2$, and $\mathcal{F}_3$ be σ-fields such that $\mathcal{F}_1 \perp\!\!\!\perp \mathcal{F}_3 \,\big|\, \mathcal{F}_2$. Then for any partially defined random variable $X \in \mathcal{F}_1$, the following equality holds:
$$\boldsymbol{E}\Big[\boldsymbol{E}\big[X\big|\mathcal{F}_2\big]\Big|\mathcal{F}_3\Big] = \boldsymbol{E}\big[X\big|\mathcal{F}_3\big].$$

Proof: Let $Y = \boldsymbol{E}\big[X\big|\mathcal{F}_2\big]$. Then
$$\mathcal{A}_Y(\mathcal{F}_2) = \mathcal{A}'(\mathcal{F}_2) = \{b \in \mathcal{A}(\mathcal{F}_2) : \mu(b) \neq 0\}$$
and for $b \in A_Y(\mathcal{F}_2)$
$$Y(b) = \frac{1}{\mu(b)} \sum_{a \in A_X(\mathcal{F}_1)} X(a)\mu(a,b).$$
Then, for any nonzero-measure atom $c \in \mathcal{A}(\mathcal{F}_3)$
$$\begin{aligned}&\boldsymbol{E}\Big[\boldsymbol{E}\big[X\big|\mathcal{F}_2\big]\Big|\mathcal{F}_3\Big](c)\\ &\quad= \boldsymbol{E}\big[Y\big|\mathcal{F}_3\big](c)\\ &\quad= \frac{1}{\mu(c)} \sum_{b \in A_Y(\mathcal{F}_2)} Y(b)\mu(b,c)\\ &\quad= \frac{1}{\mu(c)} \sum_{b \in A_Y(\mathcal{F}_2)} \frac{1}{\mu(b)} \sum_{a \in A_X(\mathcal{F}_1)} X(a)\mu(a,b)\mu(b,c)\\ &\quad\overset{(a)}{=} \frac{1}{\mu(c)} \sum_{b \in A_Y(\mathcal{F}_2)} \sum_{a \in A_X(\mathcal{F}_1)} X(a)\mu(a,b,c)\\ &\quad= \frac{1}{\mu(c)} \sum_{a \in A_X(\mathcal{F}_1)} X(a) \sum_{b \in A_Y(\mathcal{F}_2)} \mu(a,b,c)\\ &\quad\overset{(b)}{=} \frac{1}{\mu(c)} \sum_{a \in A_X(\mathcal{F}_1)} X(a)\mu(a,c)\\ &\quad= \boldsymbol{E}\big[X\big|\mathcal{F}_3\big](c)\end{aligned}$$

where equality (a) follows from the fact that $\mathcal{F}_1 \perp\!\!\!\perp \mathcal{F}_3 \,|\, \mathcal{F}_2$, and (b) follows from fact that $\mathcal{F}_1 \perp\!\!\!\perp \mathcal{F}_3 \,|\, \mathcal{F}_2$, so that

$$\sum_{b\in\mathcal{A}_Y(\mathcal{F}_2)} \mu(a,b,c) = \sum_{b\in\mathcal{A}_Y(\mathcal{F}_2)} \mu(a,b,c) = \mu(a,c). \qquad \square$$

Lemma 8: (cf. [5, Lemma A.2]) Let $\{\mathcal{F}_1,\ldots,\mathcal{F}_l\}$ and $\mathcal{F}$ be σ-fields such that $\{\mathcal{F}_1,\ldots,\mathcal{F}_l\}$ are mutually conditionally independent given $\mathcal{F}$. For each $i=1,\ldots,l$, let X_i be a partially defined random variable in $\mathcal{F}_i$. Then

$$\boldsymbol{E}\Big[\prod_{i=1}^{l} X_i \Big| \mathcal{F}\Big] = \prod_{i=1}^{l} \boldsymbol{E}\big[X_i \big| \mathcal{F}\big].$$

Proof: We shall proceed by induction. The statement is vacuous for $l=1$. For $l=2$, let $Y = \boldsymbol{E}\big[X_1X_2\big|\mathcal{F}\big]$. Then $\mathcal{A}_Y(\mathcal{F}) = \{f\in\mathcal{A}(\mathcal{F}) : \mu(f)\neq 0\}$. Also note that X_1X_2 is a partially defined random variable in $\mathcal{F}_1\vee\mathcal{F}_2$ with

$$\mathcal{A}_{X_1X_2}(\mathcal{F}_1\vee\mathcal{F}_2) = \mathcal{A}_{X_1}(\mathcal{F}_1\vee\mathcal{F}_2)\cap\mathcal{A}_{X_2}(\mathcal{F}_1\vee\mathcal{F}_2)$$

and that any atom of $\mathcal{F}_1\vee\mathcal{F}_2$ can be written as $a\cap b$ for $a\in\mathcal{A}(\mathcal{F}_1)$ and $b\in\mathcal{A}(\mathcal{F}_2)$. Then for any $f\in\mathcal{A}_Y(\mathcal{F})$ we have

$$\begin{aligned}
Y(f) &= \frac{1}{\mu(f)} \sum_{\substack{(a\cap b)\in\mathcal{A}_{X_1X_2}(\mathcal{F}_1\vee\mathcal{F}_2)\\ a\in\mathcal{F}_1, b\in\mathcal{F}_2}} X_1(a)X_2(b)\mu(a,b,f)\\
&= \frac{1}{\mu(f)} \sum_{a\in\mathcal{A}_{X_1}(\mathcal{F}_1)} \sum_{b\in\mathcal{A}_{X_2}(\mathcal{F}_2)} X_1(a)X_2(b)\frac{\mu(a,f)\mu(b,f)}{\mu(f)}\\
&= \frac{1}{\mu(f)} \sum_{a\in\mathcal{A}_{X_1}(\mathcal{F}_1)} X_1(a)\mu(a,f)\\
&\quad\times \frac{1}{\mu(f)} \sum_{b\in\mathcal{A}_{X_2}(\mathcal{F}_2)} X_2(b)\mu(b,f)\\
&= \boldsymbol{E}\big[X_1\big|\mathcal{F}\big](f)\times\boldsymbol{E}\big[X_2\big|\mathcal{F}\big](f)
\end{aligned}$$

where we have used the fact that $\mathcal{F}_1 \perp\!\!\!\perp \mathcal{F}_2 \,|\, \mathcal{F}$, so

$$\frac{\mu(a,f)\mu(b,f)}{\mu(f)} = \mu(a,b,f).$$

For $l>2$ assume inductively that the equality holds for $l-1$. Then

$$\begin{aligned}
&\boldsymbol{E}\Big[\prod_{i=1}^{l} X_i \Big| \mathcal{F}\Big]\\
&\quad= \boldsymbol{E}\Big[X_1\prod_{i=2}^{l} X_i \Big| \mathcal{F}\Big]\\
&\quad= \boldsymbol{E}\big[X_1\big|\mathcal{F}\big]\boldsymbol{E}\Big[\prod_{i=2}^{l} X_i \Big| \mathcal{F}\Big] \quad \text{since } \mathcal{F}_1 \perp\!\!\!\perp \bigvee_{i=2}^{l} \mathcal{F}_i \,|\mathcal{F}\\
&\quad= \prod_{i=1}^{l} \boldsymbol{E}\big[X_i\big|\mathcal{F}\big] \quad \text{by induction hypothesis} \qquad \square
\end{aligned}$$

Using Lemmas 7 and 8, we are in position to prove a version of the scheduling [5, Theorem 3.1] for our measure-theoretic framework.

Proof of Correctness of Algorithm 2: We will show that if E_t is the schedule for activation of the nodes (i.e., a directed edge $(i,j)\in E_t$ iff node i updates its message to its neighbor j at time t) then the message from a node i to a neighboring node j is

$$Y_{i,j}(t) = \boldsymbol{E}\Big[\prod_{k\in K_{i,j}(t)} X_k \Big| \mathcal{F}_j\Big] \tag{23}$$

where $K_{i,j}(t)$ is a subset of the nodes defined recursively by

$$K_{i,j}(t) = \begin{cases} \emptyset, & \text{if } t=0\\ K_{i,j}(t-1), & \text{if } (i,j)\notin E_t\\ \{i\}\bigcup_{l\in N_{i,j}} K_{l,i}(t-1), & \text{if } (i,j)\in E_t. \end{cases}$$

We will prove this by induction on t. Case $t=0$ is clear from the initialization. Now let $t>0$ and assume that (23) holds for $t-1$. We can also assume that the $(i,j)\in E_t$ so the message $Y_{i,j}$ is being updated at time t. Then

$$\begin{aligned}
Y_{i,j}(t) &= \boldsymbol{E}\Big[X_i \prod_{l\in N_{i,j}} Y_{l,i} \Big| \mathcal{F}_j\Big]\\
&\overset{(a)}{=} \boldsymbol{E}\Big[X_i \prod_{l\in N_{i,j}} \boldsymbol{E}\Big[\prod_{k\in K_{l,i}(t-1)} X_k \Big| \mathcal{F}_i\Big]\Big| \mathcal{F}_j\Big]\\
&\overset{(b)}{=} \boldsymbol{E}\Big[\boldsymbol{E}\Big[X_i \prod_{l\in N_{i,j}} \prod_{k\in K_{l,i}(t-1)} X_k \Big| \mathcal{F}_i\Big]\Big| \mathcal{F}_j\Big]\\
&\overset{(c)}{=} \boldsymbol{E}\Big[X_i \prod_{l\in N_{i,j}} \prod_{k\in K_{l,i}(t-1)} X_k \Big| \mathcal{F}_j\Big]\\
&\overset{(d)}{=} \boldsymbol{E}\Big[\prod_{k\in K_{i,j}(t)} X_k \Big| \mathcal{F}_j\Big].
\end{aligned}$$

Here, equality (a) follows from the induction hypothesis, (b) from the junction tree property and Lemma 8, (c) from the junction tree property and Lemma 7, and (d) from definition of $K_{i,j}(t)$.

Indeed, $K_{i,j}(t)$ above is the set of all the nodes whose "information" has reached the edge (i,j) by time t. Similarly, with $J_i(t) := \{i\}\bigcup_{j\in N_i} K_{j,i}(t)$, $J_i(t)$ is the collection of all the nodes whose "information" has reached a node i by time t. As in [5], we define a *message trellis* up to time t, which is an $M\times t$ directed graph, where for any $i,j\in\{1,\ldots,M\}$ and $n<t$, $i(n)$ is always connected to $i(n+1)$, and $i(n)$ is connected to $j(n+1)$ iff $(i,j)\in E_n$. It follows that we will have $J_i(t)=\{1,\ldots,M\}$ when there is a path from every initial node (i.e., at $t=0$) in the trellis to the node $i(t)$. Then, since the tree has finite diameter, *any* infinite schedule that activates all the edges infinitely many times has a finite subschedule, say of length t_0, such that $J_i(t_0)=\{1,\ldots,M\}$ for all i. At that time we have

$$\begin{aligned}
&\boldsymbol{E}\Big[X_i \prod_{j\in N_i} Y_{j,i}(t_0) \Big| \mathcal{F}_i\Big]\\
&\quad\overset{(a)}{=} \boldsymbol{E}\Big[X_i \prod_{j\in N_i} \boldsymbol{E}\Big[\prod_{k\in K_{j,i}(t_0)} X_k \Big| \mathcal{F}_i\Big]\Big| \mathcal{F}_i\Big]\\
&\quad\overset{(b)}{=} \boldsymbol{E}\Big[\boldsymbol{E}\Big[X_i \prod_{j\in N_i} \prod_{k\in K_{j,i}(t_0)} X_k \Big| \mathcal{F}_i\Big]\Big| \mathcal{F}_i\Big]\\
&\quad= \boldsymbol{E}\Big[X_i \prod_{j\in N_i} \prod_{k\in K_{j,i}(t_0)} X_k \Big| \mathcal{F}_i\Big]
\end{aligned}$$

$$\stackrel{(c)}{=} \boldsymbol{E}\Big[\prod_{k\in J_i(t_0)} X_k \Big| \mathcal{F}_i\Big]$$
$$= \boldsymbol{E}\Big[\prod_{k=1}^{M} X_k \Big| \mathcal{F}_i\Big]$$

where equality (a) follows from (23), (b) follows from the junction tree property and Lemma 8, and (c) follows from the definition of $J_i(t)$. This completes the proof of correctness of Algorithm 2. □

REFERENCES

[1] R. McEliece, D. MacKay, and J. Cheng, "Turbo decoding as an instance of pearl's 'belief propagation' algorithm," *IEEE J. Select. Areas Commun.*, vol. 16, pp. 140–152, Feb. 1998.

[2] D. MacKay and R. Neal, "Good codes based on very sparse matrices," in *Cryptography and Coding. 5th IMA Conf.*. Berlin, Germany: Sprilnger-Verlag, 1995, vol. 1025, pp. 100–111.

[3] R. Cowell, A. Dawid, S. Lauritzen, and D. Spiegelhalter, *Probabilistic Networks and Expert Systems*. NewYork, NY: Springer-Verlag, 1999.

[4] G. Shafer and P. Shenoy, "Probability propagation," *Ann. Math. Artificial Intell.*, vol. 2, pp. 327–352, 1990.

[5] S. Aji and R. McEliece, "The generalized distributive law," *IEEE Trans. Inform. Theory*, vol. 46, pp. 325–343, Mar. 2000.

[6] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Trans. Inform. Theory*, vol. IT-20, pp. 284–287, Mar. 1974.

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

[8] F. Baccelli, G. Cohen, G. Olsder, and J. Quadrat, *Synchronization and Linearity*. New York: Wiley, 1992.

[9] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, *Introduction to Algorithms*. New York: McGraw-Hill, 2001.

[10] P. Pakzad and V. Anantharam. (2002) Conditional Independence for Signed Measures. [Online]. Available: http://inst.eecs.berkeley.edu/payamp/signedci.pdf

[11] N. Zhang and D. Poole, "Exploiting causal independence in bayesian network inference," *J. Artificial Intell. Res.*, vol. 5, pp. 301–328, 1996.

[12] J. Walrand and P. Varaiya, *High-Performance Communication Networks*. San Francisco, CA: Morgan Kaufmann, 1996.

[13] R. McEliece, "On the bcjr trellis for linear block codes," *IEEE Trans. Inform. Theory*, vol. 42, pp. 1072–1092, July 1996.

[14] J. Cohen and U. Rothblum, "Nonnegative ranks, decompositions, and factorization of nonnegative matrices," *Linear Algebra Appl.*, vol. 190, pp. 149–168, 1993.