August 2, 1999
IMA Summer Program on Codes, Systems, and Graphical Models

(Joint work with Brendan Frey and Hans-Andera Loebliger.)

University of Toronto
Department of Electrical & Computer Engineering

Frank R. Kschischang

Algorithm
Factor Graphs and the Sum-Product
Such a factorization can be visualized using a factor graph. It decomposes a function into a product of "local" functions, in which the global function can be decomposed into simpler functions of many variables. They often derive computational efficiency by exploiting the way.

Many algorithms developed in computer science and engineering...
Factor graphs are bipartite; they represent the "is an argument
of" relation between variables and local functions.

Example:
A wide class of algorithms can be captured as specific instances of the sum-product algorithm. For computing marginal functions, the sum-product algorithm, when cycle-free, encodes an algorithm, but not only encodes the factorization of the global function. As we will see, the structure of a factor graph...
authors (and even earlier by Gallager) for decoding. Sum-product and min-sum algorithms were developed by these authors.

- Tanner/Wiberg/Loeliger/Kotter (TWLK) graphs (1995)
- Tanner graphs (1981)

For error-correction, codes:

Factor graphs are descendants of bipartite graph representations

Factor Graph Genealogy
At same ISIT, corridor discussions among „lim group“ leads to


tion (MP) problem.
non-probabilistic ones—solve the „marginalize product of func-
AII and MCEI/ice recognize that many algorithms—even some

AII
collection of random variables.

In this case of the joint probability mass function of a factor model also closely related to graphical models for probabilistic distributions, e.g., Bayesian networks.

- Markov random fields,
The Sum-Product Algorithm

Part I
observed channel output).

(E.g., the conditional joint pmf for codeword symbols given the admissible, given the observation of a related quantity. probability mass function for a collection of discrete random variables. As a working example, let the global function be the conditional the codomain, and even the notion of “product.” (It is possible to generalize both codomain of all functions is $\mathbb{R}$. For now, we can assume all functions are real-valued, i.e., the variable $x$ takes on values in a finite domain. Assume that a variable $x$ takes on values in a finite domain.)
summed over all values of its domain. Implicit is the assumption that each variable being summed, is

\[
\sum_{x_1, x_2, x_3} \sum_{y \in \{2\}} (x_1, x_2, x_3, y) = \sum_{x_1, x_2, x_3} \sum_{y \in \{2\}}
\]

Thus, if \( h \) is a function of three variables \( x_1, x_2, \) and \( x_3 \), the "not-sum over \( x_2 \)" is denoted those not being summed over. Instead of indicating the variables being summed over, indicate

\[
\text{not-sum}
\]

W. E introduce an eccentric summation notation: the \( \text{not-sum} \).
concept in probability theory. The term "marginal function" derives from the corresponding

\[
\{\{x\}\} \sim (u_1 x, \ldots, \{x\}_I) \hat{b} \overset{\sum}{\underbrace{\vdots}} = (\{x\}_I) \hat{b}
\]

Given \( \hat{b}(u_1, x, \ldots, \{I, x\}) \), define, for \( I = 1, \ldots, \),

In terms of the not-sum, it is easy to define marginal functions associated with a global function.
\[
\{1x\} \sim \\
\sum_{\cdot} (1x) \cdot (9x)
\]

and we wish to compute
\[
\cdot (x_5, x_3, x_2, x_1, x_4)(9x) \cdot (x_3, x_5, x_3, x_2, x_1, x_4)(9x) \cdot (x_3, x_5, x_3, x_2, x_1, x_4)(9x)
\]

Suppose, for example,
\[ \forall x \in \{x\} \sim \left( \forall x, x \in \{x\} \sim \left( \forall x, x \in \{x\} \sim \left( \exists x, f(x) \exists x \right) \right) \right) \forall x = (\exists x) \forall x \]
An expression tree for \( x \times (z + \bar{y}) \).

Expression trees:
\[
((\exists x \forall x) \lor f \{x\} \sim \exists) \times ((\forall x \exists x) \lor f \{x\} \sim \exists) \times (\exists x \forall x \exists x \lor f \times (\exists x) \lor f) \{\forall x \sim \exists \times (\forall x) \lor f
\]
$$\left( (\exists x \forall x \exists x \exists x \forall f \exists x) \times (\forall x \exists x \exists x \forall f \exists x) \times (\exists x \forall x \exists x \exists x \forall f \exists x \forall f \exists x) \right) = (\exists x)\exists x$$
Local transformation, converting a rooted cycle-free factor graph to an expression tree:

From Children

\[
\begin{array}{c}
\{x\} \\
\rightarrow \quad x
\end{array}
\]

To Parent

\[
\begin{array}{c}
f \times x \\
\leftarrow \quad f
\end{array}
\]
We compute sums and products. Call it the sum-product algorithm.

1. (Product rule) At a variable node $x_i^?$ take the product of expressions formed at the descendants of $x_i^?$. The algorithm is simple: for computing $g_i^? (x_i^?)$.

2. (Sum-product rule) At a function node $f$, take the product of $f$ with expressions formed at the descendants of $f$. Then perform the not-sum over the parent of $f$. We perform the not-sum over the parent of $f$. Therefore and therefore an root, the factor graph encodes an expression, and therefore an
The product of all messages sent to $x$. The marginal function $q^B(x)$ is the "final" message, namely:

- Start from the leaves and send messages "up" towards the root.
- Any particular processor can "fire" once it has received messages from its children.
- Scrtipions of local functions.
- Messages sent over channels are simply appropriate de-
- Edges are "channels" between processors.
- Vertices are "processors."

message-passing algorithm:

The sum-product algorithm can be described very naturally as a
Often we are interested in computing \( g_i(x_i) \) for all \( i \).

We can do this by \textit{overlaying} multiple copies of the single-\( i \) algorithm on a single graph.

Implicitly \textit{every} variable of the tree may at some point be regarded as root.
Again, $g_t(x^2)$ is the product of all messages received by $x^2$.

The process terminates when all vertices have received messages from all neighbors.

"Child" is, in turn, regarded as the parent. "Children" are the "children" of the one remaining neighbor is the "temporary parent." The immediate parent.

Again, the process starts at the least vertices, which "fires" but one of its neighbors.

A processor "fires" when it has received messages from all neighbors.

The sum-product algorithm in a tree:
\[(z) V \leftarrow z \cdot (f) V \leftarrow f \cdot (z, f, x) V_f \quad \{x\} \sim =
\]
\[(\exists x) V \leftarrow \exists x \cdot (\exists x, \cdots, \exists x, x) V_f \quad \{x\} \sim =: (x) x \leftarrow V \]

**Function to variable (the sum-product rule):**

\[(x) x \leftarrow \mathcal{C} \cdot (x) x \leftarrow \mathcal{B} \quad =
\]
\[
\{V\} \setminus \{x\} u \in \mathcal{D}
\]

\[(x) x \leftarrow \mathcal{B} \quad \prod =: (x) V \leftarrow x \]

**Variable to function (the product rule):**

![Diagram](image-url)
for any neighbor \( (\bar{v}x)^{i} x \rightarrow f \rightarrow l \times (\bar{v}x)^{i} x \leftarrow f \rightarrow l = (\bar{v}x)^{i} x \). \( \forall \)

**At termination,** \( f \) is always a function of \( x \), i.e., a single-argument function.

The message that passes (in either direction) over an edge in a finite number of steps.

2. **Messages have been passed,** i.e., in a finite number of steps, the algorithm terminates after all edges are finite with \( f \) edges, the algorithm terminates after exactly one message passes in each direction over any edge, i.e., the sum-product algorithm is a tree in a two-way algorithm.
(1, 1, 1) (1, 1, 0) (0, 1, 1) f (1, 1, 0) f (0, 1, 1) f (1, 1, 1) f (1, 0, 1) f (0, 1, 0) f (1, 0, 0) f (0, 0, 1) f (0, 0, 0) f

E.9. If f (x_1, x_2, x_3) = \Sigma (x_1' x_2' x_3') is a function of three binary variables,

essentially a marginal function f(x) of a function f(x_1, x_2, x_3) can be regarded as a "summary" of the mult.

We can generalize the notion of "sum" to "summary".
ratios with fixed \( x \).

minimum of \( H(x, y, z) \) taken over all argument contours.

- \( + \) is real-valued summation: \( \sum \) is the marginal function.

\[ e.g. \]

the operation by \( + \).

abelian semigroup (see the [Al] McEliece GDL paper). Designate

we can obtain a summary operator from a binary operation in an

In general a variety of summary operators are possible.
(See Ali/McEliece, "The Generalized Distributive Law" for more)

which makes the domain \( R \) of \( F \) a semiring:

\[
xx + xy = (x + y)x
\]

We assume the distributive law:
The message sent by a node on an edge is the product of the messages received on all other edges with the local function at the node, summarized for the variable associated with the edge.

**Sum-product algorithm (in words):**

In terms of general summary operators:
Part II
Codes and Probability Distributions
ability, resp. — of a physical system. These can often be viewed as models — set theoretical and prob-

characteristic functions for sets and probability distributions. By factor graphs, two particular classes of functions stand out: while many types of functions might potentially be represented
\[
\begin{cases}
    0 & \text{otherwise,} \\
    1 & \text{if } p
\end{cases}
\] = [p]

then

\text{Iverson's convention (from APL):} \text{ if } p \text{ is a Boolean proposition,}

\text{of valid behaviors or codewords. where } B \subseteq W \text{ is a subset of the possible configurations: the set}

\[
\begin{cases}
    0 & \text{otherwise,} \\
    1 & \exists m \in B
\end{cases}
\] = [B \in m] = (m)_B

Suppose the global function \( g \) is a binary indicator function, i.e., characteristic function:
\[
[0 = 7\alpha \oplus 5\alpha \oplus 4\alpha] \cdot [0 = 6\alpha \oplus 4\alpha \oplus 3\alpha] \cdot [0 = 4\alpha \oplus 2\alpha \oplus 1\alpha] = [C \in \Lambda]
\]

\[
\begin{align*}
0 &= 7\alpha \oplus 5\alpha \oplus 4\alpha \\
0 &= 6\alpha \oplus 4\alpha \oplus 3\alpha \\
0 &= 4\alpha \oplus 2\alpha \oplus 1\alpha
\end{align*}
\]

\[
\begin{bmatrix}
1 & 0 & 1 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 1 & 1
\end{bmatrix} = H
\]

For example, over \(F_2\), take
\[
\begin{bmatrix}
0 = \underline{H}^\alpha
\end{bmatrix}
\]

: \(\alpha\) = \(C\) then \(H\). Given \(H\), every linear block code can be specified by a parity check matrix \(H\). For example, every linear block code over \(\mathbb{F}_2\) can be expressed as a series of local checks.
\[ 0 = 7 \oplus 5 \oplus 4 \oplus 3 \cdot 0 = 6 \oplus 4 \oplus 3 \oplus \varepsilon \cdot 0 = 4 \oplus 2 \oplus 1 \oplus \varepsilon = [C \in \Lambda] \]
Shokrollahi, Urbanke, 1999). Most powerful long, yet decodable, codes known (Richardson,
Recent irregular enhancements of LDPC codes give rise to the

A (3,4) LDPC code.

Low-Density Parity-Check Codes (R. G. Gallager, 1962)
\[ [\vec{L} \ni (\vec{s}, \vec{\alpha}, \vec{\tau}^{-\vec{s}})] \prod_{\vec{L}} = (L_s, \cdots, 0_s, L_\alpha, \cdots, \tau_\alpha)f \]

set of edges in the \( i \) trellis section

Factor Graph for a Trellis
\[(\ell)nA + (\ell)x\circ = (\ell)\hat{h}][(\ell)nB + (\ell)x\forall = (1 + \ell)x] = (((1 + \ell)x(\ell)\hat{h}, (\ell)n(\ell)x)\ell)\ell\ 
\times \text{is} \{1, 0\} \leftarrow uF \times uF \times \omega F : ((1 + \ell)x(\ell)\hat{h}, (\ell)n(\ell)x)\ell\ell
\]

The time-\(\ell\) check function

Classical linear state-space model:
Factor Graph for a Tail-Biting Trellis
Factor graph for a turbo code

simply augmented with the channel likelihood functions. For the function indicating code membership is

\[ (x_i, \ldots, x_1) f \prod_{u \in E} (x_i) f \prod_{E \in E} ^{\exists} \prod_{F \in F} = (u x_i, \ldots, x_1, x_1) f \]

then proportional to

\[ \{N x_i, \ldots, x_1\} f \prod_{E \in E} ^{\exists} \prod_{F \in F} = (N \bar{y}, \ldots, \bar{y}) \]

and observe \((N \bar{y}, \ldots, \bar{y})\). Suppose we transmit \((N x_i, \ldots, x_1) f \prod_{E \in E} ^{\exists} \prod_{F \in F} \) over a memoryless channel,

A Posteriori Probabilities
Memoryless channel $\leftrightarrow$ augment code's factor graph with "Dongle".

(Up to scale factor)

(a posteriori distribution)

(Uniform)

(a priori distribution)
(3x|5x) ∩ \{ 3x|4x \} ∪ \{ 2x|3x \} \cup \{ 2x|5x \} \cup \{ 1x|3x \} \cup \{ 4x|4x \} = (5x, \cdots, 1x) f

Bayesian Networks
Belief propagation as an instance of the sum-product algorithm.
The forward/backward BJR algorithm.
\( (\exists x) \land (\exists s) g(\exists s) \forall [\exists L \ni (\exists s \land \exists x \land \exists s)] \subseteq \exists \exists \subseteq \exists \quad = \quad (\exists n) g \)

\( (\exists x) \land (\exists s) g[\exists L \ni (\exists s \land \exists x \land \exists s)] \subseteq \exists \exists \subseteq \exists \quad = \quad (\exists s) g' \)

\( (\exists x) \land (\exists s) \forall [\exists L \ni (\exists s \land \exists x \land \exists s)] \subseteq \exists \exists \subseteq \exists \quad = \quad (\exists s) \forall \)
Can express as sums over trellis edges:

\[
\sum_{\omega \in \mathcal{F}} (\omega)^2 \sum_{\omega \in \mathcal{F}} (1 + \omega)^2 \sum_{\omega \in \mathcal{F}} (\omega)(1) = (\omega)^2 \sum_{\omega \in \mathcal{F}} (\omega)(1)
\]
Filtertilng
Gaussian variables + summarization by integration → Kalman

(2-way version gives the same result.)
With traceback → Viterbi algorithm. (1-way version gives the
in the min-sum semiring → min-sum algorithm. One-way version
What happens when the factor graph is not cycle-free? For example, suppose we have

Coping with Cycles:
Note that the "not-sums" now involve more than one variable.

\[
\begin{align*}
(\forall x, \exists x, \exists x) & \cap \exists (\forall x, \exists x) \supset \exists (\forall x, \exists x) \\
& = \\
(\forall x, \exists x) & \cap \exists (\exists x, \exists x) \supset \exists (\forall x, \exists x) \\
& = \\
(\forall x, \exists x) & \cap \exists (\exists x, \exists x) \supset \exists (\forall x, \exists x) \\
& = \\
(\forall x, \exists x, \exists x) & \cap \exists (\forall x, \exists x) \\
& =: (\exists x) \cap \exists
\end{align*}
\]
spanning-tree in which $x^i$ is involved.

A variable $x^i$ must be "carried" over all regions of the sub-
junction tree (see Aii and McEliece).

Equivalent to forming a spanning tree by cutting the loop, with
"carried" around the cycle.

\[ x^i, x^j, x^k, x^l \]

\[ a, b, c, d \]
\[
\begin{align*}
0 h_0 x M_{0 h} x (t - 1) x_0 x (t - 1) x_1 x (t - 1) x_2 x (t - 1) & \\
\times [0 x + 1 x 2 + 2 x 4] f &= \\
(0 h + 1 h x 2 + 2 h y) (0 x + 1 x 2 + 2 x 4) M_{0 x + 1 x 2 + 2 x 4} f &= \\
(0 h + 1 h x 2 + 2 h y, y) f (x, t) & \leftarrow g (x, t, x_1, x_2, y_0, y_1, y_2)
\end{align*}
\]

**DFT Kernel Function**

Write \( u = 4 x 2 + 2 x 1 + 0 x, y = 4 x 2 + 2 x 1 + 0 x \), \( y = 4 x 2 + 2 x 1 + 0 x \), and \( x_0, x_1, x_2, y_0, y_1, y_2 \in \{0, 1\} \).

Take \( N = 8 \) and \( u = 2 x 1 + 4 x 2, y = 2 x 1 + 4 x 2 \), \( y = 2 x 1 + 4 x 2 \), \( Y = 4 x 2 + 2 x 1 + 0 x \).

Where

\[
\begin{align*}
\sum_{y=0}^{N} M_{y u} f_{N} & = [y] H
\end{align*}
\]

**The N-Point Discrete Fourier Transform:**

\[
\begin{align*}
The \text{ DFT (We follow Ali and McElice, 1997)}
\end{align*}
\]
An FFT

A Spanning Tree

Factor Graph

Factor Graph
codes and turbo codes.

This works very well in practice (witness low-density parity-check
APPS are quantized in the end.
ization is computationally infeasible, and since the approximate
This is the approach favored in decoding, since exact marginal-
Ignore them and hope for the best!

Copied with Cycles (Part II):
Decoding a Turbo Code:
ciphers. (analogs of ciphers: Loeliger)
- Use the structure of the graph to build highly parallel de-

state-space representations (Forney graphs).
- Generalized partition and can be "absorbed" as edges. Let variables all have degree two; they perform no com-

analysis (Richardson, Urbanke).
- Use "local" cycle-free neighborhood to perform approximate

(some) connections to later talks.
The sum-product algorithm connects a variety of well-known algorithms in a common framework. Factor graphs provide a natural description of the factorization of a global function into a product of local functions.