Belief propagation and Generalized Distributive Laws
Beyond posterior probabilities
©Marina Meilă and Carl de Marcken
{mmp,cgdemarc}@stat.washington.edu

We could summarize the class so far as:

- Conditional independence as expressed by a graph $G$ lets us factor $P(X_1 \ldots X_n)$ into product of potentials $\prod_i \Phi_{C_i}$ where $C_i$ are cliques of $G$

\[
R - S - T \iff R \perp T | S \iff P_{RST} = \Phi_{RS} \Phi_{ST}
\]

- Distributive law lets us build efficient algorithms for marginalizing joint probabilities.

\[
Z = \sum_{rst} P_{RST}(rst) = \sum_{rst} \Phi_{RS}(rs) \Phi_{ST}(st) = \sum_s \left( \sum_r \Phi_{RS}(rs) \right) \left( \sum_t \Phi_{ST}(st) \right)
\]

- Solve the inference problem $P(X_A | X_b = x_b)$ by entering evidence $x_b$ into potential functions and marginalizing (eliminating) all variables $X - \neq X_A$.

It’s interesting at this point to consider two questions:

- Are there ways of generalizing the methods we’ve developed to solve problems outside the domain of probability?

- Are there probabilistic operations beyond inference can we perform efficiently on graphical models?
1 Functions over configurations

Many optimization and counting (combinatoric) problems can be cast in the following form:

- There is a configuration space $X$ over a finite set of variables, $X = X_1 \times \ldots \times X_n$.
- Constraints between variables are expressed using logical or arithmetic functions over subsets of the variables, for example $X_1 \lor X_2$ or $X_1 + X_2 + X_3 \geq 2$.
- One or more value functions $f$ are defined over $X$, that decompose into the sum or product of functions over subsets of the variables; we will use $\otimes$ to denote the value composition operator: $f(x) = \otimes_c f_c(x_c)$.
- One or more aggregation functions (such as $+$, min, max, list) combine values $f(x)$ across the space $X$ (or a subset of variables) to create a summary $Z$. We will use $\oplus$ to denote the aggregation function: $Z = \oplus_x f(x)$.

Probability calculations over graphical models can be cast in this framework, where the value function is $P_X = \prod C \Phi (\text{thus, } \otimes = \times)$ and the aggregation function is $\oplus = +$ (or $\oplus = \max$ if the goal is to find the maximum a posteriori (MAP) probability). Hence we can hope to generalize algorithms for manipulating graphical models to solve problem from these other fields, typically by introducing different operators $\otimes$ and $\oplus$ that have the same algebraic properties as $\times$ and $+$.

1.1 Algebra

In our derivations of the FB, VE, JT and SP algorithm, we relied on the fact that $\times$ and $+$ over $\mathbb{R}$ form a semi-ring. A semi-ring is a set $R$ and two operators $\otimes$ and $\oplus$ such that:

- $(R, \oplus)$ is a commutative monoid with identity 0
  - $a \oplus (b \oplus c) = (a \oplus b) \oplus c$
  - $a + b = b + a$
  - $0 + a = a$
• \((R, \otimes)\) is a monoid with identity 1
  
  \[- a \otimes (b \otimes c) = (a \otimes b) \otimes c \]
  
  \[- 1 \times a = a \times 1 = a \]

• \(\otimes\) distributes over \(\oplus\)
  
  \[- a \otimes (b \oplus c) = (a \otimes b) + (a \otimes c) \]

The distributive property is what lets us write

\[
\sum_{rst} \Phi_{RS}(rs)\Phi_{ST}(st) = \sum_s \left( \sum_r \Phi_{RS}(rs) \right) \left( \sum_t \Phi_{ST}(st) \right) .
\]

In some algorithms we also relied on \(\times\) being commutative (for example, in the VE algorithm that allows us to eliminate in any order); a commutative semi-ring adds the axiom

\[- a \otimes b = b \otimes a .\]

Two other very useful aggregation functions that form a commutative semi-ring with \(\otimes = \times\) (and \(\otimes = +\)), so long as values are non-negative, are \(\oplus = \text{min}\) and \(\oplus = \text{max}\). The simple substitution of max for + in all algorithms lets us find the probability of the most probable configuration (as per the Viterbi algorithm).

In many problems it is more convenient to assume the value composition operator is + rather than \(\times\), for example when dealing with costs or energies rather than probabilities (or when manipulating log probabilities to avoid underflow issues). Again, min and max distribute over +.

Frequently one wishes to consider multiple values at once; clearly using elementwise operations semi-rings over values can be extended to semi-rings over vectors of values.

## 2 Total enumeration

Suppose we wish to enumerate the complete set of configurations, each annotated with its value as defined by \(P(x) = \prod \Phi_c(x_c)\) (or equivalently, \(f(x) = \sum f_c(x_c)\)). Consider the
following operators and potential functions defined over sets of pairs \((v; c)\) where \(v\) is a (partial) value and \(c\) is a (partial) configuration:

\[
\begin{align*}
\Psi_{X_i}(x_i) &= \{\langle 1; \{X_i = x_i\}\rangle\} \\
\Psi_{\Phi_c}(x_c) &= \{\langle \Phi_c(x_c); \{\}\rangle\} \\
0 &= \{\} \\
1 &= \{\langle 1; \{\}\rangle\} \\
A \otimes B &= \{\langle v_a \cdot v_b; c_a \cup c_b\rangle : \langle v_a; c_a\rangle \in A, \langle v_b; c_b\rangle \in B\} \\
A \oplus B &= A \cup B
\end{align*}
\]

\(Z = \oplus(\otimes \Psi)\) is the complete set of valued configurations. Solving using VE for \(A - \Phi_{AB} - B\), one starts from

\[
\begin{array}{c|c|c|c|c|c|c|c|c}
A & \Psi_A & B & \Psi_B & A & B & \Psi_{AB} \\
0 & \{\langle 1; \{A = 0\}\rangle\} & 0 & \{\langle 1; \{B = 0\}\rangle\} & 0 & 0 & \{\langle \Phi_{AB}(00); \{\}\rangle\} \\
1 & \{\langle 1; \{A = 1\}\rangle\} & 1 & \{\langle 1; \{B = 1\}\rangle\} & 1 & 0 & \{\langle \Phi_{AB}(10); \{\}\rangle\} \\
& & & & 1 & 1 & \{\langle \Phi_{AB}(11); \{\}\rangle\} \\
\end{array}
\]

Elimination of \(A\) gives

\[
\Psi_B'(b) = \oplus_a (\Psi_A(a) \otimes \Psi_{AB}(a, b)) = \begin{cases} 
0 & \{\langle \Phi_{AB}(00); \{A = 0\}\rangle\} \\
& \{\langle \Phi_{AB}(10); \{A = 1\}\rangle\} \\
1 & \{\langle \Phi_{AB}(01); \{A = 0\}\rangle\} \\
& \{\langle \Phi_{AB}(11); \{A = 1\}\rangle\}
\end{cases}
\]

and elimination of \(B\) gives
\[ Z = \bigoplus_b (\Psi_B(b) \otimes \Psi'_B(b)) = \begin{cases} \langle \Phi_{AB}(00); \{A = 0B = 0}\rangle \\ \langle \Phi_{AB}(01); \{A = 0B = 1}\rangle \\ \langle \Phi_{AB}(10); \{A = 1B = 0}\rangle \\ \langle \Phi_{AB}(11); \{A = 1B = 1}\rangle \end{cases} \]

though of course the output size is exponential size and takes exponential time to compute.

### 3 Example: paths in directed graphs

Consider problems related to paths in a directed, acyclic graph \( G = (V, E) \) with initial vertex \( S \) and terminal vertex \( T \). The space of valid paths can be expressed by introducing integer-valued variables \( X_v \) for every vertex \( v \in V \) and binary variables \( X_{uv} \) for every edge \((u, v) \in E\), and constraints

\[
\begin{align*}
X_S &= 1 \quad \text{(initial vertex is in path)} \\
X_T &= 1 \quad \text{(terminal vertex is in path)} \\
\forall v \neq S \quad X_v &= \sum_{(u,v) \in E} X_{uv} \quad \text{(flow into} \ v \ \text{is conserved)} \\
\forall v \neq T \quad X_v &= \sum_{(v,w) \in E} X_{vw} \quad \text{(flow out of} \ v \ \text{is conserved)}
\end{align*}
\]

The variables \( X_v \) must be further constrained to be binary \( X_v \leq 1 \) if we only want to consider simple paths, that visit a node no more than once.

Let there be two additive functions over the edges of a path, that can therefore be written in terms of the edge variables \( f(X) = \sum_{e \in E} X_e f_e \), \( g(X) = \sum_{e \in E} X_e g_e \) where \( f_e \) and \( g_e \) are per-edge constants. If the graph represents a flight network, then \( f_e \) and \( g_e \) might be the cost and duration of flight \( e \). Finally let there be a probability distribution \( h \) defined by potentials over the edge and vertex variables.

All of the following problems (and many more) can be solved through suitable reformulations of graphical model inference algorithms:

- count the paths in the graph
• find the shortest path in the graph as measured by $f$
• enumerate $k$ paths in order of $f$
• calculate $E_h[f], E_h[f^2], E_h[f^g]$ and more general moments $E_h[f^kg^l]$
• sample a random path according to $h$
• find all possible values of $f, g$ with their path counts
• for each edge $e$ or vertex $v$ compute the minimal $f$ for any path involving $e$ or $v$

3.0.1 Counting paths

If the flow constraints are expressed using 0-1 potentials

$$
\Phi_S = \delta(X_S, 1) \\
\Phi_T = \delta(X_T, 1) \\
\Phi_{\text{in}}^v = \delta(X_v, \sum_{(u,v) \in E} X_{uv}) \\
\Phi_{\text{out}}^v = \delta(X_v, \sum_{(v,w) \in E} X_{vw}) \\
\Phi^v = \delta(X_v \leq 1)\text{simple paths only}
$$

then $P(X) = \prod \Phi(X)$ is an unnormalized probability distribution over $X$ with value 1 for valid paths and 0 otherwise. Thus the number of paths is given by the normalization constant $Z = \sum P(X)$, easily computed by variable elimination.

3.0.2 Shortest path

To compute the additive $f$, take $\otimes = +$ and create potentials $\Phi_e(X_e) = X_e f_e$. To aggregate to the shortest path, $\oplus = \min$. The change of the value composition operator from $\times$ to $+$ causes problems for our representation of the flow constraints, but these are solved by taking logs of the flow potentials, replacing 0 with the cost $\infty$ and 1 with the cost 0. (In other words, the shortest path is the maximum log-likelihood path.)
4 Moments

Consider calculating the expected value of a function $f$ over a distribution defined by
a graphical model: $E[f] = \sum_x P(x) f(x)$. If $f(x)$ is arbitrary there is no more efficient
way to compute the expectation than by summing over the entire domain. However if
$f(x) = \sum_i f_i(x_i)$ then by linearity of expectations

$$E[f] = \sum_i E[f_i] = \sum_i \sum_{x_i} P(x_i) f_i(x_i)$$

and the expectation is easily computed by using the FB, JT or SP algorithms to compute
the marginal posterior probability $P(x_i)$ of each variable value and then $\sum_{x_i} P(x_i) f_i(x_i)$. This extends to functions that decompose over cliques $f(x) = \sum_c f_c(x_c)$, since each of the FB, JT and SP algorithms can equally efficiently compute the posterior probabilities of clique configurations.

It is natural question to ask if expectations can be computed as an integral part of prob-
ability calculations rather than as a post-processing step. Is there an easy way to extend
the FB, VE, JT or SP algorithms to return $E[f]$ directly?

Consider calculating $E[f = f_R + f_T]$ over the distribution $P_{RST} = \Phi_{RS}\Phi_{ST}/Z$. Then

$$E[f] = \frac{1}{Z} \sum_{rs} (f_R(r) + f_T(t))\Phi(rs)\Phi(st)$$
$$= \frac{1}{Z} \sum_s (\sum_r f_R(r)\Phi(rs)\sum_t \Phi(st) + \sum_t f_T(t)\Phi(st)\sum_r \Phi(rs))$$
$$= \frac{1}{Z} \sum_s (Z_{ST}(s)\sum_r f_R(r)\Phi(rs) + Z_{RS}(s)\sum_t f_T(t)\Phi(st))$$
$$= \frac{\sum_s (Z_{ST}(s)E_{RS}(s) + Z_{RS}(s)E_{ST}(s))}{\sum_s Z_{SR}(s)Z_{ST}(s)}$$

where

$$Z_{RS}(s) = \sum_r \Phi(rs) \quad Z_{ST}(s) = \sum_t \Phi(st)$$
$$E_{RS}(s) = \sum_r f_R(r)\Phi(rs) \quad E_{ST}(s) = \sum_t f_T(t)\Phi(st).$$
Picking this apart, the table $\langle Z_{RS}(s); E_{RS}(s) \rangle$ is a sufficient statistic after the elimination of $R$ (and similarly $\langle Z_{ST}(s); E_{ST}(s) \rangle$ after the elimination of $T$). This suggests replacing the numeric values $Z$ in the summary tables of our algorithms with pairs $\langle Z; E \rangle$, and extending the sum and product operators:

$$
\Psi_c(x_c) = \langle \Phi_c(x_c); \Phi_c(x_c)f(x_c) \rangle \\
\langle Z_1; E_1 \rangle \oplus \langle Z_2; E_2 \rangle = \langle Z_1 + Z_2; E_1 + E_2 \rangle \\
\langle Z_1; E_1 \rangle \otimes \langle Z_2; E_2 \rangle = \langle Z_1Z_2; E_1Z_2 + E_2Z_1 \rangle.
$$

It is easy to verify that $E[f] = \frac{E}{Z}$ where

$$
\langle Z; E \rangle = \oplus_s (\oplus_r \Psi_{RS}) \otimes (\oplus_t \Psi_{ST}).
$$

More generally, to compute $E_P[f = \sum_i f_i]$ where $P = \prod_j \Phi_j$, define extended potentials $\Psi$ as follows:

$$
\Psi_i = \langle 1; f_i \rangle \quad \text{function potentials} \\
\Psi_j = \langle \Phi_j; 0 \rangle \quad \text{probability potentials}
$$

and then calculate $\langle Z; E \rangle$ by using any of the algorithms we have studied, defined using the $\oplus$ and $\otimes$ operators in place of $+$ and $\times$.

\footnote{For VE, $\langle Z; E \rangle$ is the final result after eliminating all variables; for the FB algorithm, $\langle Z; E \rangle = \oplus_{q_i}(\alpha_i(q_i) \otimes \beta_i(q_i))$ (for any position $i$); for the JT algorithm, $\langle Z; E \rangle = \oplus_s \Phi_S(s)$ (for any separator $S$). The junction-tree algorithm divides values during absorption, so one must define $\otimes$. Verify that

$$
\otimes ((Z_n; E_n), (Z_d; E_d)) = \left\langle \frac{Z_n}{Z_d}; \frac{E_n}{Z_d} - \frac{Z_nE_d}{Z_d^2} \right\rangle
$$

ensures $A = B \otimes (A \otimes B)$ when $Z_B \neq 0$.}
4.1 Arbitrary moments and generating functions

What about higher moments, such as $E[f^2]$, useful for calculating $\sigma_f^2 = E[f^2] - E[f]^2$? Can one calculate $E[f^k]$ efficiently? (You might at this point want to try for a moment to come up with an algorithm for computing $E[f^2]$.)

While it is fairly straightforward to derive extensions of the $\oplus$ and $\otimes$ operators to compute $E[f^k]$ directly from the expansion of $f^k = (\sum_i f_i)^k$, a cleaner and more general derivation comes from considering the moment generating function for $f$:

$$F(\lambda) = E[e^{\lambda f}] = E[1 + \lambda f + \frac{\lambda^2 f^2}{2!} + \ldots].$$

Notice that

$$E[f^k] = \frac{\partial^k F(\lambda)}{\partial \lambda^k}(0).$$

To compute $E[f^k]$ it is sufficient to maintain only the first $k + 1$ terms of $F(\lambda)$, a $k$-th degree polynomial in $\lambda$ defined by $k + 1$ coefficients. Define

$$F^k(\lambda) = E[1 + \lambda f + \frac{\lambda^2 f^2}{2!} + \ldots + \frac{\lambda^k f^k}{k!}]$$

$$= \sum_x P(x)(1 + \lambda f(x) + \frac{\lambda^2 f(x)^2}{2!} + \ldots + \frac{\lambda^k f(x)^k}{k!})$$

$$= \sum_{i=0}^k \lambda^k F_i^k$$

represented by the coefficients.
\[ \langle F^k_0; F^k_1; \ldots; F^k_k \rangle = \langle \sum_x P(x); \sum_x P(x)f(x); \ldots; \sum_x P(x)\frac{f^k(x)}{k!} \rangle. \]

Consider a single clique \( c \) with probability potential \( \Phi_c \) and function term \( f_c \). Let

\[
\Psi_c(x_c) = \Phi_c(x_c)\langle 1; f(x); \ldots; \frac{f^k(x)}{k!} \rangle
= \langle \Phi_c(x_c)\Phi_c(x_c)f(x); \ldots; \Phi_c(x_c)\frac{f^k(x)}{k!} \rangle.
\]

and define the normal addition and multiplication operators for formal power series (polynomials):

\[
\langle Q_0; \ldots; Q_k \rangle \oplus \langle R_0; \ldots; R_k \rangle = \langle Q_0 + R_0; \ldots; Q_k + R_k \rangle
\]
\[
\langle Q_1; \ldots; Q_k \rangle \otimes \langle R_1; \ldots; R_k \rangle = \langle S_1; \ldots; S_k \rangle
\]

where \( S_i = \sum_{j=0}^{i} Q_jR_{i-j} \). Notice that terms of \( S \) higher than \( k \) have been dropped, since we only need to maintain to \( k \) to compute \( \text{E}[f^k] \). After computation of \( \langle F^k_1; \ldots; F^k_k \rangle \) (for example, through total VE), we have \( \text{E}[f^i] = i!F_i^k \) (or \( \text{E}[f^i] = \frac{i!F_i^k}{F_0^k} \) if the probability distribution is not normalized).

It is easy to extend this method to compute covariances between functions, for example \( \text{E}[f^kg^l] \) where both \( f \) and \( g \) are decomposable.

### 4.2 Example

Consider the following transition diagram over binary variables \( X_1, X_2 \) and \( X_3 \):
as defined by the factor graph $X_1 - \Phi_{12} - X_2 - \Phi_{23} - X_3$ with potentials

$$\Phi_1^p = \frac{1}{2}$$

<table>
<thead>
<tr>
<th>$X_1$</th>
<th>$X_2$</th>
<th>$\Phi_{12}$</th>
<th>$X_2$</th>
<th>$X_3$</th>
<th>$\Phi_{23}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2/3</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1/3</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1/3</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2/3</td>
</tr>
</tbody>
</table>

Problem: For $f = X_1 + X_2 + X_3$, compute $E[f]$ and $\sigma_f^2$.

Solution: First, create potentials over formal power series up to order 2 as follows (where $\Psi^p$ are probability potentials and $\Psi^f$ are function potentials):

where
\[
\Psi_i^f(x_i) = \left\langle 1; f(x_i); \frac{f(x_i)^2}{2!} \right\rangle
\]

\[
\Psi^f_i(x_i) = \begin{cases} 
\left\langle 1; 1; \frac{1}{2} \right\rangle & \text{if } x_i = 1 \\
\left\langle 1; 0; 0 \right\rangle & \text{if } x_i = 0
\end{cases}
\]

\[
\Psi^p_{ij}(x_i x_j) = \langle \Phi_{ij}(x_i x_j); 0; 0 \rangle
\]

Solving for \( F^2 \) using variable elimination. First eliminate \( X_1 \) by constructing \( \Psi_2(x_2) = \sum_{x_1} \Psi^f_1(x_1) \Psi^p_{12}(x_1 x_2) \Psi^f_1(x_1) \):

<table>
<thead>
<tr>
<th>( x_2 )</th>
<th>( \Psi^f_1(0) )</th>
<th>( \Psi^p_{12}(0, x_2) \Psi^f_1(0) ) + ( \Psi^f_1(1) )</th>
<th>( \Psi^p_{12}(1, x_2) \Psi^f_1(1) ) = ( \Psi_2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>\left\langle 1; 0; 0 \right\rangle</td>
<td>\left\langle \frac{1}{2}; 0; 0 \right\rangle + \left\langle 1; 1; \frac{1}{2} \right\rangle</td>
<td>\left\langle 0; 0; 0 \right\rangle = \left\langle \frac{1}{4}; 0; 0 \right\rangle</td>
</tr>
<tr>
<td>1</td>
<td>\left\langle 1; 0; 0 \right\rangle</td>
<td>\left\langle 0; 0; 0 \right\rangle + \left\langle 1; 1; \frac{1}{2} \right\rangle</td>
<td>\left\langle \frac{1}{2}; 0; 0 \right\rangle = \left\langle \frac{1}{2}; \frac{1}{2}; \frac{1}{4} \right\rangle</td>
</tr>
</tbody>
</table>

Then eliminate \( X_2 \) by constructing \( \Psi_3(x_3) = \sum_{x_2} \Psi^f_2(x_2) \Psi^p_{23}(x_2 x_3) \Psi_2(x_2) \):

<table>
<thead>
<tr>
<th>( x_3 )</th>
<th>( \Psi^f_2(0) )</th>
<th>( \Psi^p_{23}(0, x_3) \Psi_2(0) ) + ( \Psi^f_2(1) )</th>
<th>( \Psi^p_{23}(1, x_3) \Psi_2(1) ) = ( \Psi_3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>\left\langle 1; 0; 0 \right\rangle</td>
<td>\left\langle \frac{1}{3}; 0; 0 \right\rangle + \left\langle 1; 1; \frac{1}{2} \right\rangle</td>
<td>\left\langle \frac{1}{2}; 0; 0 \right\rangle = \left\langle \frac{1}{2}; \frac{1}{2}; \frac{1}{2} \right\rangle</td>
</tr>
<tr>
<td>1</td>
<td>\left\langle 1; 0; 0 \right\rangle</td>
<td>\left\langle \frac{1}{3}; 0; 0 \right\rangle + \left\langle 1; 1; \frac{1}{2} \right\rangle</td>
<td>\left\langle \frac{1}{2}; 0; 0 \right\rangle = \left\langle \frac{1}{2}; \frac{1}{2}; \frac{1}{2} \right\rangle</td>
</tr>
</tbody>
</table>

Then eliminate \( X_3 \) to solve \( F^2 = \sum_{x_3} \Psi^f_3(x_2) \Psi_3(x_3) \):

\[
\begin{array}{cccc}
\Psi^f_3(0) & \Psi_3(0) & + & \Psi^f_3(1) & \Psi_3(1) = F^2 \\
\left\langle 1; 0; 0 \right\rangle & \left\langle \frac{1}{2}; \frac{1}{3}; \frac{1}{3} \right\rangle & + & \left\langle 1; 1; \frac{1}{2} \right\rangle & \left\langle \frac{1}{2}; \frac{1}{3}; \frac{1}{3} \right\rangle = \left\langle 1; \frac{3}{2}; \frac{9}{12} \right\rangle \\
\end{array}
\]

Finally
\[
\begin{align*}
Z &= F_0^2 = 1 \\
E[f] &= \frac{1!F_1^2}{Z} = \frac{3}{2} \\
E[f^2] &= \frac{2!F_2^2}{Z} = \frac{23}{6} \\
\sigma_f^2 &= E[f^2] - E[f]^2 = \frac{23}{6} - \frac{3^2}{2^2} = \frac{5}{6}
\end{align*}
\]

5 Ordered enumeration

Suppose the goal is to find the \( k \) highest valued configurations. It should be clear that it is sufficient to modify the total enumeration algorithm to maintain only the \( k \) highest valued configurations in every set, but if \( k \) is large the burden of maintaining such sets is substantial and one can do better.

Instead, construct a junction tree and apply \texttt{CollectEvidence} from an arbitrarily chosen root clique \( C_R \), where the Absorb\((C \rightarrow C')\) procedure is defined with \( \oplus = \max \):

\[
\begin{align*}
\Phi_{S}^{\text{new}} &= \max_{C/S} \Phi_C \\
\Phi_{C'}^{\text{new}} &= \Phi_{C'} \frac{\Phi_{S}^{\text{new}}}{\Phi_{S}}
\end{align*}
\]

If for any clique or separator \( Y \) we define \( \Psi_Y \) to be the product of the pre-absorption potentials for the subtree headed by \( Y \), and use desc\((Y)\) to refer to the variables (other than \( Y \)) in this subtree, then \texttt{CollectEvidence} ensures
\[ \Phi_{C}^{\text{new}} = \max_{\text{desc}(C)} \Psi_{C} \]
\[ \Phi_{S}^{\text{new}} = \max_{\text{desc}(S)} \Psi_{S} \]

and of course the maximum probability configuration is given by

\[ p^* = \max \prod_{C} \Phi_{C}^{\text{new}} = \max_{r \in R} \Phi_{R}^{\text{new}}. \]

This leads to the following algorithm for enumerating the top \( k \) configurations, based on a priority queue of partial configurations where each queue element is a pair \( \langle \pi; \rho \rangle \), where \( \pi \) is the maximum (unnormalized) probability for the partial configuration \( \rho \) (maximized over the variables not in \( \rho \)):

1. construct a junction tree and choose a root clique \( R \)
2. apply \texttt{CollectEvidence} from \( R \) using \( \oplus = \max \)
3. create a priority queue \( Q \) over \( \langle \pi; \rho \rangle \) ordered by decreasing \( \pi \)
4. for each \( r \in R \), insert \( \langle \Phi_{R}^{\text{new}}(r); \{R = r\} \rangle \) into \( Q \)
5. while \( Q \) is nonempty and fewer than \( k \) configurations have been emitted
   (a) dequeue top element \( \langle \pi; \rho \rangle \) from \( Q \),
   (b) if \( \rho \) is complete
      i. emit \( \rho \)
   (c) otherwise
      i. let \( C \) be some clique separated from \( \rho \) by \( S \) (where \( s = \rho(S) \))
      ii. for every \( x \in C \) consistent with \( s \), enqueue \( \langle \Phi_{C}^{\text{new}}(c); \Phi_{S}^{\text{new}}(s); \rho \cup \{C = c\} \rangle \)

14
This algorithm iteratively expands configurations. The key is that probabilities can be updated after assignment \( c \) to a clique \( C \) by

\[
\pi \leftarrow \pi \frac{\Phi_{C}^{\text{new}}(c)}{\Phi_{S}^{\text{new}}(s)}
\]

where \( s \) is the assignment to separator \( S \) in \( \rho \). This can be understood as follows: \( \pi \) includes a term \( \Phi_{S}^{\text{new}} = \max_{C/S} \Phi_{C}^{\text{new}} \) reflecting the best possible assignment to the variables \( T = C - S \) not in \( \rho \); if instead a suboptimal choice \( c \in C \) is made, then \( \pi \) must be updated to reflect that by replacing \( \Phi_{S}^{\text{new}}(s) \) with \( \Phi_{C}^{\text{new}}(c) \).

### 5.1 Example

Consider the following example: \( X_1 - \Phi_{12} - X_2 - \Phi_{23} - X_3 \) where the potentials are given by:

\[
\begin{align*}
X_1 &= 1 \quad 1 \quad X_2 &= 1 \quad 1 \quad X_3 &= 1 \\
X_1 &= 0 \quad 1 \quad X_2 &= 0 \quad 5 \quad X_3 &= 0
\end{align*}
\]

Running \texttt{CollectEvidence} on the junction tree \( C_{12} - S_2 - C_{23} \) with root \( C_{12} \), we have
\[ \Phi_{2\text{new}} = \max \Phi_{23} = \begin{array}{c|c}
X_2 & \Phi_{2\text{new}}^2 \\
0 & 5 \\
1 & 7 \\
\end{array} \]

\[ \Phi_{12}^{\text{new}} = \Phi_{12} \frac{\Phi_{2\text{new}}^2}{1} = \begin{array}{c|c|c|c|c}
X_1 & X_2 & \Phi_{12}^{\text{new}} \\
0 & 0 & 1 \cdot 5 = 5 \\
0 & 1 & 4 \cdot 7 = 28 \\
1 & 0 & 3 \cdot 5 = 15 \\
1 & 1 & 1 \cdot 7 = 7 \\
\end{array} \]

From \( \Phi_{12}^{\text{new}} \), we initialize

\[ Q = \langle 28; \{ X_{12} = 011 \} \rangle \]
\[ \langle 15; \{ X_{12} = 10 \} \rangle \]
\[ \langle 7; \{ X_{12} = 11 \} \rangle \]
\[ \langle 5; \{ X_{12} = 00 \} \rangle \]

and then proceeding.
<table>
<thead>
<tr>
<th>Top element</th>
<th>Expansion</th>
<th>New queue</th>
</tr>
</thead>
<tbody>
<tr>
<td>⟨28; {X_{12} = 01}⟩</td>
<td>\langle 287; {X_{12} = 01, X_{23} = 10} \rangle \langle 287; {X_{12} = 01, X_{23} = 11} \rangle</td>
<td>⟨28; {X_{12} = 01, X_{23} = 10}⟩ \langle 15; {X_{12} = 10}⟩ \langle 7; {X_{12} = 11}⟩ \langle 5; {X_{12} = 00}⟩ \langle 4; {X_{12} = 01, X_{23} = 11}⟩</td>
</tr>
<tr>
<td>⟨28; {X_{12} = 01, X_{23} = 10}⟩</td>
<td>(complete configuration 1)</td>
<td>⟨28; {X_{12} = 01, X_{23} = 10}⟩ \langle 15; {X_{12} = 10}⟩ \langle 7; {X_{12} = 11}⟩ \langle 5; {X_{12} = 00}⟩ \langle 4; {X_{12} = 01, X_{23} = 11}⟩</td>
</tr>
<tr>
<td>⟨15; {X_{12} = 10}⟩</td>
<td>\langle 15\frac{5}{7}; {X_{12} = 10, X_{23} = 00} \rangle \langle 15\frac{3}{7}; {X_{12} = 10, X_{23} = 01} \rangle</td>
<td>⟨15; {X_{12} = 10, X_{23} = 00}⟩ \langle 15; {X_{12} = 10}⟩ \langle 7; {X_{12} = 11}⟩ \langle 6; {X_{12} = 10, X_{23} = 01}⟩ \langle 4; {X_{12} = 01, X_{23} = 11}⟩</td>
</tr>
<tr>
<td>⟨15; {X_{12} = 10, X_{23} = 00}⟩</td>
<td>(complete configuration 2)</td>
<td>⟨15; {X_{12} = 10, X_{23} = 00}⟩ \langle 15; {X_{12} = 10}⟩ \langle 7; {X_{12} = 11}⟩ \langle 6; {X_{12} = 10, X_{23} = 01}⟩ \langle 5; {X_{12} = 00}⟩ \langle 4; {X_{12} = 01, X_{23} = 11}⟩</td>
</tr>
<tr>
<td>⟨7; {X_{12} = 11}⟩</td>
<td>\langle 7\frac{4}{7}; {X_{12} = 11, X_{23} = 10} \rangle \langle 7\frac{2}{7}; {X_{12} = 11, X_{23} = 11} \rangle</td>
<td>⟨7; {X_{12} = 11, X_{23} = 10}⟩ \langle 7; {X_{12} = 11}⟩ \langle 6; {X_{12} = 10, X_{23} = 01}⟩ \langle 5; {X_{12} = 00}⟩ \langle 4; {X_{12} = 01, X_{23} = 11}⟩ \langle 1; {X_{12} = 11, X_{23} = 11}⟩</td>
</tr>
<tr>
<td>⟨7; {X_{12} = 11, X_{23} = 10}⟩</td>
<td>(complete configuration 3)</td>
<td>⟨7; {X_{12} = 11, X_{23} = 10}⟩ \langle 7; {X_{12} = 11}⟩ \langle 6; {X_{12} = 10, X_{23} = 01}⟩ \langle 5; {X_{12} = 00}⟩ \langle 4; {X_{12} = 01, X_{23} = 11}⟩ \langle 1; {X_{12} = 11, X_{23} = 11}⟩</td>
</tr>
<tr>
<td>⟨6; {X_{12} = 10, X_{23} = 01}⟩</td>
<td>(complete configuration 4)</td>
<td>⟨6; {X_{12} = 10, X_{23} = 01}⟩ \langle 5; {X_{12} = 00}⟩ \langle 4; {X_{12} = 01, X_{23} = 11}⟩ \langle 1; {X_{12} = 11, X_{23} = 11}⟩</td>
</tr>
</tbody>
</table>

and so on.
6 Integer programming

Most integer programs (IP) can easily be recast into the graphical model framework. Given an integer program

\[
\begin{align*}
\text{variables} & \quad x_i \in [l_i, u_i] \cap \mathbb{Z} \\
\text{constraints} & \quad \sum_i A_{ji} x_i = b_j \\
\text{goal} & \quad \text{maximize } \sum_i w_i x_i
\end{align*}
\]

clearly one can convert to

\[
\begin{align*}
\Phi_i(x_i) & = w_i \\
\Phi_j(x) & = \begin{cases} 
0 & \text{if } \sum_i A_{ji} x_i = b_j \\
-\infty & \text{otherwise}
\end{cases} \\
\otimes & = + \\
\oplus & = \text{max}
\end{align*}
\]

In fact, integer program solvers often use graphical model algorithms like VE and the JT for cases where the sparsity of the constraints is such that there is an elimination order with small induced width.

The standard technique for solving integer programs when there is not a small induced width is to solve the linear relaxation (remove the integer constraint on the variables and then use LP algorithms to solve for fractional values in polynomial time) and either round the result or apply branch-and-bound techniques. These methods can be applied equally well to finding the MAP configuration of a graphical model, and form a natural complement to triangulation-based techniques. This can be done despite the fact that graphical model probabilities are not necessarily linear in the variables\(^2\), by basing the ILP on configurations of potential domains and separators (cliques and separators of the untriangulated graph):

\(^2\)That is, \(\log P(x) = \sum_i w_i x_i\) only for the case of potentials over singleton variables.
<table>
<thead>
<tr>
<th><strong>Graphical model</strong></th>
<th><strong>Integer program</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>clique $C$</td>
<td>$\forall c \in C$, variable $y_c \in {0, 1}$</td>
</tr>
<tr>
<td>separator $S$</td>
<td>$\forall s \in S$ variable $y_s \in {0, 1}$</td>
</tr>
<tr>
<td>potential $\Phi_C$</td>
<td>$\forall c \in C$, cost function term $w_c = \log \Phi_C(c)$</td>
</tr>
<tr>
<td>edge $C - S$</td>
<td>$\forall s \in S$ constraint $y_s = \sum_{c \sim s} y_c$</td>
</tr>
<tr>
<td>maximize $\prod_C \Phi_C(x)$</td>
<td>maximize $\sum_c w_c y_c$</td>
</tr>
</tbody>
</table>

While optimally solving ILPs is of course NP-complete, in some cases rounding the linear relaxation is an effective polynomial-time approximation algorithm. For example, it has been applied as an alternative to the sum-product algorithm for error-correcting-code decoding.